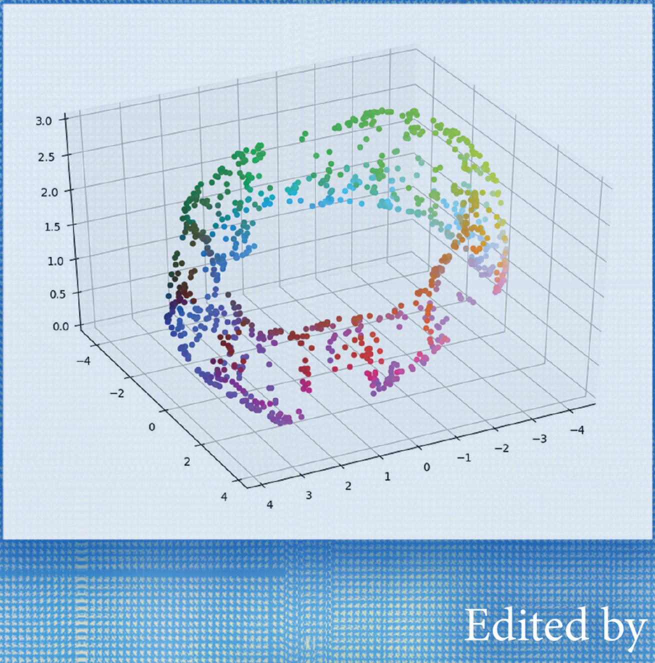
Chapman & Hall/CRC
Handbooks in Mathematics Series

Data Science for Mathematicians



Edited by
Nathan Carter

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Data Science for Mathematicians

Data Science for Mathematicians

Edited by
Nathan Carter

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

International Standard Book Number-13: 978-0-367-02705-6 (Hardback)
978-0-429-39829-2 (ebook)

Library of Congress Cataloging-in-Publication Data

Names: Carter, Nathan C., editor.
Title: Data science for mathematicians / Nathan Carter, ed.
Description: First edition. | Boca Raton, FL : CRC Press, 2020. | Includes bibliographical references and index. |
Contents: Programming with data / Sean Raleigh -- Linear algebra / Jeffery Leader -- Basic statistics / David White -- Clustering / Amy S. Wagaman -- Operations research / Alice Paul and Susan Martonosi -- Dimensionality reduction / Sofya Chepushtanova, Elin Farnell, Eric Kehoe, Michael Kirby, and Henry Kvinge -- Machine learning / Mahesh Agarwal, Nathan Carter, and David Oury -- Deep learning / Samuel S. Watson -- Topological data analysis / Henry Adams, Johnathan Bush, Joshua Mirth. |
Summary: "Mathematicians have skills that, if deepened in the right ways, would enable them to use data to answer questions important to them and others, and report those answers in compelling ways. Data science combines parts of mathematics, statistics, computer science. Gaining such power and the ability to teach has reinvigorated the careers of mathematicians. This handbook will assist mathematicians to better understand the opportunities presented by data science"-- Provided by publisher.
Identifiers: LCCN 2020011719 | ISBN 9780367027056 (hardback) | ISBN 9780367528492 (paperback) | ISBN 9780429398292 (ebook)
Subjects: LCSH: Mathematical analysis. | Mathematical statistics. | Data mining. | Big data--Mathematics.
Classification: LCC QA300 .D3416 2020 | DDC 515--dc23
LC record available at https://lccn.loc.gov/2020011719

Contents

Foreword

As we often hear, we live in an era where data can be collected, stored, and processed at an unprecedented (and rapidly accelerating) scale. Whether or not that happens in a way that can properly be called science, however, is a critical issue for our society.

The recently-concluded Roundtable on Data Science Postsecondary Education, held under the auspices of the Board on Mathematical Sciences and Analytics at the National Academies, brought together representatives from academia, professional societies, industry, and funding agencies to share perspectives on what tools our students need to participate in this space. Throughout these discussions, it was clear that, while there is no single model for what constitutes data science, there are a wide variety of tools from mathematics, statistics, and computer science that are essential ingredients for anyone interested in exploring this rapidly-evolving discipline.

For those of us in mathematics, it is essential that we become better informed about the role of our discipline in this emerging field. Doing so will help prepare our students for careers that will increasingly depend on some level of competency in understanding how to use data to inform decisions, regardless of the specific discipline or industry in which they find themselves. It can also bring heightened awareness of the importance of rigorous mathematical perspectives to the enterprise.

I believe it is an ethical and, in fact, an existential imperative for the mathematical sciences community to develop a deeper understanding of the role of our disciplines in data science and to change our educational programs to enable our students to engage with data effectively, and with integrity.

Nathan Carter and his colleagues have made an important contribution by providing an overview of many of the key tools from the mathematical, statistical, and computational sciences needed to succeed as a data scientist, written specifically for those of us in the mathematical sciences—faculty interested in learning more about data science themselves, graduate students, and others with a reasonable level of mathematical maturity. From my perspective, for those of us concerned with undergraduate mathematics education, it cannot have come too soon.

Michael Pearson
Executive Director, Mathematical Association of America

Chapter 1

Introduction

Nathan Carter

Bentley University

This chapter serves as an introduction to both this text and the field of data science in general. Its first few sections explain the book's purpose and context. Then Sections 1.4 and 1.5 explain which subjects will be covered and suggest how you should interact with them.

1.1 Who should read this book?

The job market continues to demand data scientists in fields as diverse as health care and music, marketing and defense, sports and academia. Practitioners in these fields have seen the value of evidence-based decision making and communication. Their demand for employees with those skills obligates the academy to train students for data science careers. Such an obligation is not unwelcome because data science has in common with academia a quest for answers.

Yet data science degree programs are quite new; very few faculty in the academy have a PhD in data science specifically. Thus the next generation of data scientists will be trained by faculty in closely related disciplines, primarily statistics, computer science, and mathematics.

Many faculty in those fields are teaching data-science-related courses now. Some do so because they like the material. Others want to try something new. Some want to use the related skills for consulting. Others just want to help their institution as it launches a new program or expands to meet increased demand. Three of my mathematician friends have had their recent careers shaped by a transition from pure mathematics to data science. Their stories serve as examples of this transition.

My friend David earned his PhD in category theory and was doing part-time teaching and part-time consulting using his computer skills. He landed a full-time teaching job at an institution that was soon to introduce graduate courses in data science. His consulting background and computing skills made him a natural choice for teaching some of those courses, which eventually led to curriculum development, a new job title, and grant writing. David is one of the authors of Chapter 8.

Another friend, Sam, completed a PhD in probability and began a post-doctoral position in that field. When his institution needed a new director of its data science masters program, his combination of mathematical background and programming skills made him a great internal candidate. Now in that role, his teaching, expository writing, and career as a whole are largely focused on data science. Sam is the author of Chapter 9.

The third and final friend I'll mention here, Mahesh, began his career as a number theorist and his research required him to pick up some programming expertise. Wanting to learn a bit more about computing, he saw data science as an exciting space in which to do so. Before long he was serving on a national committee about data science curricula and spending a sabbatical in a visiting position where he could make connections to data science academics and practitioners. Mahesh is the other author of Chapter 8.

These are just the three people closest to me who have made this transition. As you read this, stories of your own friends or colleagues may come to mind. Even if you don't know a mathematician-turned-data-scientist personally, most mathematicians are familiar with Cathy O'Neil from her famous book *Weapons of Math Destruction* [377], who left algebraic geometry to work in various applied positions, and has authored several books on data science.

In each of these stories, a pure mathematician with some computer experience made a significant change to their career by learning and doing data science, a transition that's so feasible because a mathematical background is excellent preparation for it. Eric Place[1] summarized the state of data science by saying, "There aren't any experts; it's just who's the fastest learner."

But mathematicians who want to follow a path like that of David, Sam, or Mahesh have had no straightforward way to get started. Those three friends cobbled together their own data science educations from books, websites, software tutorials, and self-imposed project work. This book is here so you don't have to do that, but can learn from their experiences and those of others. With a mathematical background and some computing experience, this book can to be your pathway to teaching in a data science program and considering research in the field.

[1]Director of Informatics, Martin's Point Health Care, speaking at the University of New Hampshire Northeast Big Data Conference, November 17, 2017

But the book does not exist solely for the benefit of its mathematician readers. Students of data science, as they learn its techniques and best practices, inevitably ask why those techniques work and how they became best practices. Mathematics is one of the disciplines best suited to answering that type of question, in data science or any other quantitative context. We are in the habit of demanding the highest standards of evidence and are not content to know just that a technique works or is widely accepted. Bringing that mindset to data science will give students those important "why" answers and make your teaching of data science more robust. If this book helps you shift or expand your career, it will not be for your benefit only, but for that of our students as well.

1.2 What is data science?

In 2001, William Cleveland published a paper in *International Statistical Review* [98] that named a new field, "Data Science: An Action Plan for Expanding the Technical Areas of the Field of Statistics." As the title suggests, he may not have been intending to name a new field, since he saw his proposal as an expansion of statistics, with roughly 15% of that expanded scope falling under the heading of computer science. Whether data science is a new field is a matter of some debate, as we'll see in Section 1.3, though I will sometimes refer to it as a field for the sake of convenience.

In *Doing Data Science* [427], Cathy O'Neil and Rachel Schutt say that the term "data scientist" wasn't coined until seven years after Cleveland's article, in 2008. The first people to use it were employees of Facebook and LinkedIn, tech companies where many of the newly-christened data scientists were employed.

To explain this new field, Drew Conway created perhaps the most famous and reused diagram in data science, a Venn diagram relating mathematics, statistics, computer science, and domain expertise [107]. Something very close to his original appears in Figure 1.1, but you're likely to encounter many variations on it, because each writer seems to create one to reflect their own preferences.

You can think of the three main circles of the diagram as three academic departments, computer science on the top left, math on the top right, and some other (usually quantitative) discipline on the bottom, one that wants to use mathematics and computing to answer some questions. Conway's top-left circle uses the word "hacking" instead of computer science, because only a small subset of data science work requires formal software engineering skills. In fact, reading data and computing answers from it sometimes involves unexpected and clever repurposing of data or tools, which the word "hacking" describes very well. And mathematicians, in particular, can take heart from

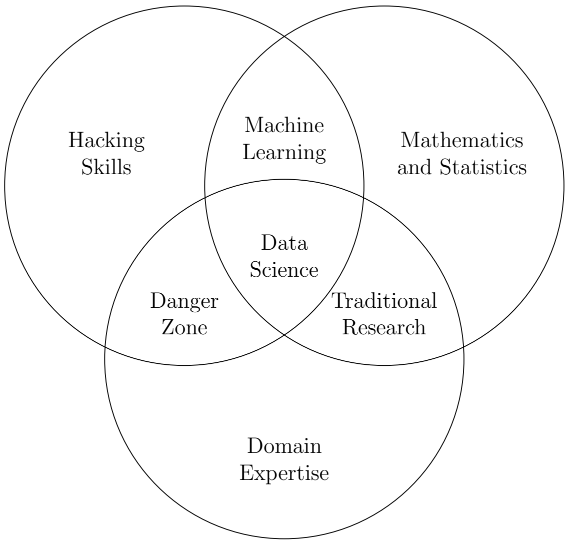


FIGURE 1.1: Rendering of Drew Conway's "data science Venn diagram" [107].

Conway's labeling of the top-right circle not as statistics, but *mathematics and* statistics, and for good reason. Though Cleveland argued for classifying data science as part of statistics, we will see in Section 1.4 that many areas of mathematics proper are deeply involved in today's data science work.

The lower-left intersection in the diagram is good news for readers of this text: it claims that data science done without knowledge of mathematics and statistics is a walk into danger. The premise of this text is that mathematicians have less to learn and can thus progress more quickly.

The top intersection is a bit harder to explain, and we will defer a full explanation until Chapter 8, on machine learning. But the gist is that machine learning differs from traditional mathematical modeling in that the analyst does not impose as much structure when using machine learning as he or she would when doing mathematical modeling, thus requiring less domain knowledge. Instead, the machine infers more of the structure on its own.

But Figure 1.1 merely outlines which disciplines come into play. The practice of data science proceeds something like the following.

1. **A question arises that could be answered with data.**

   This may come from the data scientist's employer or client, who needs the answer to make a strategic decision, or from the data scientist's own curiosity about the world, perhaps in science, politics, business, or some other area.

2. **The data scientist prepares to do an analysis.**

   This includes finding relevant data, understanding its origins and meaning, converting it to a useful format, cleaning it, and other necessary precursors to analysis of that data. This includes two famous acronyms, ETL (extract, transform, and load the data) and EDA (exploratory data analysis).

3. **The data scientist performs an analysis based on the data.**

   Though mathematical modeling is very common here, it is not the only kind of analysis. Any analysis must leverage only those mathematical, statistical, and computing tools permissible in the given situation, which means satisfying necessary mathematical assumptions, ensuring computations are feasible, complying with software license agreements, using data only in legal and ethical ways, and more.

4. **The data scientist reports any useful results.**

   If the question was posed by an employer, "useful" probably means actionable and desirable for that employer. If the question was a scientific or academic one, "useful" may mean results that satisfy the intellectual curiosity that motivated the question. A report of results may go to an employer, an academic journal, a conference, or just a blog. If no useful results were obtained, returning to an earlier step may be necessary.

Organizing the parts of data science chronologically is not uncommon (Figure 2 in reference [414]) but it is also not the only way to explain the subject (pages 22 and following in reference [132]). But every such sequence includes steps similar to those above, each of which is covered in this text, some in multiple ways.

Mathematicians may be prone to thinking of step 3 in the list above as the real heart of the subject, because it's where the most interesting mathematics typically happens. But step 2 can be even more central. Hadley Wickham's famous and influential paper "Tidy Data" [501] suggests that up to 80% of data analysis can be the process of cleaning and preparing data (quoting Dasu and Johnson [119]).

And while step 4 may seem like the task that follows all the real work, it, too, can be the heart of the matter. In his online data science course, Bill Howe[2] says that Nate Silver's seemingly miraculous predictions of the 2012 United States presidential election were (in Silver's own admission) not sophisticated data work but were important largely because they were new and were communicated through clear visualizations and explanations. In that famous work, perhaps step 4 was the most innovative and impactful one.

This leaves room for specialization in many directions. Those with better communication skills will be better at step 4, like Silver and his team. Those with greater subject matter expertise will be better at step 1, setting goals and asking questions. Those with better stats skills will be better at modeling; those with math skills will be better at knowing when models are applicable and when new ones need to be invented and proven valid. Those with software

[2]Associate Director, eScience Institute, University of Washington

engineering skills may be better at packaging algorithms for reuse by other scientists or creating dashboards for nontechnical users.

Names have begun to appear for some of these specializations. The term "data analyst" can refer to one who specializes in the statistics and modeling aspects of data science, the top-right circle of Figure 1.1. To refer to someone who specializes in one data-driven domain, such as finance or genetics, you may see the term "domain specialist." This carries the connotation of familiarity with general data science knowledge (of mathematics, statistics, and computing) but less expertise in those areas than in the chosen domain. This is the bottom circle of Figure 1.1. Finally, a "data engineer" is someone who specializes in the storage, organization, cleaning, security, and transformation of data, and will typically have strength primarily in computer science, perhaps databases specifically. This is somewhat related to the top left of Figure 1.1.

Though I've quoted some foundational sources when explaining data science, it's important to recognize that not everyone defines data science the same way. The field is still young, and there are many stakeholders and thus many opinions. We'll see one of the central disagreements in Section 1.3.

And there are different trends in how the phrase "data science" is used internationally. For instance, in the United States it has been more common to interpret Figure 1.1 as defining a data scientist very narrowly, as someone who has mastered all parts of the Venn diagram, and thus sits in the exalted center spot. The buzzword "unicorn" suggests how rare and magical such an individual is. Only a small percentage of jobs in the field require this level of expertise. In Europe, the interpretation has been more broad, permitting one to specialize in various parts of the diagram while having some working knowledge of the others.

1.3 Is data science new?

Donoho's famous survey "50 Years of Data Science" [132] covers a debate about whether data science is a qualitatively new thing or just a part of statistics. Much of the debate centers around changes in technology, which we can break into two categories. First, the proliferation of data-generating devices (sensors, mobile phones, websites) give us access to enormous amounts of data. As Bill Howe puts it, "data analysis has replaced data acquisition as the new bottleneck to discovery." Second, advances in hardware and software have made it possible to implement ideas that were once outside the capability of our computing tools (such as deep neural networks, covered in Chapter 9).

One side of the debate might point out that statistics didn't stop being statistics when computers were invented or when SAS or R became popular.

It's still statistics as technology evolves. The four steps in Section 1.2 might as well describe the work of a statistician.

The other side would point to the unprecedented scope and diversity of technology changes in so short a span of time, many of which necessitate whole new areas of expertise, some of which require very little work with statistics. Their claim would be that the sum of these changes amounts to a qualitatively new situation.

Regardless of whether we prefer either side of the debate, we should avoid thinking of these emerging and new technologies themselves as data science. Because the emergence of data science as a field is intimately intertwined with these technological changes, there is a temptation to conflate the two ideas. But data science is the use of these technologies, which includes all the parts of Figure 1.1, not just the technology itself.

Perhaps the most common example of this misunderstanding surrounds the buzzphrase "big data," which refers to data that is so extensive or is being generated so quickly that new hardware or software is required to manage it. Having to deal with big data may put technological constraints on the entire data science process (steps 1 through 4 in Section 1.2), but we should not confuse those technologies nor their constraints with the data science process itself. Data science is much more than big data, or any other aspect of the technology it uses.

And yet at the same time, there is something very important about the technology. One of the sea changes in recent years has been how many analyses are automated by writing code in a scripting language like R or Python rather than by manipulating buttons or menus in a piece of statistical software. Donoho says this is a "game changer," turning algorithms, which were once things described only in natural language in statistics papers, into things that are downloadable, actionable, and testable. This leads to their being shared more often on code websites than in academic papers.

Changes since Donoho's writing make the case even more strongly. Now you don't even have to download anything to run an algorithm you find online; you can read computational notebooks on GitHub or open them immediately in a host of cloud services, make your own copy, and begin hacking, reproducing other investigators' results or tweaking them to your own needs. Dashboards make it possible for a data scientist to automate an analysis and create an interface for it, through which nontechnical users can apply the analysis to new data as that data arrives. Those users can gain new insights without subsequent interaction with the analyst. Such applications are called "data products," and so data science has become not just about answering questions, but about building tools that can answer questions on demand.

This book takes no side in the debate over whether data science is a new field. But it should be clear that the capabilities and activities surrounding its new technologies are what a lot of the excitement is about.

1.4 What can I expect from this book?

You can expect that the chapters in this text represent the essentials of data science and that each chapter provides references to where you can dive deeper. Thus from this book you will get a solid foundation paired with ample opportunities to grow into whatever specialized expertise interests you.

You cannot expect that every corner of data science is represented in this text. It's too huge a field, and it's constantly evolving. For example, we do not have a chapter on big data. We hit related topics in Chapters 7 and 10, but big data is constantly changing with technology and is more related to computer science than mathematics, so it does not have its own chapter.

We motivate each chapter by stating its purpose at the outset. Why did someone invent the concepts in the chapter? What properties of the world necessitated that tool in the analytics toolbox?

In most chapters, we give only a brief review of the mathematical background required for a chapter's content, because the text is for mathematicians. We assume that you know most of the mathematical background, perhaps requiring only a short review or summary, which lets us move more quickly into the material that's likely to be new to you. In some chapters, this may include more advanced mathematics, but most of the time the new content is statistics, computing, or surprising applications of mathematics you already know.

Each chapter covers all the most common concepts within its topic so that you have a broad understanding of that topic. Most chapters also go beyond just the foundations, but cannot do so comprehensively.

Each chapter ends with citations for further reading, sometimes even on cutting edge research, so that those who were particularly intrigued by the material know where to go for more depth.

Most chapters do not have exercises, because we assume that the reader is capable of assessing whether they understand the chapter's basics, and assigning themself exercises to verify this if needed. Chapters typically give only large, project-sized assignments. I cannot emphasize enough that every reader should do these projects. Skipping them will leave only the illusion of understanding.

What I cannot create, I do not understand.

Richard Feynman
found on his chalkboard after his death

Each chapter has its own distinctives, but as a whole, they follow the logical progression described below.

Chapter 2, Programming with Data. While we assume that the reader knows a little programming, most mathematicians aren't up on the tools, practices, and ideas data scientists use when coding. Consider this chapter your fast track to the computing knowledge you'll need to get started, including computational notebooks, reproducibility, and best practices.

Chapter 3, Linear Algebra. Why does a text for mathematicians have a chapter on an undergraduate topic? It quickly reviews the basics to get into the most relevant new ideas for working with data. Organized around matrix decompositions, this chapter highlights computational issues such as data storage, sparse matrices, and numerical stability, culminating with unique projects on text analysis, databases, and web searching.

Chapter 4, Basic Statistics. The last time most mathematicians spent time studying statistics was probably before the data revolution. We need a review. I've wanted for a long time to read a chapter like this one, covering a huge swath of statistics fundamentals, but aimed at the level of a mathematician, and with examples using R. It also highlights tips for first-time statistics teachers, especially those whose students want to go into data science. Here you'll encounter the crucial concept of overfitting vs. underfitting a mathematical model, which reappears throughout the text.

Chapter 5, Cluster Analysis. Some aspects of computing with data today seem almost magical. This chapter shows the first examples of that through the most common type of unsupervised learning, with examples of automatically finding, in data, structure that is meaningful to humans. Readers may find this to be the first entirely new content in the text, and it is a great example of how data science brings many topics together, such as discrete mathematics, statistics, analysis, and optimization.

Chapter 6, Operations Research. Recall the mandate in data science that derived insights must have practical value. This chapter emphasizes that by expanding the optimization concepts from Chapter 5 to many more methods and applications. The authors distinguish descriptive, prescriptive, and predictive analytics, all in service to making decisions from data, and end with an impressive array of deep projects suitable for the reader or as term projects for advanced undergraduates.

Chapter 7, Dimensionality Reduction. Readers eager to dive into the mathematical foundations of data science will enjoy Chapter 7, touching topics as disparate as Grassmannian manifolds, group theory, and principal components analysis. While focusing primarily on theory, the chapter still finds time to provide specific examples (with Python and

MATLAB code) of finding low-dimensional patterns in high-dimensional data. The techniques and the mathematical theory upholding them are at times equally surprising.

Chapter 8, Machine Learning. When you think of data science, you probably think of the topics in this chapter, which surveys many common algorithmic methods and central tools like gradient descent and the scikit-learn library. Many common supervised learning techniques appear, including support vector machines, decision trees, and ensemble methods, as well as key workflow principles, like preventing leakage between training and test data.

Chapter 9, Deep Learning. Neural networks and deep learning have received a lot of hype, which this chapter addresses head-on in two ways. First, it introduces readers to many of the amazing capabilities of these technologies, and second, it reveals their implementation with clear exposition and beautiful illustrations. Topics include stochastic gradient descent, back-propagation, network architectures (CNNs, RNNs), and code to get started in Python.

Chapter 10, Topological Data Analysis. This relatively new area of data science is a hot topic not only for its successes in applied problems, but for the unexpected marriage of pure mathematics (topology) to the applied world of data. It reviews essential ideas from topology and connects them to data through the computation of persistent homology. The chapter contains over 20 exercises to draw the reader in and has extensive online supplements.

This diversity of chapter content and style provides something for every reader, and because chapters are largely independent, you can start reading in whichever one you like. Those who need to brush up on their computing skills may want to start with Chapter 2. Or if you're confident in your computing skills and want to see what you can do with them, try Chapter 8 or 9. To see new applications of familiar mathematics, try Chapter 3, 5, or 6; to learn some new mathematics, try Chapter 7 or 10. Those more interested in being a better teacher in data-related courses may want to start with Chapter 4.

Or, as the outline above shows, the text is also laid out so that it can be read in order. Ample cross-references between chapters let you know what's coming next, and where you can flip back to review a concept from earlier.

1.5 What will this book expect from me?

We assume you know some computer programming. One statistician I spoke with said that if he had to tell mathematicians one thing that would help them get on with data science, it would be programming in a general purpose language. You don't need excellent coding skills, but you need to have coded a bit before and to be willing to do plenty more. Chapter 2 will give you more specifics.

O'Neil and Schutt define a data scientist as "a data-savvy, quantitatively minded, coding-literate problem-solver." Mathematicians are quantitatively minded problem-solvers, and if you're also coding literate, you can read this book to become data-savvy. One of the main ways this will happen (and your coding strength expand) is by tackling the large projects at the end of each chapter.

You need to know or be willing to learn those domains to which you want to apply data science. Bill Franks[3] said that regardless of the mathematical and statistical methods underpinning an analysis, the goal is always the same: to turn data into actionable value. INFORMS defines analytics [163] as "the scientific process of transforming data into insight for making better decisions." If you don't know any domain of application for data science, you'll need to be willing to learn one, or you won't be able to assess which insights are actionable or desirable.[4]

But of course you already have expertise in at least one field to which data science can be applied, perhaps many. Do you like golf? Video games? Politics? Mountain biking? Quilting? Investments? Rock music? There is data analytics to be done in every field and more data coming online every minute. Indeed, even if your life were exclusively within academia, there is data analytics to be done there; bibliometrics uses data science to study academic publications. One research group at my institution published bibliometric analyses in the Proceedings of the National Academy of Sciences [174].

The final thing this book expects of you is perhaps the biggest. It's a mental shift you'll need to cultivate as you move out of the realm of theorems and proofs. Rob Gould[5] emphasized to me how challenging it can be to **shift from deductive to inductive reasoning.** Mathematicians spend their whole lives removing context, seeking abstraction, but in data science, context is crucial. Data takes on meaning only in context.

Jennifer Priestley[6] echoed that sentiment when she and I discussed this book's goals. She says that this book should help you learn how to put together things you already know—mathematical concepts like graph theory—to do things you don't yet know, like analyze supply chain data or health care problems. The connection from what you know to what you don't is data. For

[3]Chief Analytics Officer, International Institute for Analytics, speaking at the University of New Hampshire Northeast Big Data Conference, November 17, 2017

[4]While this is less true when the data scientist is part of a group whose other members are domain experts, at least some of that expertise is necessarily absorbed by the data scientist in order for the group to do its work.

[5]Statistician, University of California, Los Angeles, and founding editor of *Teaching Innovations in Statistics Education*

[6]Professor of Statistics and Data Science, Associate Dean of the Graduate College, Kennesaw State University

example, we're no longer thinking about graph theory in the abstract, but we're looking at the specific graph given by the data we have.

The data scientist's job is to apply to the available data appropriate mathematical and statistical theory, usually using computer code, in a way that yields useful results. Mathematicians have tons of expertise in the concepts but not typically in how to apply them. This book is here to fill that gap.

Chapter 2

Programming with Data

Sean Raleigh

Westminster College

2.1 Introduction

The most important tool in the data science tool belt is the computer. No amount of statistical or mathematical knowledge will help you analyze data if you cannot store, load, and process data using technology.

Although this chapter is titled "Programming with Data," it will not teach you how to write computer programs. There are lots of books and online resources that can teach you that. Neither is the goal to familiarize you with any particular programming language; such an attempt would be out-of-date practically from the moment the book left the presses.

Instead, the aim of this chapter is to introduce you to some aspects of computing and computer programming that are important for data science applications.

A theme that runs throughout this chapter is the idea of reproducible research. A project that can be reproduced is one that bundles together the raw data along with all the code used to take that data through the entire pipeline of loading, processing, cleaning, transforming, exploring, summarizing, visualizing, and analyzing it. Given the exact same hardware and software configuration, and given access to the raw data, anybody should be able to run the code and generate the exact same results as the original researchers. Several sections below speak to tools and workflows that support the data scientist in creating and documenting their work to make it more reproducible.

Each of the following topics could fill whole books. We'll just go on a quick tour, introducing key vocabulary and giving some big-picture perspective.

2.2 The computing environment

2.2.1 Hardware

Although the focus of this chapter is on software, it's worth spending a little time talking about the machinery that powers the processes we run.

The choice of hardware for doing data science depends heavily on the task at hand. Modern desktop and laptop computers off the shelf are generally well equipped with large hard drives and a reasonable amount of RAM ("Random-Access Memory," the kind of memory used to run the processes that are happening in your current computing session). Memory is cheap.

One common definition of big data is any data that is too big to fit in the memory your computer has. For big data situations there are several options. Your employer may have larger servers that allow more data to be stored and allocate more memory resources for processing big data, and perhaps you can access those resources. Another popular option is to use a cloud-based service. Some are offered by "big tech" companies like Google [195], Amazon [17], and Microsoft [345], but there are many companies out there that host data and data processing tools. These services are convenient because they make vast resources available to you, but typically only charge you for the resources you actually use on a pay-as-you-go basis.

When possible, it's nice to take advantage of parallel processing. The central processing unit (CPU) is the brain of your computer. It's responsible for managing all the instructions that are sent to the various parts of your computer and for doing the heavy lifting when computations need to be performed. Most modern CPUs have multiple "cores" that operate somewhat independently of each other. This allows for complex computations to be broken up into pieces that can run on different cores simultaneously. There's a cost-benefit analysis for parallelizing code. For example, you might run a one-off task overnight rather than spending time figuring out how to split the process into pieces that can run on parallel cores, perhaps for only a few hours of time savings. Nevertheless, when you have an especially time-consuming bit of code (and perhaps one you plan to run repeatedly or regularly), it's worth finding out whether your machine has multiple cores and whether the functions you use can be parallelized to leverage those cores.

Some data scientists are working with graphics processing units (GPUs), the part of your computer responsible for rendering graphics. GPUs are, in some ways, far more powerful than CPUs. However, GPUs are highly specialized and not designed for the kinds of use cases data scientists typically have. Working with them is more demanding from a technical point of view. Unless your work involves image processing, frameworks and interfaces for doing data science with GPUs are not widely available; they might be out of reach for the average user. (The situation is improving, though, so by the time you read this, such frameworks may be more commonplace; one such framework is discussed in Chapter 9.)

2.2.2 The command line

Back in the days when computing was done on large mainframes, the user interface was a simple "terminal." This was usually a box that didn't have much functionality on its own but simply provided a way for the user to enter commands that were passed to the mainframe, where all the serious computational work was done. In the early days of personal computing, the mainframe/terminal distinction went away for the average home computer user, but the user interacted with their machine in much the same way, typing commands on a screen next to a prompt that just sat there waiting for the next command.

A lot of serious computing is still done at the command line. If you have to access resources on a server, your computer can act as a terminal sending commands to that server. Some functionality on even our own machines is available principally through some kind of terminal program with a command line. (For example, Git [468]—described in Section 2.2.6 below—is one such application.)

If you want to check the terminal on your machine, there are multiple ways of doing it (depending on the operating system), but here are some easy ones:

- In Windows 10, go to the Start Menu and type `cmd` to open the Command Prompt.
- In MacOS, use Command + Space to open Spotlight Search and type `Terminal`. It's located in the Utilities folder in your Applications list. It's also easy to find through the Finder.
- For Linux users, Ubuntu and Mint use the keyboard shortcut Ctrl + Alt + T to open the terminal. It's also easy to find through Applications or Apps (depending on your installation).

One important application of command-line skills is the Linux operating system. While most home computers (not counting mobile devices) use Windows or MacOS as an operating system, Linux is the *lingua franca* for web servers, Internet-of-things devices, and serious production machines. Linux users can opt to use a graphical user interface (GUI), pointing and clicking a mouse to execute commands instead of typing something at a command prompt, but power users frequently work at the command line.

2.2.3 Programming languages

There are hundreds of programming languages in existence and users of those languages are often passionate proponents of their preferences. As new languages develop and old ones fall out of favor, there is a constantly shifting landscape, especially in a field like data science that is growing and evolving so rapidly.

Having said all that, at the time of the writing of this book, there are two languages that stand out for their popularity among data scientists: R [397] and Python [164].

It is often said that R was written by statisticians and for statisticians. Because of that focus, R has a lot of built-in functionality for data visualization and analysis.

Python is a general-purpose programming language that was designed to emphasize code readability and simplicity. While not originally built for data science applications per se, various libraries augment Python's native capabilities: for example, `pandas` [340] for storing and manipulating tabular data, `NumPy` [370] for efficient arrays, `SciPy` [255] for scientific programming, and `scikit-learn` [385] for machine learning.

Both R and Python have features that make them attractive to data scientists. One key feature is "dynamic typing." By way of contrast, "static typing" requires the programmer to declare up front any variables they intend to use along with the type of data that will be stored in those variables. R and Python figure out what kind of variable you want on the fly by looking at the content of the data you assign to it at the time of its definition. This makes the process of writing programs faster, although it can also make programs more error-prone. Rapid prototyping is much easier in R and Python than, say, in C or Java.

Both R and Python are open source, which means that their source code is made freely available, and consequently the languages are also free to download and use. Open source projects also come with licenses that allow the user to copy, remix, and redistribute the source code freely as well.

These are obviously not the only two choices. Julia [49] is a relative newcomer that shows a lot of promise in speed, functionality, and readability for some types of mathematical work. Java [196] is a traditional workhorse that many developers already know (and love?). Scala [366] is closely related to Java and acts as a standard interface for a variety of big data tools. A few Internet searches will reveal whatever is the new hotness at the time you read this.

2.2.4 Integrated development environments (IDEs)

Computer code is just text. You can write code in any text editor, and it will be usable in R, Python, or whatever language you're working in. Programming languages generally provide a basic "shell," which usually takes the form of a simple application with a command prompt you can use to issue commands to run functions or programs in that language. The shell implements something called a REPL, short for Read-Evaluate-Print Loop, an apt description of its purpose.

Having said that, because programming is hard, we like to find ways to make it easier. So rather than writing text files and running them from a shell, we can run an Integrated Development Environment (IDE). IDEs often provide tools like the following.

- Syntax highlighting makes it easier to read code on the screen by adding color to distinguish functions, arguments, and variables.
- Linters clean up code by indenting and formatting it according to style guides, flagging deprecated code, and checking for syntax errors or unused variables and function arguments.
- Debugging tools provide ways to trace through functions as they're executing and set breakpoints that pause the program to check the state of variables while the program is running.
- Project management allows you to organize files with related code and store needed data files and other dependencies alongside your code.

- Code completion watches as you type and suggests functions and variable names as you start to type them. Then you can hit a shortcut key (such as tab) to auto-complete the rest of the name before you finish typing, saving many keystrokes.

- Version control is often integrated so that you don't have to open up a separate terminal and remember complicated command-line syntax. (See Section 2.2.6.)

The de facto standard IDE for R is RStudio [413] due to its tight integration with R and its orientation toward data projects. There are several IDEs available that are specifically designed for Python, like Spyder [109] and PyCharm [251]. Both languages are also supported in a wide range of general-purpose IDEs that work with many different languages, like Atom [189], Sublime [449], Eclipse [460], and Visual Studio Code [346].

2.2.5 Notebooks

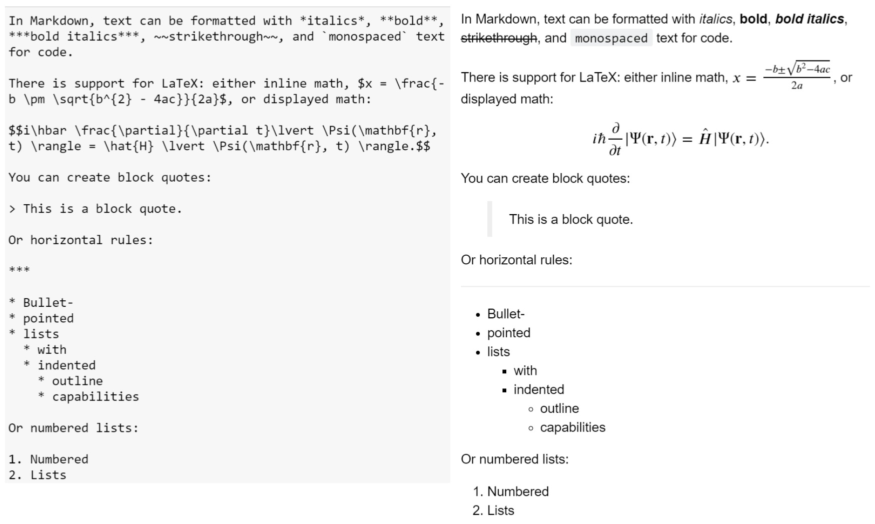
In 1984, Donald Knuth (of TEX fame) wrote about a new paradigm for organizing code called "literate programming" [272]:

> Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

Traditional code appears in plain text files—sometimes called "script files"—containing just the code and, we hope, some comments. Knuth suggests reversing that balance, using a document that is primarily narrative to explain the function and purpose of the code, interspersed with delimited blocks containing the code itself.

One popular, modern implementation of the literate programming paradigm is the "notebook." (Software familiar to many mathematicians like Mathematica, MATLAB, Maple, and Sage also use notebooks.) Many flavors of notebook use Markdown for formatting text, which is actually a markup system similar in function to markup in TEX, but much simpler and with fewer options.[1] (See Figure 2.1 for an example of Markdown in both plain text and rendered forms.) A notebook author indicates which portions of the content are code, and that code is executed when the notebook is "run." But the more common use of notebooks is running each block of code interactively, one at a time, typically while the author is creating the file, possibly still exploring how to solve a problem or analyze a dataset.

[1]Because markup documents often get processed to produce either HTML or PDF output files, HTML tags and LATEX commands, respectively, can also be inserted when plain vanilla Markdown is insufficient for the job.



```
In Markdown, text can be formatted with *italics*, **bold**,
***bold italics***, ~~strikethrough~~, and `monospaced` text
for code.

There is support for LaTeX: either inline math, $x = \frac{-
b \pm \sqrt{b^{2} - 4ac}}{2a}$, or displayed math:

$$i\hbar \frac{\partial}{\partial t}\lvert \Psi(\mathbf{r},
t) \rangle = \hat{H} \lvert \Psi(\mathbf{r}, t) \rangle.$$

You can create block quotes:

> This is a block quote.

Or horizontal rules:

***

* Bullet-
* pointed
* lists
  * with
  * indented
    * outline
    * capabilities

Or numbered lists:

1. Numbered
2. Lists
```

In Markdown, text can be formatted with *italics*, **bold**, ***bold italics***, ~~strikethrough~~, and `monospaced` text for code.

There is support for LaTeX: either inline math, $x = \frac{-b \pm \sqrt{b^{2} - 4ac}}{2a}$, or displayed math:

$$i\hbar \frac{\partial}{\partial t}\lvert \Psi(\mathbf{r}, t) \rangle = \hat{H} \lvert \Psi(\mathbf{r}, t) \rangle.$$

You can create block quotes:

> This is a block quote.

Or horizontal rules:

- Bullet-
- pointed
- lists
  - with
  - indented
    - outline
    - capabilities

Or numbered lists:

1. Numbered
2. Lists

FIGURE 2.1: Left: Some plain text using Markdown syntax. Right: The rendered text. Markdown has many more features than those sampled here.

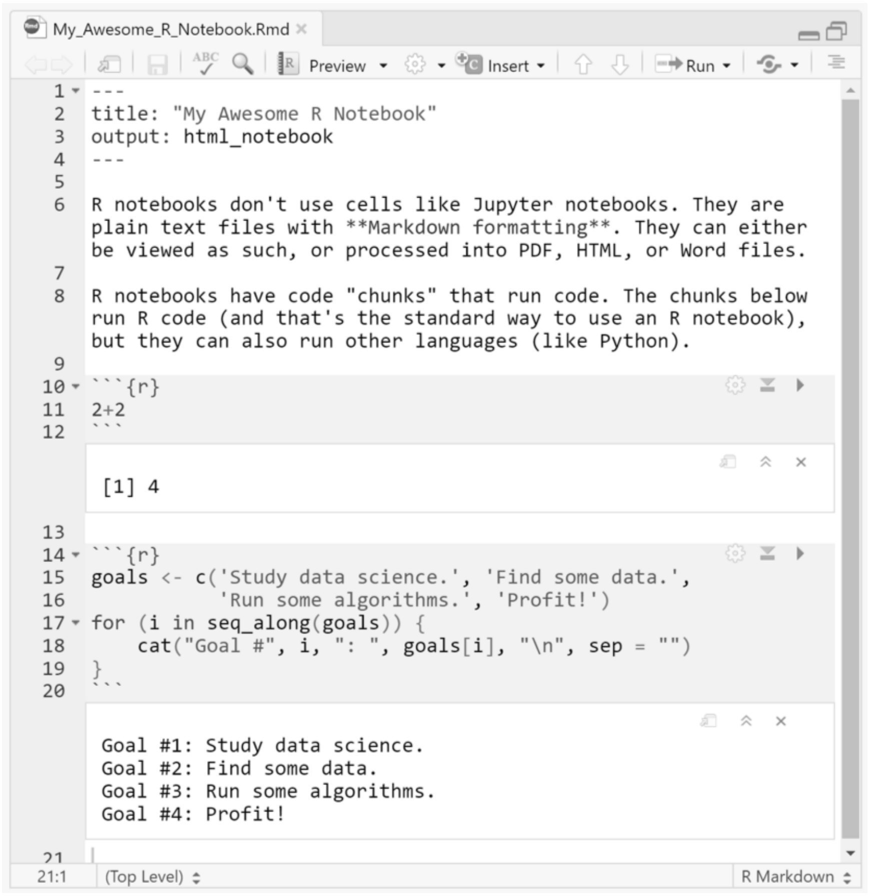
R notebooks are Markdown files that use a special set of delimiters called "code chunks" to indicate where R code goes. Figure 2.2 shows an R Notebook as it appears in RStudio. Figure 2.3 shows the HTML generated from the notebook file. R Notebooks can also be processed to other file types, like PDF and Word.

The standard notebook format for Python is the Jupyter [271] notebook (formerly called IPython) that uses a series of "cells," each of which can contain either code or Markdown-formatted text. See Figure 2.4 for an example.

While not strictly necessary for doing reproducible research, notebooks are helpful because they allow their creators to explain each step of the pipeline and motivate the workflow. Instead of requiring a separate file of instructions for compiling and processing code, the notebook file *is* the set of instructions, containing all the text and code necessary to reproduce and explain the results.

Notebooks are especially valuable in educational settings. Rather than having two documents—one containing code and the other explaining the code, usually requiring awkward references to line numbers in a different file, notebooks allow students to see code embedded in narrative explanations that appear right before and after the code. While it's possible in script files to embed large commented sections alongside the code, those comment sections are difficult to format and have no way to take advantage of the pretty, easy-on-the-eyes, rich-text output that Markdown provides, such as typesetting of mathematical formulas in TEX notation.

Notebooks play an important role in data science, but they are not solutions to every problem. In software development, "production code"—meaning



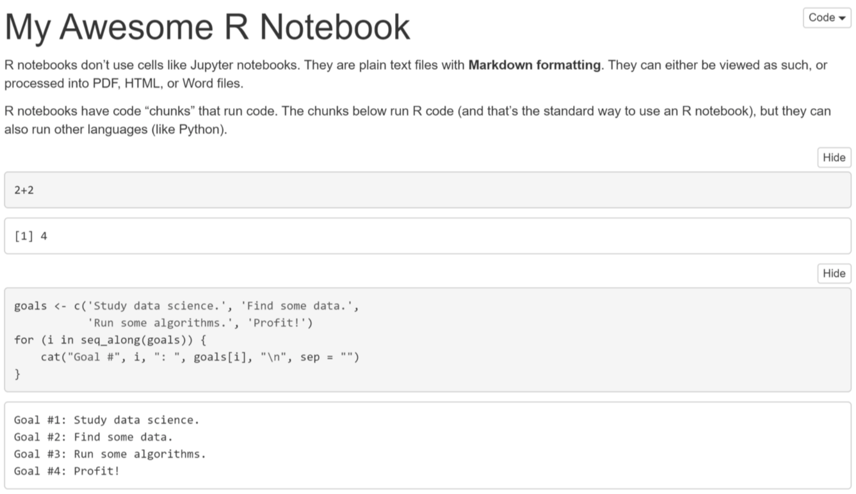
~~~
---
title: "My Awesome R Notebook"
output: html_notebook
---

R notebooks don't use cells like Jupyter notebooks. They are
plain text files with **Markdown formatting**. They can either
be viewed as such, or processed into PDF, HTML, or Word files.

R notebooks have code "chunks" that run code. The chunks below
run R code (and that's the standard way to use an R notebook),
but they can also run other languages (like Python).

```{r}
2+2
```
~~~

```
[1] 4
```

~~~
```{r}
goals <- c('Study data science.', 'Find some data.',
           'Run some algorithms.', 'Profit!')
for (i in seq_along(goals)) {
    cat("Goal #", i, ": ", goals[i], "\n", sep = "")
}
```
~~~

```
Goal #1: Study data science.
Goal #2: Find some data.
Goal #3: Run some algorithms.
Goal #4: Profit!
```

FIGURE 2.2: An example of a simple R notebook presented in RStudio [413]. R notebooks will also generate HTML files. Figure 2.3 shows the file generated from the notebook in this figure.



My Awesome R Notebook

R notebooks don't use cells like Jupyter notebooks. They are plain text files with **Markdown formatting**. They can either be viewed as such, or processed into PDF, HTML, or Word files.

R notebooks have code "chunks" that run code. The chunks below run R code (and that's the standard way to use an R notebook), but they can also run other languages (like Python).

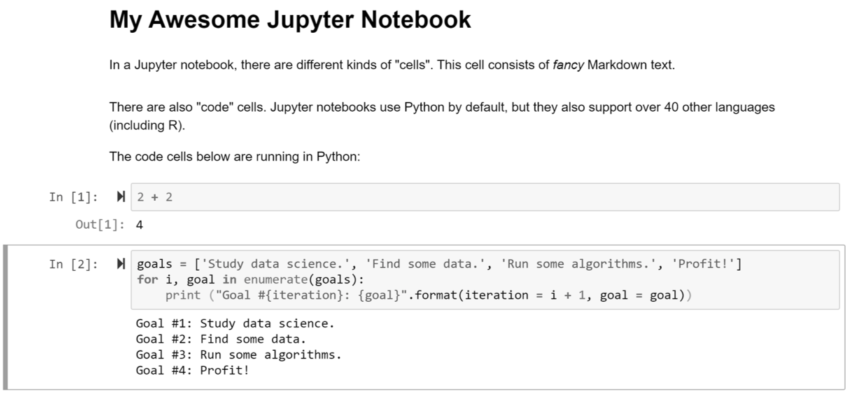
```
2+2
```

```
[1] 4
```

```
goals <- c('Study data science.', 'Find some data.',
           'Run some algorithms.', 'Profit!')
for (i in seq_along(goals)) {
    cat("Goal #", i, ": ", goals[i], "\n", sep = "")
}
```

```
Goal #1: Study data science.
Goal #2: Find some data.
Goal #3: Run some algorithms.
Goal #4: Profit!
```

FIGURE 2.3: This is the HTML file generated by the R Markdown file from Figure 2.2.



My Awesome Jupyter Notebook

In a Jupyter notebook, there are different kinds of "cells". This cell consists of *fancy* Markdown text.

There are also "code" cells. Jupyter notebooks use Python by default, but they also support over 40 other languages (including R).

The code cells below are running in Python:

```
In [1]:  2 + 2
Out[1]: 4
```

```
In [2]:  goals = ['Study data science.', 'Find some data.', 'Run some algorithms.', 'Profit!']
         for i, goal in enumerate(goals):
             print ("Goal #{iteration}: {goal}".format(iteration = i + 1, goal = goal))

         Goal #1: Study data science.
         Goal #2: Find some data.
         Goal #3: Run some algorithms.
         Goal #4: Profit!
```

FIGURE 2.4: An example of a simple Jupyter notebook. The Markdown cells can be edited by double-clicking them and working with the plain text Markdown. The code cells run Python code. The grey box indicates the currently active cell.

the final version of code that is used to, say, run a business—needs to work silently in the background. It would be foolish to spend computing resources generating pretty text that nobody will read as it's running on a server. Having said that, though, there could be a notebook (or multiple notebooks) that explain the code that is later extracted and put into production.

2.2.6 Version control

Without version control, when editing a file, you have two choices:

1. Save your edits using the same filename, overwriting your old work.
2. Save your edits using a different filename, creating a new file.

In the former case, you have only one file, but no way to recover an older version of that file. In the latter case, you have some older versions of the file saved, but now you have a proliferation of files lying about.

A version control system will track your files in a "repository" (or "repo" for short) by recording only the differences between each version and the next. As a result, the version control system can recover the state of the project at any point in the past by simply "replaying" the sequence of changes starting from the creation of the file up to the point you wish to recover. You have to keep only one copy of your files with a little extra overhead for storing the "diffs," or changes along the way.

Without version control, when working on a file collaboratively with someone else, you have two choices:

1. Edit the file one person at a time, sending it back and forth.
2. Edit two copies of the file simultaneously, and try to manually reconcile the changes after comparing both copies.

Most types of modern version control systems are distributed version control systems, which can solve this problem as well. In the distributed paradigm, anyone who needs to work on a set of files will copy the entire repository to their local machine. After making edits, they push those changes back to a central server, and the software will take care of merging all the changes together. In the event that two people have edited the exact same lines of text in two different ways, a human will have to resolve the merge conflict to decide which of the versions should be accepted. But as long as people are working in different parts of the file, the system will perform merges silently and correctly. The distributed version control system also solves the problem of backing up your files; if either your machine or the server goes down, the repository can be restored easily from the other.

There are a number of version controls systems out there, but Git has become somewhat of a de facto standard. Learning Git for the first time can be intimidating due to all the specialized verbs that have evolved around version control: clone, fork, commit, push, pull, fetch, checkout, merge, squash, rebase, etc. These commands are usually run from a command line (unless you use an IDE with version control support, as in Section 2.2.4), and the user has to learn many of them, as well as the various options that can be passed to those commands. Even seasoned Git users often have to look up the syntax for various commands on a regular basis.

There are a number of websites that host Git repositories in the cloud (e.g., GitHub, GitLab, Bitbucket). In addition to serving a critical role as the central location for projects involving multiple users, these services also play an important role in reproducible research. Assuming you store your files in a publicly available repo, others can see and reproduce your work. In fact, GitHub will render Jupyter Notebooks right in the browser (as opposed to just printing the plain text code from the underlying file). If your work has an appropriate open source license, others will be able to download a copy of your work and build on it, modify it for their own work, or take it in different directions. Not only does version control give access to your code, it gives access to every past version of your code as well. If someone wanted to run an analysis based on the way your code ran three years ago, they could actually download the exact version of your code from three years ago to get started.

2.3 Best practices

While this chapter is not meant to make anyone an expert programmer, there are some basic principles that experienced coders understand that make their work more functional and robust. This section explains a few of the most important ones.

2.3.1 Write readable code

> "Always code as if the [person] who ends up maintaining your code will be a violent psychopath who knows where you live." — John F. Woods

One of the benefits of R and Python is that their code is relatively easy to understand when you read it. Even still, there are steps you can take as the programmer to make your code even easier to read.

Most people know about the importance of indenting code in a systematic way. In Python, it's actually built in to the language itself—rather than using braces or some other kind of punctuation to enclose the body of functions or control statements (like `if/then/else` statements or `for` or `while` loops), Python uses white space:

```
for i in range(10):
    if i % 2 == 0:
        print("{number} is even".format(number = i))
    else:
        print("{number} is odd".format(number = i))
```

Variables should be named in sensible, comprehensible ways. It's probably okay to use something like `i`, `j`, or `k` as a simple index in a `for` loop, but other variables should be named to indicate what they are supposed to store:

```
last_name = "Raleigh"
total_count = previous_count + new_count
```

One helpful convention for naming functions or methods is to use verbs that communicate the "action" of the function. The following code defines a Python function to find the divisors of a number:

```
def find_divisors(n):
    div_list = []
    for i in range(n):
        if n % (i + 1) == 0:
            div_list.append(i + 1)
    return div_list
```

If a function returns a logical value, it might be named as follows:

```
def are_equal(x, y):
    if x == y:
        return True
    else:
        return False
```

Mathematicians are accustomed to writing out long, multiline expressions. This is much harder to do with code and the results are difficult to read and debug. Consider the following Python code, which computes the sample standard deviation of a list of numbers, using the standard mathematical formula converted directly into Python code. (It assumes that a statistics module has been imported with the name `stats`.)

```
def calculate_sample_variance_short(data):
    return sum([(data_value - stats.mean(data))**2
        for data_value in data])/(len(data) - 1)
```

A much better approach is to keep expressions simple by first defining simple subexpressions on their own lines. Following that, you can define the main expression as a simple formula using the previously defined subexpressions. So, for example, note the improved readability in the next Python example, which accomplishes the same task. While the former is more concise, it's also less readable than the second and less clear about the sequence of steps involved in the calculation.

```
def calculate_sample_variance_readable(data):
    data_mean = stats.mean(data)
    n = len(data)
    squared_deviations = [(data_value - data_mean)**2
                          for data_value in data]
    return sum(squared_deviations)/(n - 1)
```

Avoid hard-coding numbers and parameter values into formulas and functions, especially if you plan to use those values in more than one place in the code. It's much better to use variables to store values. As long as a variable is named well, it will be more clear when you use it *how* you are using it (rather than just having a number mysteriously appear). And if you ever need to change the value, you can change it in one prominent place toward the top of your code (rather than hunting down all the places you use it and possibly missing a few). For example, the following R code calculates the mean of 10 randomly sampled values from a uniform distribution on $[0, 1]$ (`runif` stands for "random uniform") and then repeats that process 100 times. (See Section 2.3.3 for more information about the `set.seed` command.)

```
set.seed(42)
simulated_data <- vector(length = 100)
for(i in 1:100) {
  simulated_data[i] <- mean(runif(10, 0, 1))
}
```

A more flexible way to do this—and one that will usually result in fewer bugs—is to give names to various parameters and assign them values once at the top:

```
set.seed(42)
reps <- 100
n <- 10
lower <- 0
upper <- 1
simulated_data <- vector(length = reps)
for(i in 1:reps) {
  simulated_data[i] <- mean(runif(n, lower, upper))
}
```

Now you can run any number of simulations using any sample size and any interval simply by changing the values in one easy-to-find spot. This format also makes it easier to convert any code into a reusable function later by turning the list of variables into a list of parameters for that function.

While programming languages vary in how "readable" they are based on the idiosyncrasies of their syntax, the above tips should make any code more clear, both to other people reading your code, and to yourself in the future.

2.3.2 Don't repeat yourself

Suppose we have a dataset about colleges and universities containing a `Tuition` column recorded using the characters "$" and "," (like $45,000). We won't be able to compute summary statistics until these values are stored as proper numbers (like 45000). We could use the following Python code for a one-off transformation.

```
data["Tuition"] = data["Tuition"].str.replace("$","")
data["Tuition"] = data["Tuition"].str.replace(",","")
data["Tuition"] = data["Tuition"].astype(float)
```

Then suppose that we encounter more columns in the same dataset with the same problem, columns such as `Room_Board` and `Fees`. It's tempting to take the code for the first iteration and copy and paste it multiple times for the subsequent repetitions of the task. There are many tasks like this in programming that have to be repeated multiple times, perhaps with minor variations.

An important principle goes by the acronym "DRY," for Don't Repeat Yourself. While cutting and pasting seems like a time-saving strategy, it usually ends up being a suboptimal solution in the long run. For example, what if you decide later that you need to tweak your code? You will have to make the same tweak in every other spot you copied that code. If you miss even one spot, you're likely to introduce bugs.

A much better solution is to write a function that accomplishes the task in more generality. Then, every time you need to repeat the task, you can just call the function you wrote. Adding arguments to the function allows you to introduce flexibility so that you can change the parameters of the function each time you wish to run it.

For example, given the utility of the task performed by the code above, it's not much more effort to write a little function that does the same thing, but for any column of any dataset:

```
def convert_currency_to_float(data, col):
    data[col] = data[col].str.replace("$","")
    data[col] = data[col].str.replace(",","")
    data[col] = data[col].astype(float)
    return data
```

It is a little more time-consuming to write a general function that depends on flexible parameters than to write a single chunk of code that executes a concrete task. If you only have to do the task, say, one more time, there is a cost/benefit scenario you must consider. However, even when you only repeat a task once, there are benefits to abstracting your task and distilling it into a function. If nothing else, you will improve your programming skills. The abilities you develop will be transferable to future tasks you want to automate. But you might also be surprised how often you later decide that you do need to repeat a task more than the one time you originally planned. Abstracting tasks into functions ultimately makes your code more readable; rather than seeing the guts of a function repeated throughout a computation, we see them defined and named once, and then that descriptive name repeated throughout the computation, which makes the meaning of the code much more obvious.

2.3.3 Set seeds for random processes

Data scientists often have the need to invoke random processes. Many machine learning algorithms incorporate randomness, for example, when setting initial weights in a neural network (Chapter 9) or selecting a subset of features in a random forest (Chapter 8). Validation techniques like bootstrapping and cross-validation generate random subsets of the data (Chapter 4). Often data scientists will set up simulations to prototype or test their algorithms.

Using random number generation and stochastic processes seems counter to the notion of reproducible research. How can someone reproduce a result that was generated at random?

The solution is to realize that the random numbers generated by our computers are not really random, but are "pseudorandom." Roughly speaking, a sequence of numbers is pseudorandom if it has approximately the same statistical properties as a sequence of truly random numbers. Computers generate pseudorandom numbers using an algorithm, and each algorithm starts with an initial condition known as a "seed." Two computers running the same pseudorandom-number-generating algorithm starting with the same seed will produce identical sequences of pseudorandom values.

Therefore, the way to make your work reproducible is to set a specific random number seed in your code. Modulo some technical issues (like running parallel processes—are the same seeds passed to each processor, for example?), any other person who later runs your code will then generate the same output, accomplishing the goal of reproducibility.

2.3.4 Profile, benchmark, and optimize judiciously

When you run your code, there are tools that can record the amount of memory used and the time it takes to execute each step of your code. This is called profiling your code. This is closely related to benchmarking, which is comparing performance, either that of your code on two or more different machines, or that of two or more pieces of code on the same machine, depending on your goals. The purpose of profiling and benchmarking is generally to understand the bottlenecks in your workflow, whether caused by hardware or software. These bottlenecks can often be relieved by rewriting code, perhaps using different algorithms or data structures, or even using different programming languages.

When rapidly prototyping code, especially for one-off applications, it's generally not that important to spend a lot of time in "premature optimization." But for tasks that are clearly taking too long and need to be run more than once, it is often valuable to try to get some insight into why your code might be slower than it needs to be.

It's hard to give specific advice about how to speed up your code because it's very context-dependent. In general, it's worth doing some research to find existing libraries or packages that implement slow portions of your code in faster ways; if a function is slow for you, it was likely slow for someone in the past who took the opportunity to try to make it better. Both R and Python also allow you to call subroutines in other languages like C and Fortran that are "closer to the metal," meaning that they are lower-level languages that have less overhead. The `Rcpp` [140] package, allowing C++ code inside of R, is extremely popular. There are several ways to get performance benefits in Python from C. One is Cython [27], which takes existing code in Python and translates it into C automatically. Another is the `cffi` [406] library that allows the user to use C code inside of Python.

It's often possible to achieve orders of magnitude in speed and/or memory improvement by changing the way you store your data and process it. For example, in R, the `data.table` [133] package imports and transforms data much more quickly than tools that access standard data frames (at the cost of using syntax that doesn't look like the rest of R). In Python, using `numpy` arrays for matrix operations will generally be faster than using Python's native matrix operations. These are just a few examples; doing some Internet research for existing packages is typically extremely valuable before beginning a new project.

2.3.5 Test your code

How do you know you can trust the output of your code?

A *Science* magazine news piece [347] describes a "nightmare" scenario involving a young researcher, Geoffrey Chang, studying protein structures in cell membranes:

> When he investigated, Chang was horrified to discover that a homemade data-analysis program had flipped two columns of data, inverting the electron-density map from which his team had derived the final protein structure. Unfortunately, his group had used the program to analyze data for other proteins. As a result, [...] Chang and his colleagues retract three *Science* papers and report that two papers in other journals also contain erroneous structures.

These types of errors, while not malicious in intent, can have devastating ramifications—imagine having to retract five papers!

It's not just about the output of code either. How do you know your functions are robust against a variety of inputs (often supplied by other users whose intentions are hard to guess)?

The process of writing code that tests other pieces of code is called "unit testing." A unit test takes a small piece of code, usually a single function, and runs it with a variety of inputs, comparing the results to the expected output to ensure that there is agreement. For example, using the `testthat` [500] package in R, we might write a unit test to see if a function does the right thing with a few sample values:

```
test_that("test_parity prints correctly", {
    expect_that(test_parity(-3), prints_text("odd"))
    expect_that(test_parity(0), prints_text("even"))
    expect_that(test_parity(6), prints_text("even"))
})
```

Although this example is rather trivial because the parity of an integer is easy to calculate, the same ideas can be used to run many more tests on the behavior of functions with an arbitrary level of complexity, where a correct implementation is far less certain and needs much more verification.

As with profiling and benchmarking, unit testing does take time to implement and may not be that important for one-off tasks that are simple and have a clear and direct implementation. But as your code grows in complexity, and especially as you start designing your code for other users, you should consider learning how to build in unit tests. Especially if scientific results depend on the output of your work, doing otherwise may very well be irresponsible.

2.3.6 Don't rely on black boxes

As data science has become a hot career path, there are many people who want to move into the field. Some work hard to learn new skills in making such a transition. However, there is a problematic perception out there that all it takes to be a data scientist is to learn a few new lines of code that can be thrown blindly at data to produce insight and, most importantly, profit!

The key problem is that there is no single algorithm that performs best under all circumstances. Every data problem presents unique challenges. A good data scientist will be familiar not only with a variety of algorithms, but also with the circumstances under which those algorithms are appropriate. They need to understand any assumptions or conditions that must apply. They have to know how to interpret the output and ascertain to what degree the results are reliable and valid. Many of the chapters of this book are specifically designed to help the reader avoid some of the pitfalls of using the wrong algorithms at the wrong times.

All this suggests that a data scientist needs to know quite a bit about the inner workings of the algorithms they use. It's very easy to use sophisticated libraries and functions as "black boxes" that take in input and generate output. It's much harder to look under the hood to know what's really happening when the gears are turning.

It may not be necessary in all cases to scrutinize every line of code that implements an algorithm. But it is worthwhile to find a paper that explains the main ideas and theory behind it. By virtue of their training, mathematicians are in a great position to read technical literature and make sense of it.

Another suggestion for using algorithms appropriately is to use some fake data—perhaps data that is simulated to have certain properties—to test algorithms that are new to you. That way you can check that the algorithms generate the results you expect in a more controlled environment.

We live in an era in which most statistical processes and machine learning algorithms have already been coded by someone else. We should enjoy the benefits of being able to build and test models rapidly. But those benefits also come with the responsibility of learning enough to know that our models are trustworthy.

2.4 Data-centric coding

There are a few programming concepts and computational skills that are somewhat specific to data science. In this section, we'll explore a few of these.

2.4.1 Obtaining data

Data exists in a variety of forms. When we obtain data, we have to work with the data as it comes to us, which is rarely in an ideal format.

2.4.1.1 Files

The most widely used format for data is a spreadsheet. Each spreadsheet application has its own proprietary file format, and these formats can be quite complicated. For example, a Microsoft Excel file is actually a zipped folder containing a number of XML files that keep track of not just the data, but all sorts of metadata about the file's structure, formatting, settings, themes, etc. (XML stands for eXtensible Markup Language and uses tags, like HTML does. It's a fun exercise to rename a Microsoft Excel file to have a `.zip` extension, unzip it, and explore the underlying XML files.) Importing data from Excel spreadsheets is computationally complicated, but since it's a central task to data science, there are great libraries and packages that do it seamlessly, like the `xlsx` [135] or `readxl` [504] packages in R or the `read_excel` function in `pandas` in Python. (There are many packages in both languages that provide this functionality.)

Even easier, if you can open the spreadsheet in Excel, you can export it in a plain text format that's easier to parse. (Plain text file formats leave less room for extra formatting that could cause files to be imported in a corrupted state.) For example, a Comma-Separated Values (CSV) file has column entries separated by commas with newlines to indicate where new rows begin. A Tab-

```
lastname,state,customerID
Behunin,UT,12248
Martinez,ID,12249
Fitzsimmmons,CO,12250
Alexander,UT,12251
```

```
lastname  state customerID
Behunin UT  12248
Martinez  ID  12249
Fitzsimmmons  CO  12250
Alexander UT  12251
```

```
lastname      state customerID
Behunin       UT    12248
Martinez      ID    12249
Fitzsimmmons  CO    12250
Alexander     UT    12251
```

FIGURE 2.5: The same file shown in three different plain text formats: CSV (left), TSV (center), and fixed-width (right) with fields of size 13, 6, and 10.

Separated Values (TSV) file is the same idea, but with tabs instead of commas. There are also "fixed-width" formats that store the data values padded with whitespace so that each one occupies the same fixed number of characters in the file. (See Figure 2.5.)

2.4.1.2 The web

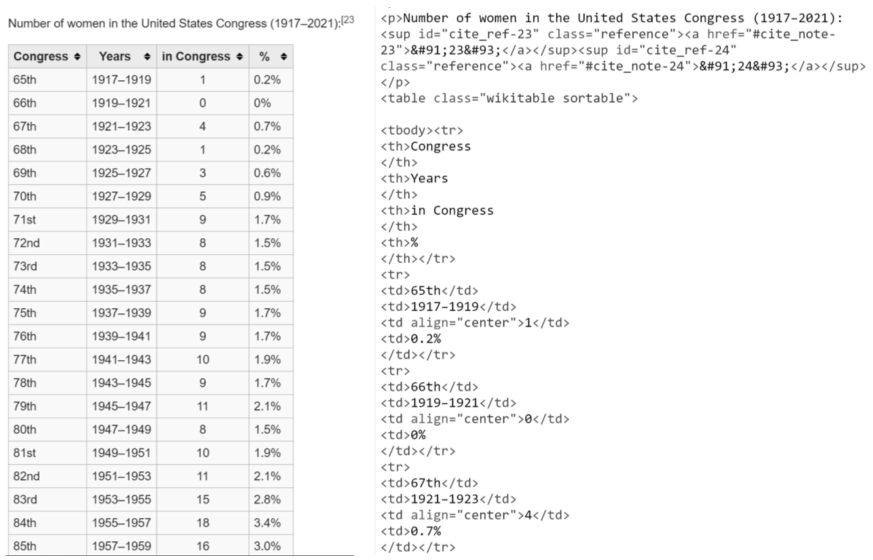
Some data is located on the web, but not in the form of a downloadable spreadsheet or CSV file. In an ideal world, websites would have a public-facing Application Programming Interface (API). Such an interface allows users to send requests to the website to retrieve—or in some cases, even submit—data. You can visit apitester.com and use their example API to learn more about how API calls work.

Even without an API, an organization might provide a website that simply lists data on a web page, say, in an HTML table. Or maybe the data you want isn't even formatted in a convenient way, but just appears on a web page, like in Figure 2.6.

These situations call for "web scraping." A web scraping tool will save the raw HTML text from the web page and provide utilities to select which elements you want based on the HTML tags that form the structure of the page. Assuming the designer of the web page applied some kind of consistent formatting to the data in which you're interested, it's usually not too difficult to extract only what you need from the page. The go-to web scraping tool in Python is Beautiful Soup [405]. In R, there are several options, but the `rvest` [503] package was designed to have similar functionality to Beautiful Soup.

2.4.1.3 Databases

Often, data is stored in a database. A relational database is a set of tables (each made up of rows and columns) that are connected together by relationships that identify when the values of one table are related to the values of another table. For example, a database for an online vendor might have customer information stored in one table and orders in another table. The link between these tables would be the customer IDs that appear in both tables. Relational database systems have strong constraints in place that protect the integrity of the data as it's accessed. Relational data is usually processed using



Number of women in the United States Congress (1917–2021):[23

Congress	Years	in Congress	%
65th	1917–1919	1	0.2%
66th	1919–1921	0	0%
67th	1921–1923	4	0.7%
68th	1923–1925	1	0.2%
69th	1925–1927	3	0.6%
70th	1927–1929	5	0.9%
71st	1929–1931	9	1.7%
72nd	1931–1933	8	1.5%
73rd	1933–1935	8	1.5%
74th	1935–1937	8	1.5%
75th	1937–1939	9	1.7%
76th	1939–1941	9	1.7%
77th	1941–1943	10	1.9%
78th	1943–1945	9	1.7%
79th	1945–1947	11	2.1%
80th	1947–1949	8	1.5%
81st	1949–1951	10	1.9%
82nd	1951–1953	11	2.1%
83rd	1953–1955	15	2.8%
84th	1955–1957	18	3.4%
85th	1957–1959	16	3.0%

```
<p>Number of women in the United States Congress (1917–2021):
<sup id="cite_ref-23" class="reference"><a href="#cite_note-
23">&#91;23&#93;</a></sup><sup id="cite_ref-24"
class="reference"><a href="#cite_note-24">&#91;24&#93;</a></sup>
</p>
<table class="wikitable sortable">

<tbody><tr>
<th>Congress
</th>
<th>Years
</th>
<th>in Congress
</th>
<th>%
</th></tr>
<tr>
<td>65th</td>
<td>1917–1919</td>
<td align="center">1</td>
<td>0.2%
</td></tr>
<tr>
<td>66th</td>
<td>1919–1921</td>
<td align="center">0</td>
<td>0%
</td></tr>
<tr>
<td>67th</td>
<td>1921–1923</td>
<td align="center">4</td>
<td>0.7%
</td></tr>
```

FIGURE 2.6: Left: Part of a table showing the number of women serving in the United States Congress by year from Wikipedia [506]. Right: Some of the HTML that generates the table on the left. Web scraping tools are able to extract the data from the web table and convert it to an appropriate data structure that you can work with.

SQL, short for Structured Query Language.[2] SQL queries are relatively easy to learn because of the natural way they are constructed. For example, the following query lists customers who are over 21 years old from the state of Utah (assuming the information is stored in a table called `customers` with columns `age` and `state`).

```
SELECT * FROM customers
    WHERE age > 21 AND state = "UT";
```

The `*` means "all columns." It gets a little more complicated when you have to aggregate data across multiple tables (called a "join"), but it's still not a very steep learning curve compared to most programming languages.

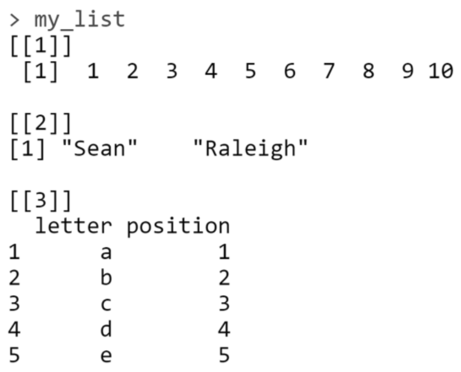
Much of the data out there that is considered "big data" is non-relational. Some of this has to do with performance limitations that relational databases can suffer once databases hit a certain size or complexity. A lot of it is related to the types of data being stored (e.g., video or audio) that relational databases weren't designed to store and access easily. There are a wide variety of non-relational databases that store data in non-tabular formats to try to overcome some of the limitations of relational data. Each type of database has its own systems for querying or summarizing the data depending on its architecture and application. Some of these non-relational databases are called NoSQL databases (which is a little misleading, because NoSQL doesn't necessarily imply that you can't use SQL to query them). NoSQL databases use a variety of systems to store data, including key-value pairs, document stores, and graphs.

2.4.1.4 Other sources and concerns

Worst-case scenarios include data that is embedded in a PDF file, data that appears in some kind of image file (like a photograph), or handwritten data. Sometimes we can leverage sophisticated optical character recognition (OCR) software to convert the data to a computer-friendly format, but that's not always the case. Manual data entry is always an option, albeit a painful and error-prone one.

There are ethical considerations to take into account when obtaining data. Some data is proprietary, and you may need to secure permission or possibly even pay for the rights to use it. Just because you can find a free copy online somewhere doesn't guarantee you the right to use it. Web scraping, in particular, lies in a somewhat murky territory legally and ethically; although the data is clearly publicly accessible, the terms of service for some websites explicitly prohibit web scraping. Even when web scraping is allowed, there are best practices you should follow to ensure that you are not bombarding servers with high-volume, high-frequency requests. Many websites maintain a `robots.txt` file (usually found by appending `/robots.txt` to the base URL)

[2]Wars are fought over the pronunciation of SQL: some say the individual letters "ess-cue-ell" and some say the word "sequel."



```
> my_list
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
[1] "Sean"    "Raleigh"

[[3]]
  letter position
1      a        1
2      b        2
3      c        3
4      d        4
5      e        5
```

FIGURE 2.7: A list in R with three elements: a numerical vector of length ten, a character vector of length two, and a data frame with five observations and two variables.

that explicitly lists the permissions granted to automated tools to access certain files or directories.

2.4.2 Data structures

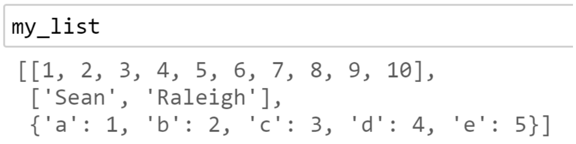
Once we have imported the data, the computer must store that data in memory in a way that makes sense to our statistical software and to us as programmers and analysts.

Every programming language has a number of "types," or ways that the computer stores each kind of data. For example, integer is a type, as are floating-point or double-precision numbers (discussed further in Section 3.4.1). Character (including numeric or non-numeric symbols) is a type. Programming languages also typically have arrays that are collections of objects of one type.

In R, the most generic data structure is a "list," which can contain named elements of any type (including other lists) and can be of any length. (See Figure 2.7.) For data science, we often store data in a "data frame," which is a special kind of list in which each element is a vector of some type, all vectors have the same length, and represent columns of data of different types, allowing the user to store both numerical and categorical data.

In Python, there are two data structures that are roughly similar to R's lists. One is also called a list, and it's just an unnamed, ordered sequence of objects of any type. (See Figure 2.8.) The other is called a dictionary, which consists of key-value pairs, in much the same way that mathematicians represent functions as sets of ordered pairs.

But Python also has "tuples" (ordered sequences like lists, but immutable, meaning they can't be modified after their creation) and sets (unordered collections).



```
my_list
```

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
 ['Sean', 'Raleigh'],
 {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}]
```

FIGURE 2.8: A list in Python with three elements: a list of ten numbers, a list of two strings, and a dictionary with five key-value pairs.

While there are no data frames native to Python, we can use the `pandas` library that was built to handle tabular data. Each column (of one specific type) is called a Series, and a collection of Series is called a DataFrame.

For data that is completely numeric, matrices (and their higher dimensional analogues, sometimes called arrays or tensors) are often a good way to store it. Processing matrices is easy due to advanced linear algebra libraries that make matrix operations very efficient. R has matrix manipulation built right into the language. Python has the `numpy` library that defines array-like structures like matrices. We will see this in much greater detail in Chapter 3.

2.4.3 Cleaning data

When we obtain data, it's almost never in a form that is suitable for doing immediate analysis. But many data analysis routines expect that we have "tidy" data. Although the ideas around tidy data have been around for a long time,[3] Hadley Wickham coined the term [501] and characterized it using three properties:

1. Each set of related observations forms a table.
2. Each row in a table represents an observation.
3. Each column in a table represents a variable.

For example, tidy data for a retail store might have a table for customer data. Each customer occupies one row of the dataset with their characteristics (variables like name and phone number) stored in different columns. Another table might hold product info with a separate product on each row, each with a product ID, a price, and a product description as possible columns. Yet another might be available inventory, and that might simply consist of two columns, one with a product ID (from the product info table) and one with the number of that item left in stock.

Even data from relational databases—seemingly tidy by definition—is tidy only if the database designer follows correct database design principles and only if the data is queried properly.

[3]Being "tidy" is closely related to Codd's "third normal form" from database theory.

Often, the most time-consuming task in the data pipeline is tidying the data, also called cleaning, wrangling, munging, or transforming. Every dataset comes with its own unique data-cleaning challenges, but there are a few common problems one can look for.

2.4.3.1 Missing data

One of the biggest problems for data analysis is missing data.

How is missing data recorded? For example, do blank cells in a spreadsheet get converted to null values when importing them into a statistical platform? In R, there are multiple representations of null values, including `NA`, `NaN`, and `NULL`. In Python, missing data might be called `None` or `NaN`. Note that there's a difference between the text string "None" and the special Python value `None`. What if those blank spreadsheet cells are not actually blank, but have some kind of invisible white space character like a space or tab? When those values are imported, they will be stored as text and not as a missing value as expected. Some datasets use special codes for missing data; for example, suppose data entry staff represented missing values as 999 and the analyst isn't aware of that. Now imagine taking the mean of a bunch of values where several are 999, and you can see the type of trouble such coding schemes can cause.

What is the cause of missing values? Do they represent people who left something blank on a form? Did they leave it blank because they didn't want to answer, or because they ran out of time or lost interest? Are the missing values data entry errors? Are the missing values artifacts of the import process? For example, sometimes when importing a messy spreadsheet, there will be extra blank (or nearly-blank) rows or columns you didn't expect, often caused by stray characters in cells far from away the rectangular data, or by formulas someone inserted off to the side.

What do we do with missing values? It depends heavily on the context of our data. For example, missing values for sales probably mean no sales, so they could be replaced with zeros. Missing values for patient height and weight definitely don't mean zero height and zero weight, and we'll likely have to leave them blank because there's no way we will be able to know what the original values were. Missing values for one year in a time series probably don't mean zero, but it might make sense to try to interpolate between two adjacent years assuming that local trends roughly follow a functional form for which interpolation makes sense.

Even when you have grappled with the context and representation of missing values, how do they affect your analysis? Are these values missing at random, or are missing values correlated with some other property of the population you're studying? For example, if you ask about sensitive topics in a survey, are certain marginalized groups more likely to leave that question blank? If that is the case, then by ignoring the missing data, your analysis will be biased. There is a large body of research on the technique of "imputation," which is the act of filling in missing values with educated guesses at various levels of sophistication. In some cases, if certain assumptions are met, imputing missing data can result in more correct analyses than just dropping rows for which there is missing data.[4]

2.4.3.2 Data values

Another central issue is how values are stored. For example, a dataset may have a `name` column in which full names are stored. What if you needed to sort the data by last name? You would first need to find a function (or write one yourself) that split the full name column into multiple columns, maybe `first_name` and `last_name`. But what about people with middle names or initials? That would require a `middle_name` column, but now we have to be careful when only two names are listed. What about Hispanic individuals who often use two last names? This is not a trivial problem. And if names are hard, imagine parsing a variable containing dates!

Categorical variables cause a number of problems. For example, a rigorously designed online form should generally present users with a drop-down box that limits the possible responses. But many web forms use free response boxes instead. So you may have users from the state of "Utah," "utah," "UT," "Ut," "ut," or even "Utha." The data cleaning process requires the data scientist to identify this issue and then re-code all these values into one consistently spelled response. (And you have to do it for every state!)

It's worth considering the level of specificity for your data-cleaning functions. For example, if you find an instance of "Ut" in an R data frame that you want to change to "UT," you could just note that it appears in the 12th row and the 4th column and fix it with code like the following one-liner.

```
df[12, 4] <- "UT"
```

But if there are other instances of "Ut" in the data, it would make a lot more sense to write code to fix every instance of "Ut." In R, that code looks like the following.

```
df$state[df$state == "Ut"] <- "UT"
```

Going a step further, the following code uses the `toupper` function to convert all state names to uppercase first, which would also fix any instances of "ut."

```
df$state[toupper(df$state) == "UT"] <- "UT"
```

Another problem is the representation of categorical variables in your data. Many datasets use integers as codes for categorical data. For example, a color variable might have values "red," "blue," and "green" that are recorded as 1, 2, and 3. Using those integer codes in real-valued functions would generally

[4]Horton and Kleinman wrote a nice article that surveys some common imputation methods along with computational tools that implement them. [232]

not be appropriate; nevertheless, you will see the mean of that color variable if you generate summary statistics for your data. Clearly, that mean has no reasonable interpretation.

You also need to know how your statistical platform deals with such data. R has a string data type, but it also has a special data type called "factor." Factor variables represent categorical data efficiently by using numbers under the hood, but they present human-friendly labels in most output and summarize such data in appropriate ways (e.g., frequency tables that simply count the instances in each category). Python (through the `pandas` library) has a similar "category" data type.

Before running any analysis, it's critical to inspect the values in your data, both the data types for each variable and the specific values recorded therein.

2.4.3.3 Outliers

A key problem for numerical variables is the presence of outliers. As with missing data, outliers have to be evaluated in the context of the data. It is important to take time to try to understand why these outliers exist and what they say about the data and the data collection process.

Sometimes outliers represent real, actual data values that just happen to be unusual. For example, in salary data, there might be a value that's several orders of magnitude higher than all other salaries. But if that value corresponds to a professional athlete, it might be accurate. On the other hand, if one person's weight is orders of magnitude higher than all others, that value must be an error.

If you can ascertain that outliers are truly errors, then it makes sense to correct them (when possible) or delete them. For all other outliers, it's usually *not* okay just to delete them, sweeping the issue under the rug. If you do have a good rationale for setting aside certain data points, you should be transparent about it. For example, the pro athlete whose salary is way higher than everyone else's might not make it into the final analysis, not because it's an invalid or erroneous value, but because that person may not be representative of the population you are trying to study. Your exclusion criteria should be clearly communicated.

2.4.3.4 Other issues

Even when the data is technically tidy, there may still be formatting issues to deal with. For example, you may need to "reshape" your data to be either "long" or "wide." One form is not necessarily better than the other, but certain algorithms expect their inputs to be either long or wide. For example, Figure 2.9 shows the same data represented both ways. If one considers all the test scores to be the same kind of measurement, then the long form on the right is likely more tidy, treating each combination of student and test as an observation. On the other hand, if one considers the student to be the observational unit with the three test scores as variables, then the wide form

student	test1	test2	test3
<fctr>	<dbl>	<dbl>	<dbl>
A	72	75	69
B	90	92	98

2 rows

student	test	score
<fctr>	<chr>	<dbl>
A	test1	72
A	test2	75
A	test3	69
B	test1	90
B	test2	92
B	test3	98

6 rows

FIGURE 2.9: Wide data (left) and long data (right).

observation	color
<fctr>	<fctr>
A	Red
B	Red
C	Blue
D	Green
E	Red
F	Green

6 rows

observation	Red	Blue	Green
<fctr>	<dbl>	<dbl>	<dbl>
A	1	0	0
B	1	0	0
C	0	1	0
D	0	0	1
E	1	0	0
F	0	0	1

6 rows

observation	Blue	Green
<fctr>	<dbl>	<dbl>
A	0	0
B	0	0
C	1	0
D	0	1
E	0	0
F	0	1

6 rows

FIGURE 2.10: The data frame on the left has a categorical variable called `color`. The data frame in the center is the one-hot encoded version; the three levels of color have been turned into three binary variables with a 1 indicating which color corresponds to each observation. The data frame on the right uses dummy encoding which recognizes that only two binary columns are necessary to uniquely identify each observation's color; observations that are "Red" will have a 0 for "Blue" and a 0 for "Green."

on the left might be more helpful, and indeed may be the required format for certain "repeated measures" or "time series" procedures.

Another formatting issue arises with categorical variables. Many statistical procedures require you to convert categorical variables to a sequence of numerical "indicator" variables using various encoding techniques with names like "one hot" and "dummy." (See Figure 2.10. Section 4.6.5 goes into more detail.)

One final note about data cleaning and formatting: the gold standard is to make the process reproducible. If there are only one or two small changes to make, it seems simple enough just to open the data in a spreadsheet program and make those changes. However, the sequence of mouse clicks and key presses required to do this will not be written down anywhere. It requires more work and discipline to write computer code to make those changes. And if there are many changes to make (like re-coding thousands or even millions of rows), you will likely have no choice but to write code that automates the process. Either way, cleaning your data through code will allow others to start with similar raw data files and convert them to correspondingly cleaned data files in the same way in the future by reusing your code. If you are in a situation where you will continue to receive future data files that use the same (messy) format, it will be to your long-term advantage to write functions that will also work with that future data.

2.4.4 Exploratory data analysis (EDA)

Once the data is clean, it's very common and typically very useful to visualize the data before jumping into analysis. For one thing, exploring your data visually helps you learn about your data, its features and idiosyncrasies. More importantly, the statistical techniques you choose are predicated on certain mathematical assumptions that ensure that the output of the analysis can be trusted. Checking conditions often involves looking at graphs, tables, and charts.

For example, some forms of statistical inference require that numerical values of interest be normally distributed in the population. Of course, all we have is sample data, and no real-world data is going to be shaped like a perfectly symmetric bell curve. Nevertheless, a histogram or a QQ-plot (as in Section 4.3.4) can suggest that our data is more or less distributed as if it were drawn from a normally distributed population, and that can give us some confidence that our statistical techniques will result in correct results.

There are many ways of visualizing data: frequency and contingency tables, bar graphs, boxplots, scatterplots, density plots, heatmaps, and many others. It's important to note that not every graph type is appropriate in every situation. In fact, for any given combination of variables, there are usually only a handful of table, graph, or chart types that make sense. If you're an educator teaching data science, this is an important point to impress upon students, who often wish to use the most visually unique or stunning display without considering whether it is a good way to communicate the quantitative concept they need to communicate.

There are three main ways people make plots in R. Base graphics (commands built into the core R functionality) are very flexible due to the fine control the user has over each element of the graph. As a result, though, it has a steep learning curve and requires a lot of work to make graphs look pretty. The `lattice` [420] graphics package uses a much easier interface, but is not as commonly used as the newer `ggplot2` [502] package.

Python also provides a number of common packages to produce graphs. The `matplotlib` [240] library is somewhat analogous to base R graphics: hard to use, but with much more flexibility and fine control. Other popular and easier-to-use options are `seaborn` [492] and `Bokeh` [56]. You can also use the `ggplot` [296] package, implemented to emulate R's `ggplot2`.

Outside of R and Python, there are also a number of commercial software products that specialize in data visualization. Tableau [453] is common, especially in the business world, and provides a sophisticated GUI for visual EDA and reporting. Many R and Python users export their final results to Tableau for sharing with clients, because of Tableau's superior interface and style.

2.5 Getting help

Learning any new skill—a new programming language or even a completely new field of study—can feel daunting. Fortunately, there are many ways of getting help. The Internet is your best friend, because many data science topics, especially about programming, are so new and change so quickly that books and journals are not sufficiently current. Search engines often provide dozens of links to any question you may have. Some of the top hits for any kind of technical search lead to the helpful Stack Overflow website. It's been around long enough that your question has likely been asked and answered there already, but if not, you can always post a new question. Do be sure to conduct a thorough search, possibly using several wording variations of your question, to make sure you aren't duplicating a question already present. Also note that some answers are for older versions of a language, like Python 2 versus Python 3. Even in those cases, users will often helpfully add additional answers for newer versions, so check the date of the original question and answer and scroll down to see possibly newer responses as well.

For deeper dives, you can search for blog posts, online books, and online courses. Of course there are also physical books you can purchase, although for some technical topics, these tend to go out of date somewhat quickly.

Sometimes there is no substitute for face-to-face interactions with like-minded people. Many cities have user groups for R, Python, and any number of other languages and data science topics. Data scientists—especially those in industry—often meet in formal and informal ways to discuss their craft. The largest conference for R users is "rstudio::conf" (sponsored by RStudio). Python users can attend "PyCon" meetings in many countries all over the world. While those are the more prominent conferences, there are loads of smaller, local conferences being held all the time. The website meetup.com might list gatherings close to you.

2.6 Conclusion

There is a lot of content in this chapter, covering a wide range of topics. Depending on where you are in your data science journey, not all of the information here will be immediately applicable. Hopefully, you can refer back as needed and revisit pieces that become more relevant in your future work.

If you are brand new to programming—especially in a data science context—here are some recommendations for getting started:

- Choose a language and get it installed on your machine. If you know the kind of data science work you're likely to do in the future, you might spend some time on the Internet to find out which language might be right for you. (But honestly, it doesn't matter that much unless you have an employer with an extremely specific demand.)
- Install an IDE that is known to work seamlessly with your language of choice. Consider working in a notebook environment.
- Find a project to start working on. (Tips below.)
- As you're working on the project, review some of the sections of this chapter to see if there are places you can apply best practices. For example, once you have code that works, check to see if you can make it better by following the tips in Section 2.3. Install Git, create an account on a cloud platform like GitHub, and start tracking your project using version control and sharing it publicly online.

Here are two ideas for interesting data projects:

- Try some web scraping. Find a web page that interests you, preferably one with some cool data presented in a tabular format. (Be sure to check that it's okay to scrape that site. "Open" projects like Wikipedia are safe places to start.) Find a tutorial for a popular web scraping tool and mimic the code you see there, adapting it to the website you've chosen. Along the way, you'll likely have to learn a little about HTML and CSS. Store the scraped data in a data format like a data frame that is idiomatic in your language of choice.
- Find some data and try to clean it. Raw data is plentiful on the Internet. You might try Kaggle [242] or the U.S. government site data.gov [482]. (The latter has lots of data in weird formats to give you some practice importing from a wide variety of file types.) You can find data on any topic by typing that topic in any search engine and appending the word "data."

Chapter 3

Linear Algebra

Jeffery Leader

Rose-Hulman Institute of Technology

This chapter covers three important aspects of linear algebra for data science: the storage of data in matrices, matrix decompositions, and eigenproblems. Applications are interspersed throughout to motivate the content.

Section 3.1 discusses how matrices and vectors are used to store data in data science contexts, and thus is a foundational section for the remainder of the chapter. Section 3.2 covers four matrix decomposition methods that show up in several different applications, together with example uses of each. Some portions of that section review common linear algebra material, and as such, readers familiar with the material can skip those sections, as the text indicates. Example applications include least-squares curve fitting and recommender systems, as used by Amazon, Netflix, and many other companies.

Section 3.3 introduces applications of eigenvalues and eigenvectors, together with the complexities that arise when doing numerical computations with them. Again, the first portion of that section can be skipped by readers who do not need a review of the foundational concepts. The primary application covered in this section is Google's PageRank algorithm.

The chapter then concludes with Section 3.4, on the numerical concerns that arise when doing any kind of precise computation on a computer, followed by Section 3.5, which suggests several projects the reader could undertake to solidify his or her understanding.

3.1 Data and matrices

3.1.1 Data, vectors, and matrices

Linear algebra techniques and perspectives are fundamental in many data science techniques. Rather than a traditionally ordered development of the subject of linear algebra, this chapter provides a review of relevant concepts mixed with and motivated by ideas from computational data analysis methods that emphasize linear algebra concepts: In particular, what are called vector space models, as commonly seen in information retrieval applications. Hence, the linear algebra material will be introduced as needed for the data science methods at hand.

Although data is a broad term, data that is processed mathematically is often stored, after some degree of pre-processing, in one of the forms common from matrix algebra: a one-dimensional array (a vector) or a two-dimensional array (a matrix).

A vector is appropriate for one-dimensional data, such as rainfall in inches over a series of n days. We use the vector $r = (r_i)_{i=1}^n$ to store the measurements $r_1, r_2, \ldots, r_n$. By convention, we take a vector to be a column rather than a row if not otherwise specified. Addition and subtraction of two vectors of common length n are to be performed elementwise, as is the multiplication of a vector by a scalar. In principle our vector could be categorical rather than numerical—a list of book genres, say—but we will consider only real numbers as individual data in this chapter.

Many datasets of interest are two-dimensional. A video streaming service tracks which of its m users has watched which of its n programs; such data is naturally stored in a two-dimensional array with, say, m rows and n columns, $A = (a_{i,j})$ $(i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n)$, that is, an $m \times n$ matrix. The entry $a_{i,j}$ in this users-by-programs matrix may represent the number of times user i has watched program j, or it may simply be set to one if the user has watched the program at least once, and zero otherwise. (Other less obvious weightings are frequently used.) This table represents data that might be mined for recommendations: If you liked the movie corresponding to $j = 2001$, you'll love the TV show corresponding to $j = 4077$! In a data analysis context, the indices, and perhaps the corresponding values, may be referred to as "dimensions," "features," or "attributes" [75].

In general, there's no reason the number of rows should equal the number of columns for a given matrix. (Why should the number of people streaming shows equal the number of available shows?) Hence we do not expect our matrices to be square, in general.

It's sometimes useful to be able to flip such a tabular representation; perhaps we decide it will be easier to have the video streaming data stored with the programs corresponding to the rows and the users to the columns instead. For this reason and others the transpose, A^T, will occur in our work.

We often describe two-dimensional data arranged rectangularly as being in a table. But vectors and matrices differ from (mere) columns or tables of numerical data in an important way: Vectors and matrices have algebraic properties. Addition, subtraction, and multiplication of matrices of the conforming sizes are possible. (Of course, unlike matrices, tables often carry non-numerical information concerning what is actually in the table.)

Occasionally elementwise multiplication will also make sense: If a table lists our customers who rented movies from us in rows and the titles of those movies as columns, with a 0 or 1 entry to indicate rental status (that is, it is a binary, or boolean, matrix), and we obtain from another data collector a corresponding table which lists individuals who have rated movies by rows and the titles of those movies as columns, with a rating from 1 to 5, then after eliminating from the latter table any rows corresponding to viewers who are not actually our customers and otherwise formatting the new data to match our internal format—the unavoidable task of data cleaning and management that accounts for so much effort in data mining applications—the element-by-element product shows what ratings our users gave to titles they rented from us, while dropping out their ratings of other titles.

Surprisingly often, however, traditional matrix multiplication meets our needs. In Section 3.3 we'll use powers and products of matrices to identify multistep paths in a graph. For example, if matrix A has viewers as rows and movies as columns with zero-one entries indicating the viewer has seen the movie, and matrix B has movies as rows and actors as columns with zero-one entries indicating that the actor was in the movie, then the product AB contains information as to whether a viewer has seen a movie containing a given actor.

It's common in data science applications to consider not just a rectangular array of data but a higher-dimensional array. For example, a video streaming service tracks which of its m users has watched which of its n programs on which of the $p = 7$ days of the week. The data is naturally stored in a three-dimensional array with, say, m rows, n columns, and p tubes (the terminology for directions beyond rows and columns is not standardized), which forms a multidimensional array $M = (m_{i,j,k})$ $(i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n,\ k = 1, 2, \ldots, p)$; that is, an $m \times n \times p$ object. The entry $m_{i,j,k}$ could store the number of times user i has watched program j on day k. Most major programming languages allow for such arrays.

We call objects such as M multilinear arrays, multiway arrays, or discrete tensors. They could have an arbitrary number of dimensions (called the order of the array). For a d-dimensional array we describe the data itself as being d-way; that is, vectors store one-way data, matrices store two-way data, a three-dimensional (discrete) tensor stores three-way data, etc.

These objects have their own corresponding multilinear algebra. The concepts build naturally off those of matrix algebra. There are, however, many differences. We will concentrate on building a foundation in matrices and will make only passing reference to higher-order arrays, where appropriate. For further information on multilinear arrays, the reader is directed to [6].

Matrix algebra might not seem to have a particularly deep connection to tables of data. Perhaps surprisingly, however, a number of means of data analysis begin with a table of numerical data, interpret it as a matrix M in order to imbue it with algebraic properties, then attempt to factor it into a product of two or three other matrices, $M = AB$ or $M = ABC$, where one or more of the factor matrices will allow for useful insight into the nature of the data.

3.1.2 Term-by-document matrices

In a simple model of the web search problem, we imagine that there are m words that our search engine will recognize (the terms) and n web pages that will be indexed (the documents). This already raises a number of questions—for example, will we treat words like *associated* and *associating* as two different terms, or lump them together with other related words under a prefix such as *associat-* (referred to as *stemming* the terms)? If we encounter a word on a web page that is not on our list, do we increase m or simply ignore the word? But let's put aside issues of design and data cleaning for now.

After the time-consuming task of having a large number of computers crawl the web, we presumably know that term i appears on page j exactly $t_{i,j}$ times. Of course, if m is large we expect that $t_{i,j}$ is very often zero—most web pages do not mention aardvarks—but otherwise it is a positive integer. We consider this table as a matrix $T = (t_{i,j})$, called a term-by-document matrix, and say that we are using a vector space modeling approach. We sometimes say that the resulting matrix represents a bag-of-words mindset, in which we ignore sentence structure and merely treat documents as (unordered) collections of terms.

There's nothing special about the specifics here. (Indeed, we are basically describing the adjacency matrix of the graph of a relation.) The documents could be books instead of web pages. The terms could be codes referring to images from some comprehensive list of images (the "dictionary" of the images). The documents could represent different biological species considered as "books" spelled out in their genetic code, and the terms could be genes (or other words formed from distinct segments of genetic code). We may choose to refer to a matrix as a term-by-document matrix in many cases in which we emphasize a linear algebra approach to analyzing the data, and so we understand *term* and *document* very generally (e.g., the case of a users-by-programs matrix mentioned in the previous section).

One situation in which this language is common is when we expect to perform a query (search) that will be formulated as a vector, e.g., a situation in which a search for term i is based on the vector e_i (meaning column i of the appropriately sized identity matrix). This is called a vector information retrieval model. We will see additional uses of term-by-document matrices in machine learning, in Section 8.7.

Readers who wish to solidify the concepts introduced so far may wish to try the project in Section 3.5.1 at the end of the chapter before reading further.

3.1.3 Matrix storage and manipulation issues

This section looks at some of the details involved in the actual storage and manipulation of large matrices from a computational point of view—how the software and hardware are actually handling the data. For those who have never dealt with truly massive datasets before, it can prove worthwhile to understand what is going on within a software package; however, you can skip to the next section to continue focusing on the linear algebra itself and return to this section later if desired.

Most programming languages will support declaring a variable to be a scalar, a one-dimensional array, or a two-dimensional array (of a given data type—e.g., we might insist that all the entries of our vector be integers). In most cases these will not natively be understood as having associated with them the standard algebraic operations of matrix algebra; that is, these are generally representation-and-storage schemes only.

Computer memory is, for all practical purposes, one-dimensional (that is, linear). This means that even if a matrix is stored contiguously in memory, it is not stored the way we are apt to envision it. It must be "unraveled" to a one-dimensional form. In some languages the default is to first store row one, then row two, then row three, etc., which is called row-major format; C++ is an example. In others the default is to first store column one, then column two, then column three, etc., which is called column-major format; Fortran is an example. More sophisticated options may be available, depending on the particular hardware and software being used. What's important about this issue is that entries that are far from one another in memory generally take significantly longer to retrieve; loading the values $A_{1,1}$ and $A_{1,2}$ will be much faster than loading $A_{1,1}$ and $A_{2,1}$ in the case of a row-major language, and the opposite will be true for a column-major language. Our algorithms must be aware of this. As an example, computing the product Ax by repeated dot products of the rows of A with the vector x makes sense in C++, but in Fortran it would be wise to consider using the alternative formula

$$Ax = A_1x_1 + A_2x_2 + \cdots + A_nx_n$$

(where A_i is column i of A) instead. If such an option were not available, one might attempt to rewrite the problem in terms of A^T, but this is not always convenient. Software cannot always be counted on to handle this matter seamlessly, so a users-by-programs matrix will make more sense than a programs-by-users matrix in some cases, for example.

In practice, as it turns out, very large matrices are very frequently sparse—that is, they have many more null entries than non-null entries. (We might well find them impractical to store and manipulate otherwise.) There is no strict definition of sparsity, but a common cut-off is that sparse matrices are at least 95% zeros. A matrix that is not sparse is said to be dense; a very dense matrix might be described as full.

For a large, sparse matrix, we can vastly speed up our computations if we can construct a data storage scheme that stores only the non-null entries of our matrix, together with algorithms that use only these values (as opposed to, say, multiplying values by zero time and time again). There are a great many ways to do so. Although the subject is beyond the scope of this chapter, it's worth seeing one simple approach, the coordinate format (COO). We store the matrix

$$A = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 4 \\ 0 & 0 & 1 & 5 & 7 \\ 6 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

as three vectors (plus some header information), say:

$$\begin{aligned} r &= (1, 2, 2, 3, 3, 3, 4, 4) \\ c &= (1, 4, 5, 3, 4, 5, 1, 5) \\ v &= (3, 2, 4, 1, 5, 7, 6, 8) \end{aligned}$$

where we interpret the first entries of these three vectors to mean that the $(1,1)$ entry of the matrix is 3, and the final entries to mean that the $(4,5)$ entry is 8. Note that if we don't separately store the fact that the matrix is 5×5, we risk losing the last row of A should we attempt to reconstruct it.

A few comments on this simplistic scheme: First, in this case it isn't very efficient. We are storing 24 values using COO versus the 25 entries for the entire matrix. For a matrix with floating point entries we will see some savings from the fact that the entries of the row and column index vectors can be stored using less memory as integers, but that doesn't buy us much. Second, in our example, it's relatively easy to access A by row but harder to access it by column. (We could have made the other choice, making it more easily accessible by columns instead.) Look at the three vectors and imagine a much larger example: Using r it's easy to locate row 3, but finding column 5 from c involves some searching.

This format is convenient for assembling a matrix initially because as new data arrives we can always add a new entry–say, making $a_{2,2}$ now be 9–by appending to the vectors we currently have:

$$\begin{aligned} r &= (1,2,2,3,3,3,4,4,2) \\ c &= (1,4,5,3,4,5,1,5,2) \\ v &= (3,2,4,1,5,7,6,8,9) \end{aligned}$$

(though until we re-sort these to make the row vector ascending again we've lost the ease of accessing A by rows). Perhaps if a matrix is sparse enough we could afford to store it using even more vectors so as to have it sorted both by rows *and* by columns? We will want our storage scheme to enable us to quickly access the data and to be amenable to whatever computation we will be performing. Often we'll use it to distribute the data over multiple machines in a parallel computing environment, which will certainly be facilitated by having less to move.

How best to store data is a large and important subject, but not our focus in this chapter. However, as you consider the matrices and methods we encounter along the way, it's good to give some thought to the storage schemes that might be used for them. Underlying every algorithm throughout this book that uses matrices, graphs, or other relationships, there is likely an array that would have many merely placeholding null entries in a traditional representation. Sometimes software will seamlessly hide the issue from the data scientist, but in many cases it will be incumbent upon the user to consider how best to store and compress the information. As a classic computer science text puts it, *Algorithms + Data Structures = Programs* [510].

The Basic Linear Algebra Subprograms (BLAS) standard is a specification for general-purpose computational matrix algebra routines that frequently occur in larger programs. Ideally the hardware vendor—the actual manufacturer of the computer—provides subroutines optimized for that machine's particular architecture, taking advantage of the known layout of caches, memory latencies (access times), available extended-precision registers, and so on, to get code that runs as fast as possible on that device. Well-known BLAS routines include the various "axpy" routines that compute quantities of the form

$$ax + y$$

such as `saxpy` ("single precision a x plus y") or `daxpy` ("double-precision a x plus y"), where a is a scalar and x and y are vectors.

The BLAS come in three levels. Level 1 BLAS routines perform vector-vector operations that are $O(n)$, such as the `daxpy` operation $y \leftarrow ax + y$ in which the output overwrites y. Level 2 BLAS perform matrix-vector operations that are $O\left(n^2\right)$, such as matrix-vector multiplication (e.g., `dgemv`, which performs $y \leftarrow \alpha Ax + \beta y$ in double-precision, where α and β are scalars, A is a matrix, and x and y are vectors; an optional flag can be set to compute $\alpha A^T x + \beta y$ instead). Level 3 BLAS perform matrix-matrix operations that are $O\left(n^3\right)$, such as matrix-matrix multiplications.

Even if vendor-supplied BLAS are not available, the use of generic BLAS subroutines, coded by experts for high efficiency, can speed up computations considerably. Packages are freely available on the web. The ATLAS (Automatically Tuned Linear Algebra Subprograms) package will test your machine, attempting to reverse-engineer its hardware capabilities, and produce near-optimal versions of BLAS routines.

Data analysis performed in an environment such as MATLAB already has a rich set of computational matrix algebra routines available; often, as with MATLAB, specialized packages for applications like data analysis will also be available. Lots of other languages, such as the scientific and numerical programming adjuncts for Python, provide relatively easy access to a suite of such programs. If one is using a programming language that does not provide easy support for such computations, or if a highly specialized routine is needed, it's useful to know that there are high-quality, free or minimally-restricted packages of such routines available, typically in at least Fortran and C (or C++).

An excellent repository of these is Netlib (www.netlib.org). The LAPACK (Linear Algebra Package) software library contains a wealth of routines; MINPACK (for minimization) is among the other packages available there. For multilinear arrays, TensorFlow (www.tensorflow.org) and, for MATLAB, Tensor Toolbox are available.

There are many other such packages, including commercial ones. If you don't need to write the code yourself, don't! Teams of experts, working over decades, with constant user feedback, have probably done a much better job of it already.

3.2 Matrix decompositions

3.2.1 Matrix decompositions and data science

There are several standard ways to factor matrices that occur in applications, including but by no means limited to data science. Each of them takes a matrix M and factors it as either $M = AB$ or $M = ABC$, and then finds meaning in one or more of the factor matrices. Some of these decompositions will be familiar to you (or at the very least will generalize operations familiar to you), while others may not be; but each has a role in data science. We'll begin by finding a way to take standard Gaussian elimination by row-reduction and write it in the form of a product of matrices, and then relate it to the solution of least-squares curve fitting—a basic, but very common, application in the analysis of data.

3.2.2 The LU decomposition

3.2.2.1 Gaussian elimination

Given an $m \times n$ matrix A, we often need to solve a linear system of the form $Ax = b$ where x is an n-vector and b is an m-vector. In fact, in using Newton's method to solve a linear system or in solving a partial differential equation via an implicit finite difference method, we will need to solve a linear system again and again as we iterate toward a solution (with a different A each time in the former case but often with the same A each time but a different b in the latter case). The basic means of doing this is Gaussian elimination by row-reduction (or a variant column-oriented version if the matrix is stored in column-major form). In data science applications, one of the most common ways of seeing this situation arise is when we need to solve the normal equation

$$\left(V^T V\right) a = V^T y,$$

where a is the unknown vector of coefficients in our model and V and y are known, based on the x (independent) and y (dependent) variables, respectively, as part of a regression analysis problem. (Readers who need a refresher on regression can look ahead to Section 4.3.1. The normal equation is discussed further in Sections 4.6.3 and 8.5.3.)

In most cases this operation will be done by a provided routine. For example, MATLAB's backslash command solves $Ax = b$ via the simple command `x=A\b`, and the backslash operator automatically chooses what it feels is the best way to do so (which is not always Gaussian elimination). Statistics packages will perform a linear regression without asking the user to explicitly solve the normal equations. For modest-sized datasets the response time is below the user's perceptual threshold, and so it makes little difference how the solution is being found. However, it is valuable to understand the ideas hidden underneath these routines in order to most efficiently make use of them for large datasets. Often the software will allow the user to select certain options that may improve runtime, an idea to which we'll return at the end of this section.

We assume the reader is familiar with Gaussian elimination but will review it here by example in order to introduce notation. The reader may skim the section for the necessary details or read carefully for a thorough review.

Given an $m \times n$ linear system consisting of, say, three equations R_1, R_2, and R_3, perhaps

$$\begin{aligned} R_1 &: \quad 2x_1 + x_2 + x_3 = 4 \\ R_2 &: \quad x_1 + 2x_2 + x_3 = 4 \\ R_3 &: \quad x_1 + x_2 + 2x_3 = 4, \end{aligned}$$

we note that the solution set of the linear system is left unchanged by the following three elementary equation operations: $E_1 : R_i \leftrightarrow R_j$, interchanging the order of the two equations R_i and R_j; $E_2 : cR_i$, multiplying equation R_i by the nonzero scalar c; and $E_3 : mR_i + R_j \to R_j$, adding a multiple m of equation R_i to equation R_j, replacing the latter. In Gaussian elimination, we use E_3 to make zero the coefficients of x_1 in the second row through the last, then use E_3 to make zero the coefficients of x_2 in the third row through the last, continuing in this way until we have eliminated all subdiagonal terms, using E_1 when necessary to move an entry into the diagonal position if the coefficient there happens to be zero. Here we perform $(-1/2)\,R_1 + R_2 \to R_2$, then $(-1/2)\,R_1 + R_3 \to R_3$, giving

$$\begin{aligned} R_1 &: \quad 2x_1 + x_2 + x_3 = 4 \\ R_2' &: \quad 0x_1 + \frac{3}{2}x_2 + \frac{1}{2}x_3 = 2 \\ R_3' &: \quad 0x_1 + \frac{1}{2}x_2 + \frac{3}{2}x_3 = 2, \end{aligned}$$

where, importantly, the last two equations now form a 2×2 system in the variables x_2 and x_3 alone. Finally, we perform $(-1/3)\,R_2' + R_3' \to R_3'$, giving

$$\begin{aligned} R_1 &: \quad 2x_1 + x_2 + x_3 = 4 \\ R_2' &: \quad 0x_1 + \frac{3}{2}x_2 + \frac{1}{2}x_3 = 2 \\ R_3'' &: \quad 0x_1 + 0x_2 + \frac{4}{3}x_3 = \frac{4}{3}, \end{aligned} \tag{3.1}$$

which has been significantly simplified, as the last equation now involves only one variable, the one above it only two, and the one above it all three. This completes the Gaussian elimination phase; we proceed to back-substitution, solving the last equation R_3'' to find that $x_3 = 1$; then using this in R_2' to find that $x_2 = 1$; and finally using these values of x_2 and x_3 in R_1 to find that $x_1 = 1$ as well. We have solved the system.

3.2.2.2 The matrices L and U

It's unreasonable to ask the computer to manipulate the symbolic entities x_1, x_2, and x_3, and it was noticed very early on that one need merely write the coefficients in columns and perform the corresponding operations as though the symbols were still present. In modern linear algebraic form, we write the given system in the form $Ax = b$, where

$$\begin{aligned} A &= \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \\ x &= \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \\ b &= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \end{aligned}$$

and form an augmented matrix

$$\left(A \,\middle|\, b\right) = \left(\begin{array}{ccc|c} 2 & 1 & 1 & 4 \\ 1 & 2 & 1 & 4 \\ 1 & 1 & 2 & 4 \end{array}\right),$$

where the line has no algebraic effect and may be omitted. In effect, the four columns are the x_1, x_2, x_3, and b columns, respectively. Taking the equation labels R_i to now refer to the rows of this matrix, we once again perform $(-1/2)\,R_1 + R_2 \to R_2$ and $(-1/2)\,R_1 + R_3 \to R_3$ in that order, changing $\left(A \,\middle|\, b\right)$ in turn, then $(-1/3)\,R_2' + R_3' \to R_3'$, giving

$$\begin{pmatrix} 2 & 1 & 1 & 4 \\ 1 & 2 & 1 & 4 \\ 1 & 1 & 2 & 4 \end{pmatrix} \sim \begin{pmatrix} 2 & 1 & 1 & 1 \\ 0 & 3/2 & 1/2 & 2 \\ 0 & 0 & 4/3 & 4/3 \end{pmatrix},$$

where the tilde indicates that the two matrices differ only by one or more of the three elementary equation operations, now termed elementary row operations. We say that the two matrices are row equivalent. The Gaussian elimination stage is completed, with the final matrix upper triangular. We have row-reduced the augmented matrix to one in triangular form. By convention, any rows consisting entirely of zero entries are placed at the bottom in the final result. We now reinterpret this system in the form of equation (3.1) and apply back-substitution as before. In this case, A is evidently nonsingular (that is, invertible), as it is square and not row-equivalent to a matrix with a row of all zeroes.

Gaussian elimination is usually the best way to solve small-to-medium, and certain specially structured, linear systems. We associate with each elementary row operation an elementary row matrix in the following way: The elementary matrix Λ corresponding to performing operation E on an $m \times n$ matrix is the matrix obtained by applying that operation to the identity matrix of order m. Multiplication on the left by Λ is easily shown to have the same effect as performing the operation directly on the matrix. For example, defining

$$\Lambda_1 = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

we find that

$$\Lambda_1 \left(A \,\middle|\, b \right) = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 0 & 3/2 & 1/2 & 2 \\ 1 & 1 & 2 & 1 \end{pmatrix}$$

directly implementing the elementary row operation $(-1/2)\,R_1 + R_2 \to R_2$. Continuing in this way, $\Lambda = \Lambda_3\Lambda_2\Lambda_1$ is a matrix that reduces $\left(A \,\middle|\, b \right)$ to the desired form.

We can therefore implement row reduction to upper triangular form as a sequence of matrix multiplications. This would be very inefficient as code but leads to valuable insights. Omitting the details, when the original matrix can be reduced to upper triangular form using only operations of the form $-m_{j,i}R_i + R_j \to R_j$, there is a lower triangular matrix Λ with ones on its main diagonal and entries $-m_{j,i}$ in the lower triangle, each occupying the position it was intended to eliminate, such that $\Lambda \left(A \,\middle|\, b \right)$ has the desired form, an upper triangular matrix. Hence, we define $U\Lambda = A$.

If a square matrix F admits another square matrix G of the same order such that $FG = I$ (the identity matrix), then it must be that $GF = I$ also. We say that F and G are inverses of one another and that each is invertible and write $G = F^{-1}$. The invertible matrices are exactly the nonsingular matrices.

In principle this means that we could solve $Ax = b$ when A admits an inverse in the following way: Find A^{-1}, then multiply x by it to produce $x = A^{-1}b$. Unfortunately, this method is both less computationally efficient and less accurate than the method of Gaussian elimination, and is never used. (Recall that a standard way of finding A^{-1} is to row-reduce $\left(A \,\middle|\, I \right)$ until we obtain the identity matrix on the left-hand side; this requires manipulating n columns on the right-hand side, while working directly on $\left(A \,\middle|\, b \right)$ involves only one column on the right. Surely the extra computations involve more time and more round-off error, and even then, we must still perform a matrix-vector multiplication at the end to finally find x.) Furthermore, as a rule of thumb, sparse matrices often have full inverses, meaning we may pay a considerable price in storage. Inverses are valuable primarily in symbolic operations. Even when we need to solve $Ax = b$ repeatedly with the same A but different b vectors, finding the LU decomposition of A once and using it for every following case to solve $LUx = b$ as b varies is almost always the less costly, more accurate choice.

But now consider again the equation $\Lambda A = U$. The matrix Λ is not merely lower triangular; it is unit lower triangular, that is, all its main diagonal entries are ones. It's clear that Λ^T is nonsingular because it is already in (upper triangular) reduced form, so Λ is itself nonsingular—hence invertible—as well. The inverse of a unit lower triangular matrix is also a unit lower triangular matrix, as can be seen by writing $\Gamma\Gamma^{-1} = I$, with Γ^{-1} full of unknowns below the diagonal. If one computes the product, the values of the subdiagonal entries follow immediately, and the ones below them next, and so on. In our example,

$$\Lambda = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/2 & -1/3 & 1 \end{pmatrix} \text{ and } \Lambda^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 2/3 & 1/3 & 1 \end{pmatrix},$$

and applying this to $\Lambda A = U$ gives $\Lambda^{-1}\Lambda A = \Lambda^{-1}U$. Letting $L = \Lambda^{-1}$, we obtain the LU decomposition $A = LU$ of the original matrix of coefficients A. Note that it is trivial to construct L as we perform Gaussian elimination, as its entries can be found easily from the $m_{j,i}$ (called the multipliers); in fact, we could store those entries in the space cleared from A as it is transformed in-place in memory to U, requiring no extra storage whatsoever. (There is no need to store the known unit entries on the main diagonal of L). Hence, the LU decomposition requires only the storage space of A itself.

This factorization represents the traditional means by which a system of the form $Ax = b$ is solved by machine. We begin with A, form L and U (possibly overwriting A in memory in the process), then consider the problem in the form $LUx = b$, i.e., $L\,(Ux) = b$. Temporarily writing y for the unknown vector Ux, we solve $Ly = b$ by forward-substitution in the obvious way, then solve $Ux = y$ by back-substitution to find the answer x. This has no computational advantage over the cost of simple Gaussian elimination with back-substitution for an $n \times n$ system if we only wish to solve $Ax = b$ once, but often we next seek to solve $Ax = d$ for a new vector d. In this case, using the previously computed L and U reduces the cost by an order of magnitude. (See Section 3.4.)

3.2.2.3 Permuting rows

We have neglected an important consideration—we will likely need to interchange rows at some point. First, we note that a matrix such as

$$K = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

cannot be row-reduced to upper triangular form without the row interchange operation; we must perform $R_1 \leftrightarrow R_2$. In fact, performing that operation on the identity matrix produces an elementary matrix

$$\Pi = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

and ΠK is upper triangular, as desired. Trivially, Π is its own inverse here. We permute rows as needed throughout the row reduction process, leading to an equation of the form $MA = U$ where M is a product of some number of permutation and unit lower triangular matrices. The corresponding decomposition is $A = M^{-1}U$.

Perhaps surprisingly, however, it's always possible to write M^{-1} as the product of a single permutation matrix P (a row-permuted version of the identity matrix) and a unit lower triangular matrix L. The resulting decomposition is called the PLU decomposition $A = PLU$. The solution process for $Ax = b$ is little changed; permutation matrices have the property that P^{-1} exists and equals P^T, so $PLUx = b$ becomes $LUx = P^T b$ and we proceed as before.

Because permuting rows is so common in practice, it's usual to call the PLU decomposition simply the LU decomposition. Often we work with the two matrices PL and U; the former is merely a permuted version of a lower triangular matrix and is essentially just as valuable. It's also common to simply store a permutation vector p that indicates in which order the rows of L would have been permuted, rather than to actually move the rows around in memory.

But second and more generally, the form of Gaussian elimination we have described is called naive Gaussian elimination, and it is numerically unstable in finite precision arithmetic; that is to say, it has the property that the inevitable small errors in each step of the computation can grow so large that the computed solution is grossly inaccurate. To improve its accuracy, when we begin to consider column 1 we scan down the column and select from all its entries the one that is largest in absolute value, then perform a row interchange to bring it to the topmost row. This process is called pivoting. We then clear the column as before. Moving to column 2, we consider all entries from the second row downward and again pivot (if necessary) to bring the largest entry in absolute value to the second row. We then clear down that column. This approach is called Gaussian elimination with partial pivoting (GEPP). Intuitively, when the multipliers are all less than unity in magnitude we are multiplying the unavoidable errors present by a quantity that diminishes them; when they are larger than unity, however, we risk exaggerating them.

When the LU decomposition is referred to in practice it's virtually certain that a PLU decomposition performed with partial pivoting is meant. It isn't perfect, but it's good enough that it's the standard approach. For additional stability, the more expensive method of maximal (or complete) pivoting performs both row and column interchanges to bring the largest value remaining in the part of the matrix left to be processed to the diagonal position.

3.2.2.4 Computational notes

We now offer a few computational notes that the reader may skip if desired. The purpose is primarily to alert you to the existence of possible concerns and challenges when using this decomposition with a truly large matrix. How large is large? That depends on your software and hardware, as well as how quickly you need a result. Scientific data processing problems from, say, particle physics or astronomy can involve a truly impressive amount of data.

Algorithms were traditionally judged by how many floating point operations they required—additions, subtractions, multiplications, divisions, and raising to powers (including extraction of roots). Since additions/subtractions tend to require less computer time than the other operations, sometimes the additions and subtractions are counted separately from the multiplications and divisions. For simplicity, we'll lump them together.

A careful count shows that solving a general $n \times n$ linear system by Gaussian elimination, including the back-substitution step, takes $(2n^3+3n^2-5n)/6$ additions and subtractions, $(2n^3+3n^2-5n)/6$ multiplications, and $n(n+1)/2$ divisions. This is

$$\frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n$$

floating point operations (flops). For large matrices, this is approximately $(2/3)\,n^3$, and we say that the method has computational complexity that is of the order of n^3, denoted $O\left(n^3\right)$: Working with twice as much data (i.e., doubling n) requires about eight times as many operations for large n. This doesn't account for the cost of the searches required by partial pivoting, about $n^2/2$ comparisons.

This is one reason why we use the LU decomposition. Although we pay an $O\left(n^3\right)$ initial cost to find it, subsequent solutions involving the same matrix incur only an $O\left(n^2\right)$ cost for the forward- and back-substitution phases. We do need to be somewhat suspicious here: Being told the method has an $O\left(n^2\right)$ cost means that for large n the cost is proportional to n^2, but it's not clear what that constant of proportionality is; what if it actually grows like $10^6 n^2$? In fact, each phase takes about n^2 operations, so the cost is about $2n^2$. Even for modest n, this is much smaller than $(2/3)\,n^3$.

If the matrix is sparse, we can almost surely do much better. For example, a tridiagonal matrix, that is, one that has nonzero entries only on the main diagonal and the immediate superdiagonal and subdiagonal, can be solved in $8n-7$ flops via a variant of Gaussian elimination called the Thomas algorithm.

When processor speeds were lower and most computations were done on serial machines that processed a single floating point operation at a time, one after another, this type of analysis allowed for easy comparison between competing algorithms. But after decades of Moore's Law—the continual increase of processor speeds—computation time is much less of a bottleneck than it used to be. Add in the widespread use of parallel machines, from increasingly common quad core laptops with four processors to machines with thousands of processors, and the picture is somewhat different. And importantly, memory access speeds have not increased at the same rate as processor speeds. For large matrix computations, or for accessing data in general, retrieving information from memory is often the major concern.

For this reason, algorithms implementing this decomposition and the other methods we are going to discuss will, ideally, be aware of the memory layout of the device on which they are to be used. Many machines will have one or more levels of small-but-fast cache memory nearer the processor than main memory is, for example, and this may be accessible an order of magnitude more rapidly. When reading a value from memory, we hope to make the greatest reuse of it possible while we have it near the processor.

All of the algorithms that we will discuss have approximate versions that both reduce computation costs and reduce memory usage. For example, when computing the LU decomposition of a large, sparse matrix for which we have devised a specialized data structure and in a case in which we are overwriting the matrix in memory, the row-reduction operation will likely introduce nonzero elements where previously the entry was zero (and hence not stored). This phenomenon of "fill-in" can result in costly reworking of the data structure on-the-fly. One might choose to use an incomplete LU decomposition that simply declines to create a new nonzero entry, skipping the computation and saving current memory read/write time as well as later memory access and computation time, or one that thresholds, that is, sets small values to zero to save working with them. In many cases these provide an adequate approximation; in other cases they provide an initial guess that is used by another algorithm to refine the solution to a more accurate one.

Many issues in computational matrix algebra are highly hardware-specific, including the selection of the numerical algorithm in the first place. Importantly, memory access-time concerns typically dominate computation-time concerns in modern high-performance, large-scale computational matrix algebra. On a massively parallel machine, it's often problematic just getting the data to the processor nodes in a timely manner; processors sit idle for the majority of the runtime in many applications.

These are the kind of options that are often user-selectable: Overwrite the current matrix (or don't), use thresholding, prohibit fill-in, use a different form of pivoting, and so on. The point is not to write your own LU decomposition routine but rather to make smart use of the existing ones. Mathematicians and computational scientists do write much of the software that is used in data science, and this sort of under-the-hood application of mathematics in the field is crucial in building the software that enables data scientists to do their work. For the end-user interested in applying such tools, what's important is to know enough about how they work.

3.2.3 The Cholesky decomposition

There's an important variation of the LU decomposition that is widely used in least-squares curve-fitting. A matrix is symmetric if it is equal to its own transpose, that is, if $B = B^T$. Such a matrix is necessarily square. We say that a symmetric matrix B is positive semi-definite if for all conformable vectors x,

$$x^T B x \geq 0$$

and that it is positive definite if, in fact, $x^T Bx > 0$ for all nonzero x (so that a vector norm can be generated by taking the square root of this quantity, which is called a quadratic form in B and represents the quadratic terms in a polynomial in the vector x). Importantly, the matrix appearing in a least-squares problem is almost always of this type.[1]

In the case of such a matrix, we can perform the LU decomposition in such a way that $U = L^T$, that is, so that $B = LL^T$ (equivalently, $B = U^T U$). This is called the Cholesky decomposition. To indicate why this is possible, consider the positive definite matrix

$$B = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

and perform the initial elimination step, $(-1/2)\,R_1 + R_2 \to R_2$ using the matrix Λ_1 from earlier, giving

$$\begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 3/2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

and note that the resulting matrix has lost its symmetry. But if we now apply Λ_1^T on the right, we have

$$\begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}\begin{pmatrix} 1 & -1/2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3/2 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

because whatever row operation an elementary row matrix performs when used on the left of the matrix (premultiplication of B), its transpose performs the corresponding column operation when applied on the right (postmultiplication). We now have

$$\begin{aligned} \Lambda_1 B \Lambda_1^T &= D \\ B &= L_1 D L_1^T \end{aligned}$$

with L_1 the inverse of Λ_1. For a larger matrix, we would once again have a product of many L_k matrices, the product of which would be unit lower triangular. (Pivoting is never needed for existence of the decomposition nor even for numerical stability for a positive definite matrix, though it can be used for extra accuracy [192].) The end result on the right is always a diagonal matrix, and the positive definiteness of B compels the diagonal entries of D

[1] We will see another application of quadratic forms in distance measures used in cluster analysis in Section 5.3.

to be positive. Separating D as

$$\begin{aligned} D &= \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3/2 & 0 \\ 0 & 0 & 2 \end{pmatrix} \\ &= \begin{pmatrix} \sqrt{2} & 0 & 0 \\ 0 & \sqrt{3/2} & 0 \\ 0 & 0 & \sqrt{2} \end{pmatrix} \begin{pmatrix} \sqrt{2} & 0 & 0 \\ 0 & \sqrt{3/2} & 0 \\ 0 & 0 & \sqrt{2} \end{pmatrix} \\ &= D_{1/2}^2 \end{aligned}$$

lets us write

$$\begin{aligned} B &= \left(L_1 D_{1/2}\right)\left(D_{1/2} L_1^T\right) \\ &= \left(L_1 D_{1/2}\right)\left(D_{1/2}^T L_1^T\right) \\ &= \left(L_1 D_{1/2}\right)\left(L_1 D_{1/2}\right)^T \end{aligned}$$

and postmultiplication by a diagonal matrix implements the elementary column rescaling operation so that $B = LL^T$ where L is lower triangular but not, in general, unit lower triangular. The algorithm is of course implemented in a very different way in code, but this establishes that we can decompose a positive definite matrix in a way that uses only about half the storage of a full LU decomposition, as we need store only L.

3.2.4 Least-squares curve-fitting

One of the most common tasks in data analysis is the fitting of a straight line to some data. Suppose that we have a series of observations $y_1, y_2, \ldots, y_n$ taken at locations $x_1, x_2, \ldots, x_n$. It's highly unlikely that the data falls on a line $y = a_1 x + a_0$, but if we are curious as to whether it is well-described by such a line, we might seek to find such coefficients

$$a = \begin{pmatrix} a_1 \\ a_0 \end{pmatrix}$$

leading to the system of equations

$$\begin{aligned} a_1 x_1 + a_0 &= y_1 \\ a_1 x_2 + a_0 &= y_2 \\ &\vdots \\ a_1 x_n + a_0 &= y_n \end{aligned}$$

which almost surely admits no solution. (We say the system is inconsistent.) Ignoring this for the moment, we write

$$\begin{aligned} V &= \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \\ y &= \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \end{aligned}$$

giving the system $Va = y$. Since this is presumably inconsistent, we instead seek the best "near-solution" a, which we define as the one that minimizes the norm

$$\|y - Va\|^2 = \sum_{k=1}^{n} (y_k - (a_1 x_k + a_0))^2 ,$$

that is, the solution that minimizes the sum of squares of the prediction errors. With this criterion, it follows that the least-squares solution is the solution of the so-called normal equation[2]

$$V^T V a = V^T y,$$

i.e.,

$$a = \left(V^T V\right)^{-1} V^T y,$$

though using the inverse would not be an efficient approach. Instead, we note that $V^T V$ is symmetric,

$$\left(V^T V\right)^T = V^T \left(V^T\right)^T = V^T V,$$

and in fact positive semi-definite,

$$x^T \left(V^T V\right) x = x^T V^T V x = (Vx)^T (Vx) = \|Vx\|^2 \geq 0.$$

(In most cases the matrix $V^T V$ is actually positive definite.) This allows us to use the Cholesky decomposition to find a. If the Cholesky decomposition of the positive definite matrix $V^T V$ is $L^T L$, then the normal equation becomes

$$\begin{aligned} L^T L a &= V^T y \\ L^T (La) &= V^T y, \end{aligned}$$

and we can write $d = La$ as a temporary variable, solve for it from $L^T d = V^T y$, then solve $La = d$ for a by back-substitution.

[2] We will see more about normal equation in Sections 4.6.3 and 8.5.3.

The approach easily generalizes. If instead we seek a quadratic model, say $y = a_2x^2 + a_1x + a_0$, we define

$$a = \begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

and the system of equations

$$\begin{array}{rcl} a_2x_1^2 + a_1x_1 + a_0 & = & y_1 \\ a_2x_2^2 + a_1x_2 + a_0 & = & y_2 \\ & & \vdots \\ a_2x_n^2 + a_1x_n + a_0 & = & y_n \end{array}$$

leads to

$$V = \begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ & \vdots & \\ x_n^2 & x_n & 1 \end{pmatrix}$$

representing the presumably inconsistent system $Va = y$. Once again, the normal equation

$$V^TVa = V^Ty$$

defines the least-squares solution, that is, the one that minimizes

$$\|y - Va\|^2 = \sum_{k=1}^{n} \left(y_k - \left(a_2x_k^2 + a_1x_k + a_0\right)\right)^2,$$

and we can find this solution via the Cholesky decomposition of V^TV. Note that V^TV is actually just a 3×3 matrix in this case, even if we have $n = 1000$ data points. We could continue with higher-degree polynomials.

A matrix such as V, with the property that each column moving leftward is the next highest elementwise power of the penultimate column, and the rightmost column is all ones, is called a Vandermonde matrix. Other authors might define it so that the powers increase moving rightward instead. There are specialized methods for solving Vandermonde systems $Va = y$ when they are consistent [192], but unless the system is square—so that we are, say, fitting a quadratic to three points—our Vandermonde system will almost surely have no solution.

The Cholesky decomposition requires about $(1/3)\,n^3$ flops. The use of the Cholesky decomposition thus speeds up the process of solving the normal equation considerably, while reducing memory access requirements. This is important when there are many parameters in the model, or when a program checks a great number of potential models and then returns the best fit from among them.

3.2.5 Recommender systems and the QR decomposition

3.2.5.1 A motivating example

Let's consider a major area of application of vector information retrieval methods, recommender systems. Video streaming services, among others, are heavily interested in these. Netflix, which offered a million-dollar prize for a better system at one point, believes the quality of its recommender system is essential.

We think the combined effect of personalization and recommendations save us more than $1B per year. [193]

Suppose A is a term-by-document matrix for a video streaming service in which the terms are movies and the documents are viewers, with a 1 in the (i, j) position if movie i has been selected for viewing at some point by viewer j, and a 0 otherwise. (A streaming service user is being viewed as a document that is "written" in terms of what movies the person has selected; we are what we watch in this model!) We assume these selections are based on the users' interests, and wish to recommend movies based on this data. It's usual to weight the entries in some way. A simple approach is to normalize each column. Lets call W the matrix obtained after rescaling each column of A by its norm (unless it is entirely zero, in which case we leave it alone). For definiteness, we'll let the users be

1: Alice, 2: Bob, 3: Cori, 4: Derek, 5: Eric, 6: Fay

and the movies be

1: *Godzilla*, 2: *Hamlet*, 3: *Ishtar*, 4: *JFK*,
5: *King Kong*, 6: *Lincoln*, *7: Macbeth*,

where we recognize that films 1 and 5 are both monster movies, films 2 and 7 are both based on Shakespearean plays, films 4 and 6 are both based on the lives of U.S. presidents, and film 3 is a comedy. The point, however, is that the algorithm need never be informed of these similarities and differences; it will infer them from the data, if there is enough of it.

In some sense, we are recognizing that each viewer has some domain knowledge, as it were, regarding movies, and that we might be able to access that knowledge. Of course, we are not accessing that knowledge by interviewing them, as might have been done before the data science revolution; we are finding it from the limited data of their preferences alone.

The streaming service collects the necessary information in the course of its operations. The raw term-by-document matrix might be the 7×6 matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix},$$

and after we normalize each column we have W. Now suppose a new user George joins the service and provides information on movies previously viewed and enjoyed. Perhaps the only three on our list are *Godzilla* (1), *JFK* (4), and *Macbeth* (7). What other films might George like? There's more than one way to try to answer this, but the vector information retrieval approach would be to form a query vector

$$v = (1, 0, 0, 1, 0, 0, 1)^T$$

with a 1 for each movie liked, and a 0 otherwise. We then ask which columns of W are closest to v; commonly, we interpret this to mean which have the smallest angle between them and v, that is, the largest absolute value of the cosine of the angle between the vector and v. Using a standard formula, this angle satisfies

$$|\cos(\theta_j)| = \frac{A_j^T v}{\|A_j\| \, \|v\|}$$

if we are using the unweighted matrix A, or

$$|\cos(\theta_j)| = \frac{W_j^T v}{\|v\|}$$

if we are using the weighted matrix W, for which $\|W_j\| = 1$. In fact, if we choose to normalize v too, say as $z = v/\|v\|$, then we can perform the entire calculation of these cosine values as $A^T z$, producing a vector of query results.

A common guideline is that if $|\cos(\theta_j)|$ is at least .5, we have found a relevant "document." Our results show that users 1 and 4 both produce cosine values of about .577. Our new user seems to like the sort of shows that Alice and Derek like; indeed, they share *Godzilla* and *King Kong* in common. Perhaps the service should recommend *Hamlet*, the only other film selected by either user? With more data, we should be able to make better recommendations.

But this isn't a sufficiently smart method. There is information "buried" in A, we presume, but obscured by the fact that we don't know more about the users, nor more than titles about the films—nothing regarding genre, actors, etc. But we can infer some of this information from the assumption that the viewers tend to have preferences regarding genre of film, favorite actors, etc.; they aren't just watching at random. What do their choices reveal, not only about themselves, but about the films?

In a true term-by-document matrix, with each row representing a different word and each column representing a different book, say, word choice is always somewhat arbitrary: In this sentence the word "choice" was used, but couldn't "selection," "preference," or "usage" all have served about as well? We might refer to a tennis "match," a baseball "game," a golf "tournament," and so on, but we recognize that, at least in this context, this *cluster* of concepts, the words are very closely related. If we are searching for the word "game" we might well want to have books on tennis matches and golf tournaments included, even if the precise word "game" appears nowhere in them. The fine detail provided by the precise word choices "match," "game," "tournament" can be beneficial when writing a book but less so when searching for one. Similarly for the films—if we hope to find similar films to use as recommendations, we might find it useful to fuzz over the distinctions between a King Kong film and a Godzilla film in order to cluster monster movies together. If we are provided a thesaurus for words or a genre list for movies we may be able to use it to search for such recommendations, but this doesn't really interrogate the accumulated data to see what people *really* want to view. It can't tell us whether people who have only watched more recent monster movies would also enjoy older ones, and of course, in a large, rich dataset, we hope to be able to identify the group of people who watch, say, modern monster movies, old black-and-white detective movies, and anything starring Johnny Depp, and recommend other movies they have watched to other members of that same group.

The rank of A (and of W) is 5, which is one less than it could have been ($rank(A) \leq \min(m, n)$). But we are going to use a rank-reduction approach that will produce an approximate W that is of even lower rank. Why? This forces the information that is presumed to be lurking within the matrix to further cluster together. Hopefully, this clustering will force *Godzilla* and *King Kong* together based on the associations lurking in A, blurring the distinction between them, which will be a useful blurring for us, as we need to discover the connection between them without being told, by mining the data in this way. We will *intentionally* fuzzify the information contained in A in order to better find such similarities.

3.2.5.2 The QR decomposition

The LU decomposition is of great importance in the many applications that involve the solution, and especially the repeated solution, of a linear system $Ax = b$ (for example, when implementing Newton's method), as we've just seen in using its variant the Cholesky decomposition for least-squares problems. In vector-space modeling applications, though, other decompositions are more widely used. The next matrix factorization we consider will decompose a matrix A as the product of an orthogonal matrix Q, that is, a square matrix with the property that $Q^TQ = I$ (so that $Q^{-1} = Q^T$), and an upper triangular matrix R. (We might have used U here instead, but as it will very likely be a different upper triangular matrix than the one produced by the LU decomposition, we use the alternative term "right triangular" for an upper triangular matrix.) This will allows us to reduce the rank of our term-by-document matrix in a systematic way that tends to reveal latent information of the sort we seek.

The definition of an orthogonal matrix is driven by the concept of orthogonality of vectors. Two vectors v_1 and v_2 are said to be orthogonal if their inner product $v_1^Tv_2$ is zero; we say that the set $\{v_1, v_2\}$ is orthogonal. If the vectors are also normalized to length one we say that the set is orthonormal. The requirement that $Q^TQ = I$ is actually just the requirement that every column of Q be orthogonal to every other column of Q, and that the norm of any column be unity. Hence the columns are pairwise orthogonal and normalized to length one.

Every matrix has an LU decomposition, and every matrix has a QR decomposition as well. (In general the factor matrices are not unique in either case.) There are two main ways of finding a QR decomposition of a matrix. The first is based on the orthogonality of Q. As this relates to the orthonormality of its columns, let's consider this from the vectorial point of view for a moment. The span of a set of vectors is the set of all possible linear combinations of them, for example, $span(\{v_1, v_2\})$ is the set of all

$$\alpha_1v_1 + \alpha_2v_2$$

for all scalars α_1, α_2 (real scalars for our purposes). The span of a set of vectors is a vector space. (We assume the formal definition of this is familiar. Importantly, a vector space exhibits closure under the operations of addition and scalar multiplication.) Ideally the vectors from which we form the vector space are linearly independent, with no vector capable of being written as some linear combination of the others. In this case we say that the number of vectors in the set is the dimension of the vector space. Any set of vectors generating the same vector space is said to be a basis for that space and must have precisely the same number of vectors in it. In particular, in a vector space of dimension n, a set of more than n vectors cannot be linearly independent (and so we say that it is linearly dependent). The set of all n-vectors has dimension n.

Importantly, if $\{v_1, v_2, \ldots, v_n\}$ is a basis for a vector space, and v is an element of that vector space, then there is one and only one way to express v as a linear combination of the basis vectors; that is, there exists a unique sequence of scalars $\alpha_1, \alpha_2, \ldots, \alpha_n$ such that

$$v = \alpha_1v_1 + \alpha_2v_2 + \cdots + \alpha_nv_n$$

and if B is the matrix with columns $v_1, v_2, \ldots, v_n$, then

$$v = B \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix}$$

so that we may find the coefficients $\alpha_1, \alpha_2, \ldots, \alpha_n$ by solving a linear system (say, by Gaussian elimination). Of course, if the basis vectors are orthonormal, then B is an orthogonal matrix and we can find the coefficients easily from the fact that B^{-1} is just B^T in this case, so that

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix} = B^T v. \tag{3.2}$$

This is just one reason why orthonormal bases are desirable.

Now, consider a set of vectors $\{v_1, v_2, \ldots, v_n\}$ that is linearly independent. These might represent (document) columns of a term-by-document matrix. We may want to find an orthonormal set of vectors $\{u_1, u_2, \ldots, u_n\}$ with the same span. The standard way of doing this is the Gram-Schmidt process: We begin by letting u_1 be the normalized version of v_1,

$$u_1 = \frac{v_1}{\|v_1\|}$$

and then form u_2 by subtracting off the projection of v_2 along u_1, then normalizing the result.

$$\begin{aligned} u_2' &= v_2 - \left(v_2^T u_1\right) u_1 \\ u_2 &= \frac{u_2'}{\|u_2'\|} \end{aligned}$$

Continue this process, giving next

$$\begin{aligned} u_3' &= v_3 - \left(v_3^T u_2\right) u_2 - \left(v_3^T u_1\right) u_1 \\ u_3 &= \frac{u_3'}{\|u_3'\|} \end{aligned}$$

and so on. It's easily verified that the set $\{u_1, u_2, \ldots, u_n\}$ is orthonormal. Some care must be taken when implementing the algorithm to address possible loss of accuracy when n is not small, but this is not our concern here.

It's possible to use this algorithm to find the QR decomposition of a matrix. Let's assume that A is a not-necessarily-square $m \times 3$ matrix and, for the sake of convenience, that it has linearly independent column vectors. If we let v_j represent column j of A, so that $A = \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix}$, and set

$$R_1 = \begin{pmatrix} 1/\|v_1\| & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

then postmultiplication by R_1 rescales column 1 of the matrix by the reciprocal of its norm. Comparing this to the first step of the Gram-Schmidt process, we see that

$$AR_1 = \begin{pmatrix} u_1 & v_2 & v_3 \end{pmatrix}$$

and so now letting

$$R_2 = \begin{pmatrix} 1 & -\left(v_2^T u_1\right) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

we find that

$$(AR_1)\,R_2 = \begin{pmatrix} u_1 & u_2' & v_3 \end{pmatrix}$$

so that

$$R_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1/\|u_2'\| & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

provides the desired rescaling: $AR_1R_2R_3 = \begin{pmatrix} u_1 & u_2 & v_3 \end{pmatrix}$. The first two columns of this matrix are pairwise orthonormal; we are getting closer to the desired Q matrix. Finally, we use an R_4 to find u_3' and then an R_5 to convert it to u_3. This gives

$$\begin{aligned} AR_1R_2R_3R_4R_5 &= \begin{pmatrix} u_1 & u_2 & u_3 \end{pmatrix} \\ AS &= Q, \end{aligned}$$

where S, the product of the upper triangular R_i matrices, is also upper triangular. Its inverse R exists and is also upper triangular, giving the desired QR decomposition, $A = QR$. (In general, Q is $m \times m$ and R is $m \times n$.) This is sometimes called a triangular orthogonalization of A; we use successive triangular matrices to make the matrix more and more orthogonal at each step, much as the LU decomposition is a triangular triangularization process, using (lower) triangular matrices to create an upper triangular one. Of course, an actual program would proceed in a more efficient way.

Given a matrix, the span of its row vectors is called its row space, and the span of its column vectors is called its column space. The column space of A is in fact the range of the function $y = Ax$, that is, the set of all possible results y as x ranges over all appropriately-sized vectors, for

$$y = Ax = A_1x_1 + \cdots + A_nx_n,$$

where A_j is column j of A. If the columns of A are linearly independent, the columns of Q are an orthonormal basis for the column space. More generally, if the columns of A are not necessarily linearly independent, if p is the dimension of the column space of A, that is, p is the rank of A, then the first p columns of Q form an orthonormal basis for the column space of A. (The additional columns are needed for conformability of the product QR.)

It's also possible to find the QR decomposition via orthogonal triangularization methods. The more commonly used version of this approach is based on successive orthogonal matrices implementing reflections, but a special-circumstances method uses orthogonal matrices implementing rotations [306].

The rank of a matrix is the number of linearly independent row vectors it possesses, which always equals the number of linearly independent column vectors (and hence $rank(A) \leq \min(m, n)$). While in principle we could find this via the LU decomposition by asking how many rows of U are not entirely zero, round-off error complicates the matter. A better way to do this is by asking how many nonzero diagonal entries there are in the R matrix of the QR decomposition.

Also, although finding the QR decomposition is a little more expensive than finding the LU decomposition, it also tends to yield slightly more accurate results. Indeed, when solving the normal equation of least-squares curve-fitting, it's usually advisable to consider using the QR decomposition instead, as follows.

$$\begin{aligned} A^T A x &= A^T b \\ (QR)^T (QR) x &= (QR)^T b \\ R^T Q^T Q R x &= R^T Q^T b \\ R^T R x &= R^T Q^T b \end{aligned}$$

(Recall that $Q^T Q = I$.) This avoids the creation of $A^T A$, which compresses n data points into a relatively small matrix (where n may be quite large). This can both save time and improve accuracy lost to the the large number of operations involved in creating the matrix prior to decomposing it.

The last equation certainly looks reminiscent of the equation we found using the Cholesky factorization, and there is a close connection. But this approach tends to give better results, especially in rank-deficient cases, as when A has less than full rank, the numerical stability of the computation becomes a much larger concern [526].

There are reduced-size versions of the decomposition that retain only the information we will need in a particular application. For example, if $m \gg n$ as in a typical least-squares curve-fitting case, we are retaining a lot of null rows at the bottom of R, and hence a lot of never-used entries of Q [306].

3.2.5.3 Applications of the QR decomposition

We're ready to return to the recommender system example. Recall our desire to "fuzz over" the distinctions between some movies, letting the data cluster related movies together.

We begin with the QR factors of W. (If you attempt to follow along in your favorite computing environment, recall that the QR decomposition is not unique without the addition of further constraints, so you may get slightly different results from what's shown below.) This is not a cheap computation for a large dataset, but the streaming service will have a massively parallel computing system to handle it.

In a QR decomposition of a matrix A, $rank(A)$ must always equal $rank(R)$. To implement the fuzzing over mentioned above, we will reduce the rank of A (or W) by reducing the rank of R. Consider again the normalized 7×6 matrix W. It will have rank less than 6 if and only if some diagonal entry is zero (or nearly so, because of the limitations of floating point arithmetic). In many computing environments, the diagonal entries of R will be arranged in descending order of magnitude, so the null entry would be $R_{6,6}$. In this case, the last two rows of R, rows 6 and 7, are entirely zero (again, up to round-off error). To reduce the rank of R, we make rows 4 and 5 be all zeroes also. We write R_{QRD} for the new, modified R, which now has its last four rows entirely null. It has rank 3. Now we will set

$$W_{QRD} = QR_{QRD} \tag{3.3}$$

to get an approximate, reduced-rank version of W.

The percentage error introduced, as measured by a certain norm on matrices (from Section 3.2.6), is about 43% in this case, which would be quite high in other circumstances. Here, however, we were intentionally seeking to discard some information in order to force the desired clustering, so this is actually a good thing. If we now repeat our earlier query using

$$z = \left(1/\sqrt{3}, 0, 0, 1/\sqrt{3}, 0, 0, 1/\sqrt{3}\right)^T$$

then $W_{QRD}^T z$ gives, after suppressing the negligible values and then rounding to two places,

$$(0.41, 0.58, 0.00, 0.50, 0.56, 0.00)^T,$$

indicating that viewer 2 (Bob) is the best match, closely followed by viewer 5 (Eric). Bob does share an interest in *King Kong* and *Macbeth*, and looking at Bob's viewing habits, we might still conclude that we should recommend *Hamlet* to the new user as well; this seems like a good recommendation, though no better than we had done before. Looking at Eric's record, we find *Godzilla* and *King Kong* in common, and might recommend *Lincoln*—a novel recommendation that is less obvious even to we who can reason about the underlying genres and themes of the films. It doesn't seem unreasonable that someone who likes *Hamlet* and *Macbeth* might be interested in the story of Abraham Lincoln; has reducing the rank uncovered this association?

It's usually pointless to ask whether a single, isolated finding like this is a success or merely a coincidence. Since the method has proven useful in practice, we make use of it. But we will see in the next section that there is a better, albeit more expensive, approach, requiring a different decomposition. We'll use the same term-by-document matrix, so if you're following along with the computations you may wish to save this example first.

Readers who wish to solidify their understanding of the matrix decompositions introduced so far may wish to try the project in Section 3.5.2 at the end of the chapter before we move on to another matrix decomposition method.

3.2.6 The singular value decomposition

In this section we're going to approach the same recommender system example via a different decomposition that is more expensive, but typically more revealing, than the QR decomposition. It's the last of the three big decompositions in applied linear algebra: the LU decomposition, the QR decomposition, and now the singular value decomposition (SVD). It's also the heart of the statsictical method known as principal components analysis, which is a major data science application in its own right, as will be seen in Chapter 7.

In traditional scientific computing applications, the LU decomposition usually has the starring role. Solving $Ax = b$ comes up on its own, in Newton's method, in least-squares problems in the guise of the Cholesky decomposition (as we saw earlier), in implicit finite difference methods, and so on. For applications in data science, the QR and SVD methods appear more frequently than might have been expected.

The singular value decomposition decomposes an $m \times n$ matrix A into the product of three factor matrices, in the form

$$A = U\Sigma V^T,$$

where U is an $m \times m$ orthogonal matrix (so that $U^TU = I$), V is an $n \times n$ orthogonal matrix (so that $V^TV = I$), and Σ is an $m \times n$ diagonal matrix with nonnegative diagonal entries, decreasing as one moves along the diagonal from the upper left to the lower right. The diagonal entry in the (i, i) position of Σ is called the i^{th} singular value σ_i of A. The columns of U are called the left singular vectors of A, and the columns of V are called its right singular vectors. When dealing with the SVD, it's important to note carefully when we are using V and when we are using V^T; we generally reason in terms of the columns of V but use the transpose in forming the actual product. An SVD is not unique without additional constraints placed on it; after all,

$$U\Sigma V^T = (-U)\Sigma(-V)^T$$

(and this is not the only possible way to get a different decomposition). This fact will not complicate our discussion, however.

A typical approach to finding the SVD computationally is to orthogonally reduce the given matrix to bidiagonal form—an (upper) bidiagonal matrix can be nonzero only on its main diagonal or on the first superdiagonal, that is, the first diagonal above the main one—and then use a specialized routine to find the SVD of the bidiagonal matrix. There are other algorithms to find the SVD; the details would take us too far afield, so see [192] for information on computing this decomposition.

There are many applications of the singular value decomposition. Orthogonal matrices are particularly well-behaved numerically, so solving problems using the SVD can often be the right approach for difficult cases—more expensive, yes, because of the initial cost of finding the SVD, but usually more accurate. In fact, for least-squares problems, rather than solving from the normal equation $A^TAx = A^Tb$ we can use some facts about orthogonal matrices to get a formula directly from the minimization criterion on the so called residual vector $b - Ax$ [526]. This is typically the most expensive, but more accurate, approach.

The rank of A is equal to the number of nonzero singular values of A, and this is generally the most precise way to determine rank numerically. If $rank(A) = k$, then the first k left singular vectors (from U) form an orthonormal basis for the column space of A, and the first k right singular vectors (from V) form, after transposition, an orthonormal basis for the row space of A. (Recall that these spaces have a common dimension, the rank of the matrix.) More to the point—and here we come to the important fact for our example—we can rewrite the decomposition $A = U\Sigma V^T$ in the form

$$A = \sum_{i=1}^{k} \sigma_i u_i v_i^T,$$

where u_i is column i of U, and v_i is column i of V. The k outer products

$$u_i v_i^T$$

are rank-one matrices. (For example, if u and v are vectors of all unit entries, the outer product is a matrix of all ones.) Since $\|u_i\| = 1$ and $\|v_i\| = 1$, and

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_k \geq 0$$

the expression

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

shows the contribution of the k rank-one matrices that form the rank-k matrix A, in descending order of importance.

An important result, the Eckart-Young Theorem, states that for any $p \leq k$, the approximation

$$A^{(p)} = \sum_{i=1}^{p} \sigma_i u_i v_i^T$$

is the best approximation of A among all similarly-sized matrices of rank at most p, in the sense that it minimizes the error in $A - A^{(p)}$ in the sense of the Frobenius norm

$$\|M\|_F = \left[\sum_{i=1}^{m}\sum_{i=1}^{n} m_{i,j}^2\right]^{1/2} = \left[\sum_{i=1}^{rank(M)} \sigma_i^2\right]^{1/2} .$$

We say that $A^{(p)}$ is a truncated SVD for A. More precisely,

$$\left\|A - A^{(p)}\right\|_F = \left[\sum_{i=p+1}^{k} \sigma_i^2\right]^{1/2} ,$$

which relates the error in the approximation of A to its singular values. This means that the error introduced by reducing the rank is based on the size of the singular values corresponding to the dropped dimensions, and leads directly to a useful formula: The relative error ρ of $A^{(p)}$ as an approximation for A satisfies

$$\rho^2 = \frac{\|A - A^{(p)}\|^2}{\|A\|^2} = \frac{\sigma_{p+1}^2 + \cdots + \sigma_k^2}{\sigma_1^2 + \cdots + \sigma_k^2},$$

and since the singular values are arranged in descending order, the numerator consists of the smallest $k-p$ of them. We might view this as a result in the style of Parseval's Theorem that tells us how much "energy" we lose by neglecting certain high-index, small-magnitude singular values, a loose analogue of high-frequency noise [70]. As an aside, the square of the Frobenius norm of a matrix is also equal to the trace of A^TA, $tr(A^TA)$, by which we mean the sum of the main diagonal entries of that matrix.

The truncated SVD is the best low-rank approximation of the matrix, with respect to this norm. This suggests that we revisit the viewers-and-movies example of the previous section but reduce the rank via the SVD instead. Such an approach is termed latent semantic indexing (LSI), derived from the latent semantic analysis (LSA) theory of natural language processing associated with psychologist Thomas Landauer (and others). Amongst other benefits, LSA can often identify, from a term-by-document matrix based on a sufficiently large corpus, when words are (near-)synonyms, e.g., divining that "car," "automobile," and "truck" all occur in similar contexts, so that a web search for "automobile" could return pages lacking that term but containing "truck" even if it didn't have explicit access to a thesaurus. In addition, when words have multiple meanings (that is, are polysemous), as in the word "bar," this approach can often group documents using these words into appropriate clusters representing their various contexts (documents using "bar" as in lawyer's qualifications in one cluster, "bar" as in metal rods in another, "bar" as in alcohol establishments in yet another, and "bar" as in wrestling holds in another). The reduction of rank induces such clustering of the terms and/or documents; filtering the semantic noise from the underlying signal of the actual meaning is usually beneficial here.

3.2.6.1 SVD in our recommender system

Let's look again at the term-by-document matrix of the previous section, with 7 movies as the rows and 6 viewers as the documents.

$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Let's decompose its normalized form W in the form $W = U\Sigma V^T$. How far should we reduce the rank? There's no easy answer. The singular values are, after rounding,

$$\sigma_1 = 1.7106, \sigma_2 = 1.3027, \sigma_3 = 0.9264, \sigma_4 = 0.6738, \sigma_5 = 0.2542, \sigma_6 = 0.$$

(In fact σ_6 was about 2.7079×10^{-18}.) Certainly setting σ_6 to zero was a reasonable thing to do, but if we're going to truly reduce the rank that will mean setting at least σ_5 to zero also. This would give

$$\rho^2 = \frac{\|W - W^{(4)}\|^2}{\|W\|^2} = \frac{\sigma_5^2 + \sigma_6^2}{\sigma_1^2 + \cdots + \sigma_6^2} \doteq 0.0108,$$

where the dotted equals sign indicates that the actual value has been correctly rounded to the number of places shown. Hence $\rho \doteq 0.11$, that is, per the Eckart-Young Theorem, we have thrown away about 11% of the "energy" content of the matrix. To better match what we did in the QR decomposition section, let's use $W^{(3)}$ instead. This rank-three approximation has

$$\rho^2 = \frac{\|W - W^{(3)}\|^2}{\|W\|^2} = \frac{\sigma_4^2 + \sigma_5^2 + \sigma_6^2}{\sigma_1^2 + \cdots + \sigma_6^2} \doteq 0.0864,$$

giving $\rho \doteq 0.29$, that is, we are retaining about 71% of the matrix as measured by the Frobenius norm (whatever that means in terms of viewers-and-movies). Hopefully this better reveals latent connections within it. (Using the Frobenius norm on the rank-three approximation previously found via the QR factorization, we find that $\rho \doteq 0.34$, for about 66% accuracy.) Then, using the same query as before,

$$z = \left(1/\sqrt{3}, 0, 0, 1/\sqrt{3}, 0, 0, 1/\sqrt{3}\right)^T,$$

we find that computing $\left(W^3\right)^T z$ gives, after suppressing small values and then rounding to two places,

$$(0.33, 0.58, 0.00, 0.43, 0.64, 0.00)^T,$$

indicating that viewer 5 (Eric) is now the best match, closely followed by best match from the QR approach of the last section, viewer 2 (Bob). The new user had selected *Godzilla*, *JFK*, and *Macbeth*, and Eric also selected the first two of these films; Bob had selected the latter two. With a small dataset like this, it's hard to know whether this result reflects any meaningful difference, but with a large dataset, the rank reduction will have forced a better refinement of Eric-like users from Bob-like users, and will attempt to cluster the new user in the best fit grouping, reflecting both likes and dislikes (whether recorded explicitly or inferred from what merely was not liked). We could then recommend other films preferred by those users.

The question of how far to reduce the rank does not have an easy answer. While a lower-rank approximate term-by-document matrix may better represent the latent semantic content of the original one, it's rarely clear how to decide what the best rank is. Often there is a range of ranks that all do a serviceable job, with too little clustering on the higher-rank side of this interval and too much on the lower-rank side; across a surprisingly broad range of applications, ranks of roughly 100-300 give good results [250]. Again, it's analogous to the problem of separating signal from noise: At some point, it becomes difficult to determine which is which. This is analogous to the ubiquitous problem of the bias-variance tradeoff in more statistical approaches to data analysis such as machine learning; see Chapters 4, 6, and 8 for more.

Let's demonstrate the clustering visually. To do so, the rank was further reduced to 2, giving a rank-two version of the W matrix. Its column space is therefore two-dimensional. Now, the first two columns of U form a basis for this column space, and the first two columns of V form a basis for its row space (after transposition). Letting U_2 be the corresponding 7×2 matrix and V_2 the corresponding 6×2 matrix, we have, from equation (3.2) on page 67, that $U_2^T v$ gives the coefficients of v with respect to the orthonormal basis consisting of the columns of U_2, assuming v is conformable with the transpose of U_2. This provides us a way to picture the clustering effect: We project down to the two-dimensional space by computing $U_2^T W$, giving a 2×7 matrix in which each column represents the two coordinates of the corresponding film in the two-dimensional space spanned by the two 7-vectors. Similarly, $V_2^T W^T$ gives a 2×6 matrix in which each column represents the two coordinates of the corresponding user in the two-dimensional space spanned by two 6-vectors. We could also project the query vector down into two dimensions.

The first column of $U_2^T W$ corresponds to the coordinates $(0.76, -0.50)$, so we plot film 1 with those coordinates. (Non-uniqueness of the SVD may mean that you get the opposite signs if you try this yourself.) The first column of $V_2^T W^T$ corresponds to the coordinates $(0.85, -0.08)$, so we plot user 1 with those coordinates. We'll call the axes P_1 and P_2 arbitrarily. This gives Figure 3.1, in which the films appear as boxed exes labeled F1 through F7, while the users appear as circled plusses labeled U1 through U6.

We see that films 3 and 5 (*Ishtar* and *King Kong*) are loosely grouped together, while the remaining films form a second cluster. Whatever we as

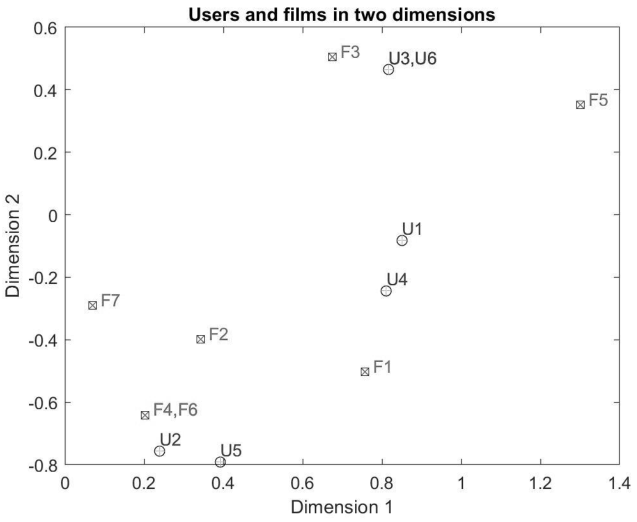


FIGURE 3.1: Rank-reduction using the SVD.

individuals may think of these films, the data-driven insight is that, at least in this set of users, these films share some factor(s) that cause them to draw interest from a certain subset of the population. Similarly, users 2 and 5, 1 and 4, and 3 and 6 appear to be clustered as like-minded souls.

At this point the question arises: Should we use the QR decomposition or the SVD? The standard LSA approach uses the SVD or one of its close variants. Depending on available software and hardware, however, and the specific structure of the term-by-document matrix, the QR decomposition may be noticeably faster. There is some indication that the QR approach is more sensitive to the choice of rank, making the SVD a safer approach. See [250] for more on all these points and a discussion of other trade-offs, as well as for further references.

When LSA is discussed, the implication is likely that the SVD will be used. As noted in the previous section, the QR decomposition is not only applicable to rank-reduction but also has separate applications in data science, including as an important algorithm for least-squares problems. We chose to demonstrate the QR and SVD approaches on a similar recommender system problem as a means of introducing, not limiting, the topics.

Readers who wish to solidify their understanding of SVD, including its relationship to the QR decomposition, may wish to try the project in Section 3.5.3 at the end of the chapter now.

3.2.6.2 Further reading on the SVD

In data analysis applications it's not uncommon to replace the SVD

$$A = U\Sigma V^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

with the semi-discrete decomposition (SDD)

$$A \approx XDY^T = \sum_{i=1}^{k} d_i x_i y_i^T,$$

where D is a diagonal matrix with diagonal entries d_i that are positive scalars, as with the SVD, but the entries of X and Y must be -1, 0, or 1. This approximation allows for a significant storage benefit [274], as we can represent the entries of the approximate versions of the large and typically dense matrices U and V, i.e., X and Y, using only 2 bits each.

There's a lot more to say about the SVD. One important interpretation of the singular values is in the context of the condition number of a square matrix with respect to inversion, a measure of how ill-behaved the matrix A is numerically when attempting to solve $Ax = b$. (We refer to the solution of $Ax = b$ as the inverse, or inversion, problem for A, with no implication that we will actually use the inverse to solve it. The forward problem is that of computing $y = Ax$ given A and x.) The condition number is generally defined as

$$\kappa(A) = \|A\| \left\|A^{-1}\right\|$$

for an invertible matrix and $\kappa(A) = \infty$ for a noninvertible matrix. The norm used may be any valid norm on matrices, such as the Frobenius norm mentioned earlier, but the most natural choice of matrix norm leads to the conclusion that

$$\kappa(A) = \frac{\sigma_1}{\sigma_r}, \tag{3.4}$$

where r is the rank of A. In other words, the condition number may be defined as the ratio of the largest to the smallest (nonzero) singular values of the matrix.

It's a crucially important rule of thumb in numerical analysis that if the condition number of A is about 10^p, then we should expect to lose about p significant figures in the solution of $Ax = b$, even if we are using a smart method. This is unsettling news; if the entries of b are known to only five places, and $\kappa(A) = 10^6$, even a go-to method like Gaussian elimination with partial or even the more expensive maximal pivoting can't be expected to give good results. The condition number is most commonly estimated by a less expensive method than using the singular values, but equation (3.4) defines what we usually mean by *the* condition number (called the spectral condition number). More details may be found in [306] at the introductory level, or [444], [445], and [192] at a more advanced level.

We mention in passing that extending the SVD to multilinear arrays is complicated. There is no natural generalization of this decomposition to three-way and higher arrays, as any choice keeps some highly desirable properties of the SVD but sacrifices others [6]. Nonetheless, the two major forms of generalized SVD are indeed applied to multilinear arrays in data science applications, and in fact, this is an active area of research; see, e.g., [452] for an introduction in the context of recommender systems.

The SVD has a rich geometric interpretation; it can be used to define a near-inverse for not-necessarily-square matrices that lack an actual inverse, and this so-called pseudoinverse does see application in data analysis (e.g., least-squares). The left singular vectors corresponding to nonzero singular values form a basis for the range of A, the set of all y such that $y = Ax$ for some conformable x, while the right singular vectors corresponding to nonzero singular values form a basis for the null space of A, that is, the set of all x such that $Ax = 0$. See [229] or [445] for details.

In fact, we are very nearly discussing the statistical data analysis technique of principal components analysis (PCA), important in areas including machine learning, because the singular value decomposition of A depends on properties of $A^T A$, the positive semi-definite matrix occurring in the normal equation $A^T Ax = A^T b$ of least-squares curve-fitting. This connection with the matrix $A^T A$ will be brought out in more detail in the next section, while PCA appears in Section 7.3.

3.3 Eigenvalues and eigenvectors

3.3.1 Eigenproblems

We now turn to the mathematics underlying Google's PageRank web-search algorithm, which we will explore in some detail. The same ideas have other, related applications, including within PCA. The foundational material of the next few sections will be familiar to many readers, who may feel free to skip to Section 3.3.4.

A central type of problem involving a matrix A is limited to square matrices: finding all scalars λ and corresponding nonzero vectors x, such that

$$Ax = \lambda x.$$

This is called the eigenvalue problem (or eigenproblem). The scalars λ are called the eigenvalues of the matrix, and the vectors x are called the eigenvectors. If $Ax = \lambda x$ we say that the eigenvector x is associated with the eigenvalue λ. We allow $\lambda \in \mathbb{C}$ even when, as in our cases, A is real. If $\lambda \in \mathbb{C}$, its associated eigenvectors will be complex also. Disallowing $x = 0$ is necessary because every λ would satisfy $Ax = \lambda x$ for $x = 0$, which is not very interesting.

It's possible to demonstrate in an elementary fashion that every matrix has at least one eigenvalue-eigenvector pair (an eigenpair); see [23]. In fact, every $n \times n$ matrix has precisely n eigenvalues in $\mathbb{C}$, counted in the same manner as we count the zeroes of polynomials; hence we associate with every square matrix the polynomial

$$p(z) = (z - \lambda_1)(z - \lambda_2) \cdots (z - \lambda_n),$$

which we call its characteristic polynomial, with $p(z) = 0$ its characteristic equation. For real matrices, $p(z)$ is real and hence complex eigenvalues occur in complex conjugate pairs. The algebraic multiplicity of the eigenvalue λ, denoted $\mu_a(\lambda)$, is its multiplicity as a root of the characteristic equation, and hence $1 \leq \mu_a(\lambda) \leq n$.

Note from $Ax = \lambda x$ that if x is an eigenvector of A associated with λ then so is cx for any nonzero c. It's easily shown that eigenvectors belonging to numerically distinct eigenvalues are linearly independent. The span of the set of all eigenvectors associated with an eigenvalue λ is called its eigenspace E_λ, and every element of this space is also an eigenvector save for the zero vector, which must be in every vector space and which satisfies the defining equation $A \cdot 0 = \lambda \cdot 0$ but is not technically an eigenvector. The dimension of E_λ is called the geometric multiplicity $\mu_g(\lambda)$ of the eigenvalue, and $1 \leq \mu_g(\lambda) \leq \mu_a(\lambda)$. Whenever $\mu_g(\lambda) < \mu_a(\lambda)$ we say that the eigenvalue λ is defective, and that the matrix itself is defective. Defective matrices play a complicating role in eigenproblems that is somewhat analogous to the case of singular matrices and the solution of linear systems.

As a simple demonstration of the issue, consider the 2×2 identity matrix I_2. Every nonzero vector is an eigenvector of I_2 associated with the eigenvalue 1, for

$$I_2 x = 1 \cdot x$$

for every x. The span of the set of all eigenvectors is thus $\mathbb{R}^2$, the entire space, and a possible basis for it is $\{e_1, e_2\}$, where by convention e_i is column i of the identity matrix (the order of which is usually apparent from context). We call $\{e_1, e_2, \ldots, e_n\}$, with the vectors drawn from I_n, the standard basis of $\mathbb{R}^n$. As this holds for *every* vector, there can be no other eigenvalues; and since every 2×2 matrix has precisely 2 eigenvalues, it must be that $\lambda_1 = 1$, $\lambda_2 = 1$, and so the characteristic polynomial is

$$p(\lambda) = (\lambda - 1)^2,$$

and $\mu_a(1) = 2$, $E_1 = \mathbb{R}^2$, and $\mu_g(1) = 2$. The matrix is nondefective, as $\mu_a(1) = \mu_g(1)$.

But now consider the canonical example of a defective matrix, the 2×2 Jordan block

$$J = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

and note that $Je_1 = 1 \cdot e_1$, so that at least one of the eigenvalues is $\lambda_1 = 1$. There must be a second eigenvalue (possibly also unity). Hence we seek a nonzero vector $x = (x_1, x_2)^T$ and a scalar λ such that

$$\begin{aligned}\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= \lambda\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ x_1 + x_2 &= \lambda x_1 \\ x_2 &= \lambda x_2,\end{aligned}$$

and we note from $x_2 = \lambda x_2$ that either $x_2 = 0$ or $\lambda = 1$. If $x_2 = 0$, x must be a multiple of e_1, and we have gained nothing new; by the linear independence of eigenvectors belonging to different eigenvalues, e_1 can't be an eigenvector associated with two numerically distinct eigenvalues. If $\lambda = 1$, $x_1 + x_2 = \lambda x_1$ forces $x_2 = 0$. There can be no eigenvectors that are not multiples of e_1, and they all correspond to the eigenvalue 1. Hence, it must be that the missing eigenvalue is $\lambda_2 = 1$. The matrix is defective: $\mu_a(1) = 2$ but $\mu_g(1) = 1$. The eigenspace is $E_1 = span(\{e_1\})$ (the x_1-axis).

For a nondefective matrix A, a basis for the entire space, $\mathbb{R}^n$, may be formed from the eigenvectors of A. Suppose the matrix has p distinct eigenvalues. We select a basis for each eigenspace and use these

$$\sum_{i=1}^{p} \mu_g(\lambda_i) = n$$

vectors to form the basis for the whole space, called an eigenbasis. For a defective matrix, we will only be able to span a subset of $\mathbb{R}^n$. (A subset of a vector space that is itself a vector space is called a subspace of the vector space.) Symmetric matrices are always nondefective, and have only real eigenvalues. Eigenbases are important in many applications; PCA is an example.

Let's write $x^{(i)}$ for some eigenvector associated with λ_i. For a nondefective matrix, we can write the relations $Ax^{(i)} = \lambda_i x^{(i)}$ as a single equation in the form

$$\begin{aligned} Ax^{(1)} + Ax^{(2)} + \cdots + Ax^{(n)} &= \lambda_1 x^{(1)} + \lambda_2 x^{(2)} + \cdots + \lambda_n x^{(n)} \\ AP &= PD, \qquad (3.5)\end{aligned}$$

where P is the matrix with the eigenvectors $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ as its columns, and D is the diagonal matrix with $\lambda_1, \lambda_2, \ldots, \lambda_n$ on the main diagonal, in matching order. (Note that because eigenvalues can be complex, there is no natural ordering of the eigenvalues.) The columns of P are linearly independent, and hence it is nonsingular; therefore we can write

$$A = PDP^{-1},$$

called the eigenvalue decomposition of A, or equivalently

$$D = P^{-1}AP,$$

called a diagonalization of A. Symmetric matrices are not only diagonalizable; they are orthogonally diagonalizable, that is, we can always choose to have P be an orthogonal matrix.

This concept brings us back to the singular value decomposition. The (nonzero) singular values of A are the square roots of the eigenvalues of the positive semi-definite matrix A^TA (or of AA^T). In fact, note that if $A = U\Sigma V^T$ then

$$\begin{aligned} A^TA &= \left(U\Sigma V^T\right)^T U\Sigma V^T \\ &= V\Sigma^T U^T U\Sigma V^T \\ &= V\left(\Sigma^T\Sigma\right)V^T, \end{aligned}$$

as $U^TU = I$. Since $\Sigma^T\Sigma$ is clearly diagonal, and the columns of V are linearly independent, if V is square this has the form of an eigenvalue decomposition of A^TA. More generally, it must be that the diagonal entries σ_i^2 are eigenvalues of A^TA with the columns of V as the corresponding eigenvectors, for

$$\begin{aligned} A^TA &= V\left(\Sigma^T\Sigma\right)V^T \\ \left(A^TA\right)V &= V\left(\Sigma^T\Sigma\right), \end{aligned}$$

using $V^TV = I$ and equation (3.5). Consideration of AA^T shows the columns of U are eigenvectors of it, associated once again with σ_i^2. Hence, the eigenvalue decomposition and the SVD are closely related, and this observation leads to other means of computing the SVD.

The decomposition $A = PDP^{-1}$ has wide applications. For example, we often need to perform iterative processes involving matrices—PageRank will provide an example—and the eigenvalues of A determine its long-term behavior under iteration. For,

$$\begin{aligned} A^m &= \left(PDP^{-1}\right)^m \\ &= PDP^{-1}PDP^{-1}\cdots PDP^{-1} \\ &= PD^mP^{-1} \end{aligned}$$

and D^m is simply the diagonal matrix with λ_i^m as its i^{th} diagonal entry. This can also be a convenient way to raise a matrix to a large power.

In fact, we can use it to analyze the behavior of large powers of A. Define the spectral radius $\rho(A)$ of a matrix by

$$\rho(A) = \max_{i=1}^{n}\left(|\lambda_i|\right),$$

i.e., its largest eigenvalue (in magnitude). We now see that if $\rho(A) < 1$ then $A^m \to 0$ (the zero matrix), while if $\rho(A) > 1$ then at least some entry of it grows without bound; the case $\rho(A) = 1$ requires a more careful analysis. This is the basis of the analysis of many iterative methods, as it means that the matrix iteration

$$v^{(k+1)} = Av^{(k)}$$

(with $v^{(0)}$ given, and presumably nonzero) converges to the zero vector (that is, is stable) if its spectral radius is less than unity—all eigenvalues lie within the unit circle—and diverges (that is, is unstable) if its spectral radius exceeds unity.

Although it is less relevant to data science, another example of the value of the eigenvalue decomposition as diagonalization may be more familiar to many readers, the solution of the linear system of ordinary differential equations

$$x' = Ax + f,$$

where $f(t)$ is a known vector and $x(t)$ is a vector to be found. If A is diagonalizable then we can proceed as follows.

$$\begin{aligned}
x' &= \left(PDP^{-1}\right)x + f \\
x' &= PDP^{-1}x + f \\
P^{-1}x' &= DP^{-1}x + P^{-1}f \\
\left(P^{-1}x\right)' &= D\left(P^{-1}x\right) + \left(P^{-1}f\right) \\
y' &= Dy + \phi,
\end{aligned}$$

where $y = P^{-1}x$ and $\phi = P^{-1}f$. But this is just a system of n decoupled first-order linear ODEs

$$\begin{aligned}
y_1' &= \lambda_1 y + \phi_1 \\
y_2' &= \lambda_2 y + \phi_2 \\
&\vdots \\
y_n' &= \lambda_n y + \phi_n,
\end{aligned}$$

easily handled by elementary means (the integrating factor). We can then find the desired solution as $x = Py$. There are more practical means of solving such a system numerically, but the insight provided by decoupling the system can often be valuable.

For a defective matrix, there is a next-best decomposition called the Jordan Normal (or Canonical) Form. The Jordan form is similar to the eigenvalue decomposition but the diagonal matrix D is replaced by an upper triangular matrix J which is "nearly diagonal."

Chapter 7 will address the popular technique of PCA, which relies on either an eigendecomposition or the SVD, depending on choice of computational approach. The eigenvalue decomposition also comes into play in model selection in Section 5.7.3. See [254] for an introduction to the subject.

3.3.2 Finding eigenvalues

For matrices small enough to be manipulated by hand, we can find the eigenvalues by finding the characteristic polynomial of the matrix directly.

Given a square matrix A with PLU decomposition $A = PLU$, its determinant $\det(A)$ is equal to the product of the diagonal entries of U if an even number of row interchanges were performed, and the negative of this quantity if an odd number were used. The notation $|A|$ is also commonly used for the determinant, but note that the determinant may be positive, negative, or zero. It's apparent that if A is already upper triangular then its determinant is just the product of its diagonal entries, as we can take P and L to be I. But beyond that, we see that $\det(A) = 0$ exactly when there is a null entry on the main diagonal of U, that is, if and only if A is singular. Hence, a matrix is nonsingular if and only if its determinant is nonzero. However, it must be emphasized that the determinant is a poor indicator of how well a matrix will behave numerically; for that, we should check the condition number.

This, or a similar operation with one of our other decompositions, is the primary means of computing the determinant numerically, if desired. The traditional approach to defining the determinant is based either on a combinatorial method [20] or the Laplace expansion method, e.g.,

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}.$$

(Note the use of vertical bars to indicate that we are computing the determinant of A, which is a scalar quantity.) For large matrices this is an unimaginably less efficient means of finding the determinant than using the LU decomposition. Recall that $\det(AB) = \det(A)\det(B)$, a matrix and its transpose have the same determinant, and that if Q is orthogonal, then $\det(Q) = \pm 1$.

Using the determinant we can give a direct formula for determining the characteristic polynomial of A by writing $Ax = \lambda x$ as

$$\begin{aligned} Ax - \lambda x &= 0 \\ Ax - \lambda I x &= 0 \\ (A - \lambda I)\, x &= 0, \end{aligned}$$

for which we seek a nonzero solution. As $x = 0$ is a solution, there must be more than one solution of this equation when λ is an eigenvalue; hence $A - \lambda I$ must be singular. The requirement that

$$\det(A - \lambda I) = 0$$

has the advantage of being a computational, rather than qualitative, one. Laplace expansion verifies that $\det(A - \lambda I)$ is a polynomial in λ of precise degree n; in fact, it's the previously defined characteristic polynomial of A.

Of course, by the time we reach $n = 5$ the characteristic polynomial is a degree 5 polynomial and hence its zeroes must be found numerically, leaving no advantage for this already expensive approach as a numerical method. Hence, eigenvalues must be found numerically by iterative methods. However, the expansion technique can be valuable for 2×2, 3×3, and perhaps even 4×4 cases, and does allow us to discover a variety of facts about eigenvalues, such as that the eigenvalues of an upper triangular matrix lie along its main diagonal, and that the determinant is the product of the eigenvalues,

$$\det(A) = \lambda_1 \lambda_2 \cdots \lambda_n.$$

(Similarly, note that the trace of a matrix is equal to the sum of its eigenvalues.)

Methods for finding all eigenvalues of a matrix numerically are often based on a reduction procedure involving a similarity: We say that two square matrices A and B are similar if they are related by

$$B = S^{-1}AS$$

for some nonsingular matrix S. The eigenvalue decomposition $A = PDP^{-1}$ is an example (with $S = P^{-1}$). Similar matrices have the same rank and the same characteristic polynomial (and hence the same eigenvalues and determinant). We reduce A to a simpler, similar form B for which the eigenvalues are more readily found. A classic method is the QR algorithm of Francis, which is equivalent to finding the QR decomposition $A = QR$ of A, then forming $A_1 = RQ$, then refactoring this as $A_1 = Q_1 R_1$ and forming $A_2 = R_1 Q_1$, factoring this, and so on. The A_k are similar to one another and converge, under appropriate assumptions, to an upper triangular matrix, so that the eigenvalues are displayed on the main diagonal. There are many other algorithms for finding all eigenvalues of a matrix, including ones that apply to special cases (e.g., symmetric matrices) [192].

3.3.3 The power method

If a matrix has an eigenvalue λ_1 such that $|\lambda_1|$ is strictly greater than $|\lambda_i|$, $i = 2, \ldots, n$, then we say that this is its dominant eigenvalue. In this case, $\rho(A) = |\lambda_1|$. A wide variety of applications require only the largest eigenvalue of a matrix, and we'll see in the next section that PageRank is among them. Fortunately, there is a convenient method for finding a dominant eigenvalue, the power method.

Suppose the matrix is diagonalizable. (This isn't necessary but does make the analysis easier.) Then there is an eigenbasis for the corresponding space, and hence any vector $v \in \mathbb{R}^n$ may be written in the form

$$v = \alpha_1 x^{(1)} + \alpha_2 x^{(2)} + \cdots + \alpha_n x^{(n)}$$

for some coefficients $\alpha_1, \alpha_2, \ldots, \alpha_n$. Here $x^{(1)}$ is the eigenvector associated with the dominant eigenvalue. We'll assume the eigenvectors have been normalized. Let v be nonzero and not an eigenvector of A, and suppose further that α_1 is nonzero. Then

$$\begin{aligned}
Av &= A\left(\alpha_1 x^{(1)} + \alpha_2 x^{(2)} + \cdots + \alpha_n x^{(n)}\right) \\
&= \alpha_1 A x^{(1)} + \alpha_2 A x^{(2)} + \cdots + \alpha_n A x^{(n)} \\
&= \alpha_1 \lambda_1 x^{(1)} + \alpha_2 \lambda_2 x^{(2)} + \cdots + \alpha_n \lambda_n x^{(n)} \\
A^2 v &= A\left(Av\right) \\
&= \alpha_1 \lambda_1 A x^{(1)} + \alpha_2 \lambda_2 A x^{(2)} + \cdots + \alpha_n \lambda_n A x^{(n)} \\
&= \alpha_1 \lambda_1^2 x^{(1)} + \alpha_2 \lambda_2^2 x^{(2)} + \cdots + \alpha_n \lambda_n^2 x^{(n)}
\end{aligned}$$

and in fact repeated multiplication by A shows that

$$A^m v = \alpha_1 \lambda_1^m x^{(1)} + \alpha_2 \lambda_2^m x^{(2)} + \cdots + \alpha_n \lambda_n^m x^{(n)},$$

which amounts to use of the observation that $Ax = \lambda x$ implies $A^m x = \lambda^m x$. We now rewrite as

$$A^m v = \alpha_1 \lambda_1^m \left[x^{(1)} + \frac{\alpha_2}{\alpha_1}\left(\frac{\lambda_2}{\lambda_1}\right)^m x^{(2)} + \cdots + \frac{\alpha_n}{\alpha_1}\left(\frac{\lambda_n}{\lambda_1}\right)^m x^{(n)}\right]$$

and it's clear from the dominance of λ_1 that as m grows larger, this looks increasingly like a multiple of $x^{(1)}$. In fact, for large m,

$$\begin{aligned}
A^m v &\approx \alpha_1 \lambda_1^m x^{(1)} \\
\left\|A^m v\right\| &\approx \left\|\alpha_1 \lambda_1^m x^{(1)}\right\| \\
&= \left|\alpha_1 \lambda_1^m\right| \left\|x^{(1)}\right\| \\
&= \left|\alpha_1\right| \left|\lambda_1^m\right|
\end{aligned}$$

because of the normalization of the eigenvectors. There are a number of ways to extract the desired eigenvalue, all based on the observation that

$$\frac{\left\|A^{m+1} v\right\|}{\left\|A^m v\right\|} \approx \frac{\left|\alpha_1\right| \left|\lambda_1^{m+1}\right|}{\left|\alpha_1\right| \left|\lambda_1^m\right|} = \left|\lambda_1\right|.$$

(Note that a dominant eigenvalue must be real and nonzero.) One approach takes v, the starting vector, to be an initial guess as to $x^{(1)}$, which has its largest entry (in magnitude) equal to 1; that is, in terms of the Holder p-norms on n-vectors,

$$\left\|v\right\|_p = \left[\sum_{i=1}^{n} \left|x_i\right|^p\right]^{1/p},$$

of which the best-known case is $p = 2$, the Euclidean norm, we use the limiting case $p = \infty$,

$$\left\|v\right\|_\infty = \max_{i=1}^{n}\left(\left|x_i\right|\right),$$

called the max (or maximum, or supremum) vector norm. (The case $p = 1$ is the taxicab norm.) If $\|v\|_\infty = 1$ then after forming $w^{(1)} = Av$ we can let ϖ_1 be equal to any entry of $w^{(1)}$ of maximum magnitude. Then

$$v^{(1)} = \frac{Av}{\varpi_1}$$

itself has unit norm with respect to the max norm. Repeating, we find $w^{(2)} = Av^{(1)} = A^2v$ and then let ϖ_2 be equal to any entry of $w^{(2)}$ with maximum magnitude. Then setting

$$v^{(2)} = \frac{Av^{(1)}}{\varpi_2} = \frac{A^2v}{\varpi_2\varpi_1}$$

ensures that $\left\|v^{(2)}\right\|_\infty = 1$. Under our assumptions, in the limit $\varpi_m \to \lambda_1$, the dominant eigenvalue, and $v^{(m)}$ tends to an associated eigenvector. This is the power method.

If we have an estimate of λ_1 then we can accelerate by a process known as shifting; even without such an estimate, the Rayleigh quotient

$$\rho^{(m)} = \frac{\left(v^{(m)}\right)^T A \left(v^{(m)}\right)}{\left(v^{(m)}\right)^T \left(v^{(m)}\right)}$$

converges to the eigenvalue significantly faster than does ϖ [192]. A variant of the method can be used to find the smallest eigenvalue of a nonsingular matrix [306].

The assumption that the eigenvectors have been normalized is actually immaterial. In principle the assumption that α_1 is nonzero is essential, but in practice even if $\alpha_1 = 0$ the presence of round-off error will introduce a small component—that is, a small α_1—in subsequent computations, so that this merely represents a slight delay in reaching an acceptable approximation. The speed of the method is governed by

$$\left|\frac{\lambda_2}{\lambda_1}\right| < 1,$$

where λ_2 is the next largest eigenvalue in magnitude. If this quantity is nearly unity, then convergence will be slow.

3.3.4 PageRank

The best-known of all web search algorithms, Google's PageRank, is built around the power method [72]. Imagine indexing the web. One option would be to create a term-by-document matrix. This method is widely used for more modest-sized collections of documents. But PageRank takes a different approach.

The key idea behind this algorithm is that the most valuable page may not be the one on which the searched-for term or its variants appear most frequently. Indeed, using methods based on frequency led to early search engine optimization efforts in which page owners packed their pages with endless repetitions of terms, often in hidden text that served merely to artificially inflate their rankings. Rather, the page recommended (via hyperlinks to it) by the greatest number of other pages may be the best choice. This also lends itself to abuse, as one could create hundreds of dummy web pages that merely serve to point to one's favored web page; for this reason, a measure of the trustworthiness of the linking sites is also required.

Let's suppose for the moment that we're searching a manageable-size web; perhaps a corpus of specialized pages internal to some database (on law, or science, or medicine, etc.). There are N pages, which we assume to be numbered from page 1 to page N. Let b_i be the number of incoming links to page i, called backlinks. The simplest approach would be to say the score of page i is b_i, the number of pages "recommending" the page. In an ideal web, this might be workable; if all recommendations are honest and accurate, then the page with the greatest number of incoming links is probably the most important and hence arguably the most valuable or authoritative, and the page containing your search term (or phrase) with the highest such score might be a good place for you to start your search. Of course, this might simply be an introductory, overview, summary, or directory page of some sort, but if so, then one of the next few links should be helpful.

Let's think about this web in terms of graph theory. The web pages themselves will be nodes and the hyperlinks will be edges between them, with a sense of direction that preserves the information concerning which is page is linking to which. (Hence this is a directed graph or digraph, because the edges are unidirectional.) For concreteness, imagine a six-page web as shown in Figure 3.2, although we typically think of the pages as point-like nodes (or vertices) and the links (or edges) as directed arcs (the shape and placing of which are immaterial). More formally, edges are thought of as ordered pairs of nodes.

For our purposes, we will disallow a page linking to itself. Also, we only count an incoming link to page i from page j once, even if page j actually links repeatedly to page i. Hence $0 \leq b_i \leq N-1$. If $b_i = 0$ then page i is unreachable by traversing the web; one could reach that page only by using its URL to go directly to it. We say a graph is (strongly) connected if it is possible to visit every node by traversing the edges, respecting their sense of direction, starting from any node of choice. There's no reason to believe this would be true of a given web.

A standard way of representing a graph with N nodes in matrix form is by its adjacency matrix (or incidence matrix), which is an $N \times N$ matrix with a 1 in the (i, j) entry if there is an edge from i to j, and a zero otherwise. (More generally, we could allow multiple edges from i to j, called a multigraph, and record the number of such edges in the matrix.) In our case, we have the

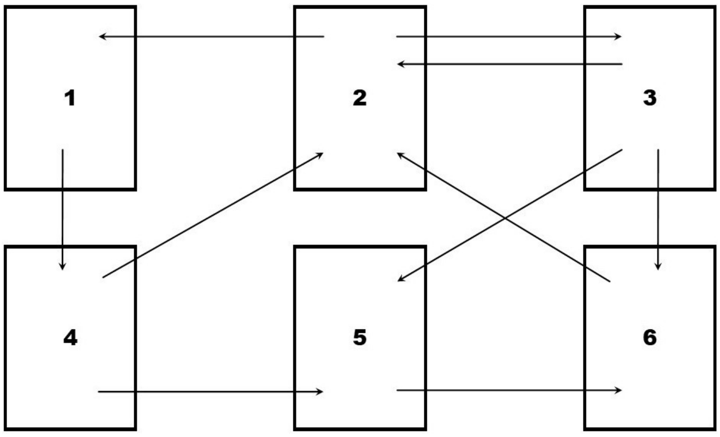


FIGURE 3.2: A six-page web, represented as a graph.

adjacency matrix

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Clearly, there must be more efficient ways to represent graphs, because this matrix will typically be very sparse.

The basic principle of a link-recommendation system is that the rank of a page—call it r_i for now, with a larger rank signifying a more valuable page—should be based on the pages linking to it. We've already said that simply setting r_i to b_i, the number of incoming links, is too simplistic. We want to weight the individual incoming links that comprise the total b_i by their own importance. Hence we might next try

$$r_i = \sum_{j \in S_i} r_j,$$

where S_i is the set of all j such that page j links to page i. This is a linear system to be solved; for our example, it gives

$$\begin{aligned} r_1 &= r_2 \\ r_2 &= r_3 + r_4 + r_6 \\ r_3 &= r_2 \\ r_4 &= r_1 \\ r_5 &= r_3 + r_4 \\ r_6 &= r_3 + r_5 \end{aligned}$$

or simply $r = A^T r$, with r the vector of the r_i. This is an eigenproblem; if a solution r exists, it's an eigenvector of A^T that is associated with a unit eigenvalue. But, in fact, there is no such solution. Though A (and hence A^T) is singular, the linear system

$$\left(A^T - I\right) r = 0$$

involves the nonsingular matrix $A^T - I$. But beyond that, looking at our web and the linear system involving the r_i, it seems apparent that, say, $r_1 = r_2$ might not really reflect the relative importance of the pages. We need to refine this approach. Although the basic adjacency matrix is very useful in many applications, we still need to find a way to factor in the relative trustworthiness (or authoritativeness, importance, or reputation) of the pages.

The PageRank algorithm uses this idea: Let ω_j represent the number of outgoing links from page j, again counting each page at most once (so that $0 \le \omega_j \le N - 1$). Then the weightings-influenced rank ρ_i of page i might be better defined by

$$\rho_i = \sum_{j \in S_i} \frac{\rho_j}{\omega_j}$$

so that if a given page $k \in S_i$ has only one outgoing link ($\omega_k = 1$) its recommendation is more heavily weighted than if it has, say, 10 outgoing links ($\omega_k = 10$). In our case this leads to the linear system

$$\begin{aligned} \rho_1 &= \frac{\rho_2}{2} \\ \rho_2 &= \frac{\rho_3}{3} + \frac{\rho_4}{2} + \frac{\rho_6}{1} \\ \rho_3 &= \frac{\rho_2}{2} \\ \rho_4 &= \frac{\rho_1}{1} \\ \rho_5 &= \frac{\rho_3}{3} + \frac{\rho_4}{2} \\ \rho_6 &= \frac{\rho_3}{3} + \frac{\rho_5}{1} \end{aligned}$$

or $\rho = P\rho$, where ρ is the vector of the ρ_i, and

$$P = \begin{pmatrix} 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/2 & 0 & 1 \\ 0 & 1/2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/2 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1 & 0 \end{pmatrix}$$

is the re-weighted A^T. Once again, we have the eigenproblem

$$P\rho = 1 \cdot \rho$$

in which we hope to find that P does, indeed, have a unit eigenvalue. In fact, it does, and this is its dominant eigenvalue; there is a null eigenvalue (as P is singular), and two complex conjugate pairs of eigenvalues, with moduli 0.76 and 0.66 (to two decimal places).

The eigenvector corresponding to the unit eigenvalue, normalized to have unit max norm by dividing it by its entry with largest magnitude, is

$$\rho = \begin{pmatrix} 1/2 \\ 1 \\ 1/2 \\ 1/2 \\ 5/12 \\ 7/12 \end{pmatrix} \doteq \begin{pmatrix} 0.500 \\ 1.000 \\ 0.500 \\ 0.500 \\ 0.417 \\ 0.583 \end{pmatrix},$$

indicating that the most important page is page 2, followed at some distance by page 6, which is followed fairly closely by pages 1, 2, and 4, and finally page 5. Looking at Figure 3.2, it certainly seems plausible to say that page 2 is the most-recommended page in the web, and that the others are grouped more-or-less together.

Looking at P, we notice that its columns sum to unity. A matrix with nonnegative entries is said to be a nonnegative matrix. A square nonnegative matrix with columns that sum to unity is called a column (or left) stochastic matrix; if the rows sum to unity instead it is said to be a row (or right) stochastic matrix. If both the rows and the columns sum to unity it is said to be a double stochastic matrix. When the type is clear from context, it is usual to simply describe the matrix as stochastic. We'll be using a column stochastic matrix.

It's a fact that a column stochastic matrix always has a unit eigenvalue, and that all non-unit eigenvalues of the matrix must have smaller magnitude. (The spectral radius of a stochastic matrix is unity.) However, note that I_n is a (doubly) stochastic matrix, and it has a unit eigenvalue with algebraic multiplicity n. Hence, the unit eigenvalue need not be unique.

It's also easy to see that if w is a row vector of all unit entries of the appropriate size, then $wP = w$ for any column stochastic matrix P; this is essentially just a restatement of the fact that the columns sum to unity. In other words, this vector is a left eigenvector of the matrix, as distinguished from the right eigenvectors we have seen already. For a row stochastic matrix, a column vector of all unit entries serves the same purpose. Indeed, whether we set up our problem to use row stochastic or column stochastic matrices makes little actual difference.

The approach we've been using isn't guaranteed to generate a stochastic matrix. If there is a web page with no outgoing links, we will have a column of zeroes in the matrix P. That isn't a stochastic matrix. However, if every web page has at least one outgoing link, P is guaranteed to be (column) stochastic.

Stochastic matrices have many desirable properties, but it's even better if the matrix is not merely nonnegative but actually a positive matrix, that is, one with all positive entries. The Perron-Frobenius Theorem states that a square positive matrix has a dominant eigenvalue that is a positive number. (Recall that a dominant eigenvalue is necessarily unique, by definition.) This eigenvalue admits a (right) eigenvector with entirely positive entries (and also a left eigenvector with positive entries), and all eigenvectors belonging to other eigenvalues must have negative or complex entries.

This means that a *positive* stochastic matrix must have a unique dominant eigenvalue, meaning it has a sole eigenvector associated with it that has positive entries and is normalized to have unit max norm. (This is sometimes referred to as the stationary vector.) This would mean an unambiguous ranking for our web-search algorithm. Hence the basic PageRank matrix P is usually modified by combining it with a small amount of "noise" to force it to be positive, as well as stochastic (by normalizing the columns after the noise is added). Effectively, this forces the graph to be connected (no "dangling nodes"). A weighted average of P and a matrix with each entry $1/n$ becomes the positive stochastic matrix used in the algorithm.

The PageRank method is implemented using a version of the power method. Naturally, the scale of the computation is truly massive! It's always important to remember that the way a concept is developed in linear algebra for the purposes of establishing and understanding results may be very different from how it is actually implemented as a computational algorithm.

There's a much more probabilistic way of viewing what PageRank does. Stochastic matrices are related to Markov chains and hence to random walks (i.e., a random web-surfer model), and these concepts shed additional light on the algorithm [302].

Of course, there are many other factors involved in a commercial web search engine than merely finding the desired eigenvector, involving tweaks and kluges to promote advertised products, to remove spam, to minimize attempts to game the system by search engine optimization techniques, etc. This section is intended only to indicate how the fundamental linear algebra concept of eigenvalues and eigenvectors might occur in a data analysis situation.

This is not the only recommendations- and eigenvector-based data analysis algorithm. See [36] or [302] for the somewhat-similar Hyperlink Induced Topic Search (HITS) method, and [158] for the eigenvector centrality method of identifying the central nodes of a network (i.e., the most highly recommended, or most prestigious, nodes of the graph).

3.4 Numerical computing

3.4.1 Floating point computing

Almost all computers implement the IEEE Standard for Floating-Point Arithmetic (IEEE 754) [376], which allows for predictable behavior of software across a wide variety of hardware types. While many applications in data science use primarily integer data, the linear algebra applications we have been considering in this chapter require the use of non-integral values. These are stored in a format called floating point, as distinguished from a fixed-point format such as the so-called (base 10) bank format in which exactly two digits after the decimal place are used. The purpose of this section is to provide a brief guide to some additional facts and issues involved in numerical computing so that you will be more aware of potential pitfalls and better equipped to work around them. The software you use will in most cases make the majority of this seamless for you, however, and so this section is optional reading.

3.4.2 Floating point arithmetic

Most programming languages will require, or at least allow, the programmer to specify a data type for each variable, such as a single integer, a vector of floating point numbers, or an $m \times n$ array with complex entries.

Integers are typically stored in a different manner than nonintegers. The most common integer format is the long integer, which uses 32 bits (4 bytes). Using a scheme called two's complement, a typical long integer format stores nonnegative integers from 0 to $2^{31} - 1$ in base two in the obvious way, while negative integers m are stored as $2^{32} + m$ $(-2^{31} \leq m \leq -1)$. There are commonly other integer formats, such as short integer, which stores numbers up to about 2^{15} in magnitude using half the space. There is also frequently a Boolean or logical data type that takes on only two values, which may be 1 and 0 but which may also be categorical values such as `true` and `false`.

The properties of integer data types may vary with hardware and software. Unpredictable behavior can sometimes happen if the program causes an overflow by attempting to store a too-large integer.

For real nonintegers, floating point arithmetic is employed. It's unlikely that you would find yourself using a machine that is not compliant with the

IEEE Standard for Floating-Point Arithmetic, so we discuss only that. Double precision floating point numbers are stored using 64 bits (8 bytes) divided into one bit representing the sign—unlike the scheme discussed above for integers—52 bits for a mantissa, and 11 bits for an exponent, giving a number of the form

$$\pm b_0.b_1b_2\cdots b_{52}\times 2^E,$$

where b_1 through b_{52} are the stored mantissa bits, and E is the exponent, which is specified by the 11 exponent bits with a built-in bias to allow for numbers both less than and greater than one in magnitude. By convention, b_0 is almost always unity; this is called normalization and allows us to gain one extra bit of significance. The finite set of numbers represented by this scheme range from about 10^{-308} to 10^{308} in magnitude, as well as 0, with 53 significant bits—about 15 or 16 significant digits—available at all exponent ranges. This is important: The spacing between the double precision floating point number $x = 1$ and the next largest double precision floating point number is about 10^{-16}, but the spacing between the double precision floating point number $x = 10^{300}$ and the next largest double precision floating point number is nearly 10^{284}. By design, floating point arithmetic provides a consistent number of significant bits of precision across a wide range of exponents, but this means that the spacing between adjacent floating point numbers grows as the numbers themselves grow. The gap size is governed by a value known as the machine epsilon, here $\epsilon = 2^{-52}$; the floating point numbers 1 and $1+\epsilon$ differ, but 1 and $1+\epsilon/2$ do not; the latter rounds down to 1.

Setting all 11 exponent bits to ones is a signal that the number should be interpreted a special way: $+\infty$, $-\infty$, or NaN ("not a number," e.g., the result of attempting the operation 0/0). Although these special values are available via the hardware, some languages (e.g., MATLAB) will allow you to work with them, while others may signal an error. Values too large in magnitude may overflow, resulting in a warning, error, or result of $\pm\infty$; values too small in magnitude may underflow, being set to zero or possibly to an unnormalized (subnormal) number with fewer significant bits.

The standard guarantees accurate computation of addition, subtraction, multiplication, and division. It specifies several available rounding modes; in fact, a technique called interval arithmetic computes a value like $z = x+y$ with both rounding down and with rounding up, giving a result $[z_{lower}, z_{upper}]$ that is guaranteed to bracket the unknown true value of z. This is rarely necessary, however, and we must learn to accept that floating point computations involve round-off error due to the finite amount of precision available. Even attempting to enter $x = \pi$, say, causes representation error when the value is stored to memory.

Single precision uses 1 bit for the sign, 23 bits for the mantissa, and 8 bits for the exponent; the resulting values range from about 10^{-38} to 10^{38} in value. Note that double precision is, in fact, a little more than double the precision of single precision. Single precision is sufficient for many purposes and uses half the storage.

There are both higher and lower precision options that may be available. Formats half the size of single precision are increasingly common in use, saving storage, time, and power usage, and requirements to lower the power used by a computation are surprisingly frequent.

Floating point arithmetic behaves differently from real arithmetic in several important ways. We have already seen that $1 + x$ can equal 1 even for a perfectly valid (normal) floating point number x (e.g., $x = \epsilon/2$). Similarly, x/y can be zero even when x and y are normal, positive floating point numbers (e.g., $x = 10^{-200}$ and $y = 10^{200}$). But in practice, perhaps the most common concern is the failure of associativity. For example,

$$\begin{aligned} 1 + \left(-1 - \frac{\epsilon}{2}\right) &= 0 \\ (1 + -1) - \frac{\epsilon}{2} &= -\frac{\epsilon}{2}. \end{aligned}$$

While this is a contrived example, the fact that the behavior of a formula depends on how it is coded is actually a common difficulty.

Two last concerns related to floating point arithmetic. Subtraction of nearly equal numbers is perhaps the gravest problem encountered in machine arithmetic. Imagine a base-10 device capable of representing 6 digits. Let $x = 3.14159$ and $y = 3.14158$. Then

$$z = x - y = 0.00001,$$

but while x and y had 6 digits of accuracy, z has only 1! Any further computations done with z will suffer this same problem; we have lost essentially all significance in our calculations by cancelling the common digits. A related problem is division by very small magnitude numbers, which tends to inflate any error already present in the numerator. Hence, for example, a computation like

$$\frac{f(x+h) - f(x)}{h}$$

for small $h > 0$ risks forming a wildly inaccurate numerator, then exaggerating that effect upon dividing by $h \ll 1$. Care must be taken when performing such calculations; ideally, alternative approaches can be found.

3.4.3 Further reading

The linear algebra material throughout this chapter may be found in most introductory texts on the subject; [20] provides an elementary look at the material, while [230], [229], [388], and [297] are higher-level references. An upcoming text by a coauthor of another chapter of this text, [270], promises to cover the subject of linear algebra for data science in much greater detail.

For computational matrix algebra and for general matters of scientific computation, the author's text [306] provides additional detail at an advanced undergraduate level. An excellent and very readable book on computational matrix algebra at the beginning graduate level is [470], while the classic reference on the subject is still [192].

The SVD application draws from the author's experience as co-author of a computational phylogenetics paper [177] using the LSI approach; the article [344] sets the stage for most of these discussions (and the similar book [36], which includes an introductory look at PageRank). These are also key sources for the rank-reduction methods in the QR and SVD sections. A good reference to begin a deeper look at vector information retrieval methods is [407]; [158] covers a range of methods for textual data. For the motivation behind LSA, [464] remains worthwhile reading.

3.5 Projects

3.5.1 Creating a database

It's commonplace among people working with data that much more time is spent on managing the data than on analyzing it. This has certainly been the author's experience. When assigning projects related to this material in general computational science and in computational data science courses, the first instruction to the students is to assemble a dataset. Sometimes this is weather data from a National Oceanic and Atmospheric Administration (NOAA) website, sometimes it's genetic data from the National Center for Biotechnology Information (NCBI), and sometimes it's a corpus of book titles or text from, say, Project Gutenberg. Rather than providing step-by-step instruction, the students are generally provided with the name of a web repository of data such as one of these three, a description of the type of data to be collected, and then set loose to devise the best way to obtain the needed information and to organize it for further use.

It's common that such a task will require multiple software programs. Data may be made available in a spreadsheet that must then be loaded into a separate scientific computing package. Or the data may be poorly organized and it may be necessary to write a script to collect and store it before porting it elsewhere for processing. Data format conversion still takes up an unfortunate amount of data wranglers' time.

So this first project, on vector information retrieval, is not much more than this: Assemble a corpus of text by selecting perhaps 300 titles from, say, Project Gutenberg, choosing around 30 books from each of 10 genres. Take perhaps the first 2000 words (after the boilerplate text) from each document. This is already a scripting problem! Next, form a term-by-document matrix with words as rows and titles as columns, with a_{ij} the number of occurrences of term i in book j. Incorporate stemming; perhaps if two terms agree to the first 6 letters, they are to be lumped together. Call this matrix A. Then weight the entries of the matrix in whatever way seems reasonable to you, such as making it boolean, normalizing by rows or columns, etc. This is the matrix W that we used in the QR Decomposition and SVD sections. We will use it in the following projects.

3.5.2 The QR decomposition and query-matching

Form a query q based on several words you expect to correspond to a certain genre and normalize it to the vector z we used previously. Query the database using $W^T z$. Does your query result in a useful recommendation of a book title?

Presumably the number of distinct terms you found significantly exceeds the number of titles, so your weighted term-by-document matrix has rank 300 or less. Find the QR factors of your matrix and reduce the rank per equation (3.3). It's far from clear how much the rank should be reduced. Look at the diagonal entries of R and find a natural breakpoint in their magnitudes, but reduce the rank by at least 50%. Query the database using this reduced-rank matrix. Does your query result in a useful recommendation of a book title? Experiment with different degrees of rank-reduction and different queries. Does rank-reduction result in better results?

3.5.3 The SVD and latent semantic indexing

You can repeat the rank-reduction query-matching experiment from above using the SVD, but you can also do some clustering based on latent semantic indexing. Mimic Figure 3.1 by reducing your data to rank 2, that is, to a two-dimensional space. Try to interpret the data; do the clusters formed seem natural or intuitive?

Try again with rank 3. Visualize your results as three-dimensional plots. Do these clusters seem more appropriate? Had reducing to rank 2 caused too many disparate items to be forced to caucus together, or has only going to rank 3 failed to remove sufficient noise?

This is a good opportunity to try different weighting systems for your basic term-by-document matrix. Compare your results across the raw matrix A of integer frequencies, its Boolean all-or-nothing counterpart, and the column-normalized version. Do you see noticeable differences?

3.5.4 Searching a web

Indexing a web relies on specialized software. To mimic the adjacency matrix of a plausible web, first create a square 100×100 matrix A of zeroes. Let's suppose that page 42 is the most important; randomly place into column $j = 42$ a scattering of 30 unit entries. Column 42 now has 70 null entries and 30 unit entries, randomly placed throughout it. Maybe column $j = 22$ is also highly recommended; randomly fill column $j = 22$ with, let's say, 20 unit entries. Then randomly fill the remaining columns with a random number of unit entries, between 5 and 15 (inclusive) per column. Remember that the diagonal entries should all be null. Now normalize the columns to make this a column stochastic matrix P.

As mentioned in the PageRank discussion, in order to assure connectivity so that we can benefit from the assertions of the Perron-Frobenius Theorem we replace P by a weighted average of two matrices, say $K = (1-m)P + mJ$, where J is a square matrix of the appropriate order with all entries set to $1/n$ (in our case, 1/100). As long as $0 \leq m \leq 1$, the matrix K is column-stochastic, and if $m > 0$ then it is also a positive matrix. Hence it has a dominant unit eigenvalue. Let's take $m = 0.15$ [72].

For a very large web the eigenvector would likely be found by some variation of the power method: Pick a vector $v^{(0)}$ that is a good guess at an eigenvector for the dominant eigenvalue if possible; if not, pick any nonzero vector. Let's use a vector of all unit entries. Then iterate

$$v^{(k)} = \frac{Kv^{(k-1)}}{\varpi_k},$$

where ϖ_k is equal to the max norm of the numerator. Then $v^{(k)}$ converges to the stationary vector as $k \to \infty$.

Either use the power method or your computing environment's eigenvalue/eigenvector routine to find the stationary vector. Make sure it's appropriately normalized, with the largest entry in the vector $+1$. What page does that correspond to in your web? What about the next largest value? Is this method identifying highly-recommended pages? Re-randomize A and repeat this experiment several times.

Don't expect that pages 42 and 22 will necessarily be the highest-scoring. In fact, in 6 runs with pseudo-randomly generated data, page 42 came to the top only once, and typically didn't make the top five. The method does not simply identify the page with the most incoming links. It simultaneously assesses the importance of all the pages. Looking at the entries of the original adjacency matrix A, does it appear that other important pages are linking to your top results?

Chapter 4

Basic Statistics

David White

Denison University

4.1 Introduction

The field of statistics is often described as "the science of turning data into information," a phrase which could be used just as well to describe all of data science. Section 1.3 covered the debate about the precise differences between data science and statistics, but there is widespread agreement that statistics is an essential piece of data science [132]. As discussed in [486], the best practices for an undergraduate curriculum in data science would include an introduction to statistical models. In this chapter, we will describe the essential topics in such an introductory course, advanced topics that could go into a second or third course in statistics, and a few tips regarding best practices for teaching such a course.

Because this chapter suggests nearly three courses worth of material, it is not essential to read it in its entirety before reading later chapters in the text. The reader should cover the essentials of a first course in statistics in Sections 4.2 through 4.5 and read about related pedagogy and connections to other data science topics in Section 4.10. Naturally, you can return to the more advanced material in Sections 4.6 through 4.9 as you find the need.

The choice of topics for this chapter is based on curricular guidelines published by statistical associations [13, 89, 372], and the author's experience teaching statistics at Denison University, where these topics are spread across two courses. Syllabi and links to course materials may be found at the author's web page (a link to which appears on the site for this text [84]), and a description of how a mathematician can develop and teach such courses may be found in [498]. Over the years, the author has taught a number of colleagues how to teach statistics, and a few patterns have emerged.

First, many mathematicians start out with a negative impression of the field of statistics. Some believe statistics is basically a subfield of arithmetic, because many formulas used in statistics are elementary. Others have negative memories from statistics courses they took as undergraduates that were simultaneously boring and lacking in rigor (giving intuitive justifications for important theoretical results, rather than rigorous proofs). Second, many find the shift from deductive reasoning to inductive reasoning counterintuitive, and long for the comfort of axioms. Mathematical culture leads us to abstract away the context from the problems we face, but that context is essential for doing applied statistics. Third, many in society inherently mistrust statistics, because they have seen statistics used to make arguments both for and against the same idea, e.g., whether or not tax cuts stimulate economic growth. Worse, many papers that use statistical arguments cannot be replicated on other datasets. (We discuss how to change this in Section 4.5.5.) Surely, these were the kinds of considerations Mark Twain had in mind when he popularized the phrase, “There are three kinds of lies: lies, damned lies, and statistics.”

A large part of this negative impression is due to how statistics was commonly taught when today’s professors were students, how statistics is presented in the media, and statistical illiteracy in society at large. Thankfully, over the past decade, statistical pedagogy has undergone a renaissance, and the world has become awash with freely available data on almost every topic imaginable. It is now an extremely exciting time to study and teach statistics, as statistical software removes the need for rote, hand calculations, facilitates the creation of engaging graphical visualizations, and allows us to focus statistics courses around real-world applications and conceptual understanding, rather than derivations of statistical formulas. Thanks to computational automation, a modern statistics course is free to focus deeply on when a given statistical model is an appropriate fit to the data, how to carry out a good statistical analysis, and how to interpret the results in context. While many non-experts have in the past used statistics to create flawed arguments, this is not a failing of statistics as a field. Society is improving in our ability to detect bad statistics, and journal publishing culture is changing to force authors into the habits of good statistics, to combat the replication crisis (Section 4.5.5).

Now is an excellent time for a mathematician to get involved with research and teaching in statistics. Many of the things we love about mathematics are inherently present in statistics, including logical reasoning, creative problem-solving, cutting to the core of a problem, quantitative intuition, and philosophical questions requiring careful analytical thought. Additionally, many of the tools of statistics are familiar to mathematicians, including sets, functions, polynomials, matrices, dot products, probabilistic reasoning, and algorithms. Every year, more and more mathematical concepts are used in creating statistical models (e.g., the trees and graphs used in network analysis and machine learning, as in Chapters 3 and 8, or techniques from homological algebra used in topological data analysis, as in Chapter 10), and at the moment, a mathematician trained in the basics of statistics is in a position to introduce new models that may quickly gain traction and application.

It is an exciting time to learn the tools of applied statistics, as these tools open doors to new and interdisciplinary collaborations. Applied statistics is an excellent source of projects suitable for student co-authors, and the volume and variety of available data makes it easy to find intrinsically interesting research topics in applied statistics. Furthermore, data science is desperately in need of more mathematicians: trained to be methodical and meticulous, and with the technical background to actually understand statistical tools, rather than using them as a black box. If we were to enrich our teaching with statistical concepts, we could improve the state of statistical literacy, and could do our part to restore public confidence in statistics. Research teams conducting bad statistical analyses often do not have a trained statistician or mathematician, and having more mathematicians doing statistics and data science would help combat this problem. A common thread in bad statistics is a failure to check the assumptions required by the mathematical results statistics is based on, or failing to verify the preconditions for statistical algorithms. As mathematicians are well-practiced in checking assumptions, we are in a position to improve statistical analyses when working alongside non-experts. Crucially, we are in a position to salvage analyses of datasets where the assumptions are not satisfied, by introducing more advanced models and a more careful approach.

This chapter aims to teach the essentials of statistics to an audience of mathematicians, with their particular abilities and preferences in mind. Since mathematicians already possess a strong technical background, we are able to move quickly through certain topics, and to emphasize other topics that are sometimes difficult to grasp. We devote particular attention to the statistical analysis of datasets that fail to satisfy the standard assumptions of statistics (Section 4.6.4). No specific background knowledge is assumed of the reader. Additionally, because statistics is being taught so differently today than a generation ago, we have interleaved a discussion about best practices for teaching these topics to students.

Throughout, the essential idea for teaching a truly applied first course in statistics is to de-emphasize unnecessary theory, and to give students as many opportunities as possible to apply statistics to real-world datasets. This can be a challenge for many mathematicians, as it involves truly embracing the shift from deductive reasoning to inductive reasoning, and because many of us never took a truly applied statistics course. Statistics curricular reform is a relatively recent phenomenon, so the curricular guidelines [13, 89, 372] often bear little resemblance to exposures today's mathematics faculty might have had to statistics as undergraduates. However, research has also shown that mathematical statistics courses, with their emphasis on probability theory and the theoretical properties of probability distributions, do not work for the majority of students [352]. Furthermore, the statisticians are clear in their recommendations: students should be given opportunities to work with real data, to build and test statistical models using computational software, to learn to check the conditions for these models, and to think about what to do when the conditions fail [13].

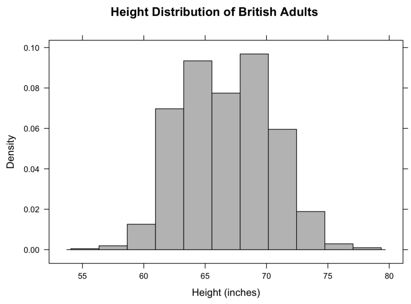
As it is impossible to cover two semesters' worth of content in a single chapter, certain topics are touched on only lightly. Our goal is to structure the conceptual core of statistics for the reader, to emphasize certain approaches that may appeal to a mathematician, to introduce key terminology, and to provide references where the reader can learn more. A standard introduction to statistics includes exploratory data analysis, visualizations, an overview of common statistical models, a dash of probability theory, simulation and resampling, confidence intervals, inference, linear models, and causality. We discuss these topics below, providing examples, definitions, and teaching tips for each topic. For the sake of brevity, we move more quickly through elementary topics.

4.2 Exploratory data analysis and visualizations

One of the most powerful things we can do with data—even before getting into building statistical models, analyzing variability, and conducting statistical inference—is create graphs summarizing the data. This is also a natural place to begin a first course in statistics: by teaching students how to explore a real-world dataset and create **visualizations** using statistical software, in line with the GAISE recommendations [13]. Figure 4.1 shows many basic visualizations typically encountered in a first course in statistics, all of which can be created using simple commands in statistical software. Our discussion below focuses primarily on **quantitative variables**, that is, variables taking numerical values representing some kind of measurement. Examples include height, age, salary, years of education, etc. Variables taking values from a finite set, like hair color, race, or gender, are called **categorical variables**.

A **histogram** (Figure 4.1a) displays a finite set of data points for a quantitative variable X (e.g., salary), by dividing the range of X into a fixed number of discrete bins, and counting the number of individuals in the dataset whose value for X is in each bin. Each bin is represented by a bar, whose height represents the number of data points in the bin. This same information can be represented by a **stem and leaf plot** (Figure 4.1b), which is essentially a histogram rotated by 90 degrees. Each row represents a bin, and the number of numbers in each row represents the count of individuals in that bin. We call X **symmetric** when its histogram exhibits rough symmetry around its center (the average value of X). When a histogram fails to be symmetric (e.g., if bar heights on the left side are much larger than bar heights on the right side), we call X **skew**.

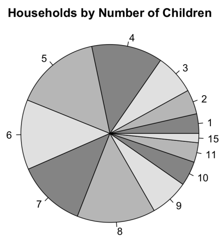
If X is categorical rather than quantitative, then the chart analogous to a histogram (but where the bins now represent the categories of X, and white space is inserted between the bars) is called a **bar chart**. For example, if X represents highest degree earned, with levels of PhD, master's, bachelor's, high school, and other, then a bar chart would have five bars, with heights corresponding to the number of individuals in the dataset of each degree type. Another way to represent the proportion of individuals of each category (as a



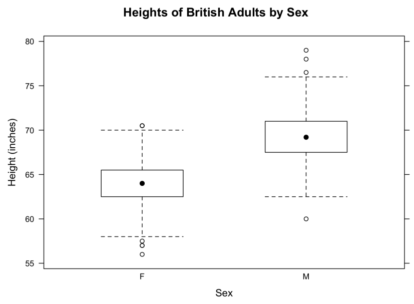
(a) Histogram

```
0 | 9
1 | 89
2 | 245
3 | 0033566778
4 | 11233477888
5 | 23389
6 | 000056668
7 | 002689
8 | 79
9 | 4
```

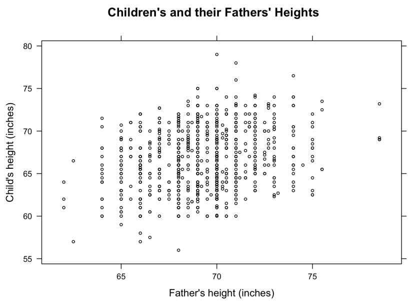
(b) Stem and Leaf Plot



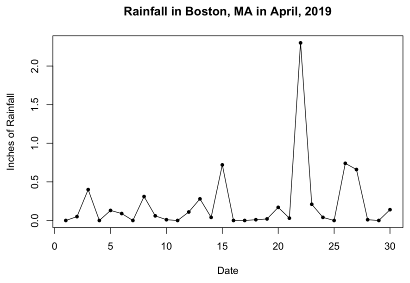
(c) Pie Chart



(d) Box and Whisker Plot



(e) Scatterplot



(f) Line Chart

FIGURE 4.1: Basic visualizations. The data for several of these visualizations comes from the Galton dataset built into R, as discussed in the text.

fraction of the total number of individuals) is via a **pie chart** (Figure 4.1c). In a pie chart, counts are represented as fractions of a circle, so that a histogram with three equal height bars is equivalent to a circle divided into three equal slices. It is easier for the human brain to distinguish heights of bars than it is to distinguish areas of slices of a circle [256], so bar charts are usually recommended over pie charts when the goal is a comparison of different levels.

A **box and whisker plot** (Figure 4.1d, also known as a box plot) is used to represent the distribution of a quantitative variable X and its outliers. The center dot of the box is the median, the box itself is the span between the first and third quartile,[1] and each whisker is a distance of $1.5IQR$ from the edges of the box. A value of X outside of the whiskers is defined to be an **outlier**.

A **scatterplot** (Figure 4.1e) is used to show the relationship between two quantitative variables. For example, if we have a dataset of children, where X is the age of a child, and Y is the child's shoe size, then each point (x, y) in a scatterplot represents a child (of age x and shoe size y), and the scatterplot as a whole shows us the trend between the age of children and their shoe sizes. Scatterplots are an essential tool for linear regression, as we will see in Section 4.3.1. A **line chart** (Figure 4.1f) is like a scatterplot, but where there are no duplicate X values, and where each point is connected to the next point by a line segment. Line charts are most often used when X represents time. In this situation, they show how Y changes over time.

Visualizations allow us to see trends (e.g., between years of education and salary), skewness (e.g., the wealth distribution), and outliers. Visualization serves as a powerful "hook" to get students on board with the inductive nature of a statistics course. Students often appreciate a discussion about how visualizations can be used to mislead. Such a discussion gives students practice developing skepticism, looking carefully at axes in graphics they are shown, and thinking about the importance of the choice of the population under study.

Common ways that visualizations are used to mislead include starting the y-axis in a bar chart at a value other than zero (to make the differences in bar heights look more significant), plotting only a small range of the data (e.g., showing sales going down in the winter months, but hiding that sales went back up in the summer months), leaving out data points in a scatterplot to fit a narrative, manipulating the y-axis in a line chart to make change appear more significant, cherry-picking the population (e.g., reporting opinions of one political party without reporting the proportion of the population in that party), or flipping the y-axis in a line chart (so that 0 is in the upper left corner rather than the lower left corner) so that it reverses the meaning of the chart.

The basic visualizations from Figure 4.1 are far from a comprehensive coverage of data visualization. Indeed, even a semester-long course can only scratch the surface of data visualization techniques, and with modern software, the only limit is creativity. Figure 4.2 shows a few examples of advanced visualizations. A **density plot** (Figure 4.2a) shows the density of the values of a quantitative variable X, and so is like a histogram, but without the need for arbitrary divisions into bins. This can combat binning bias, i.e., the

[1]The first quartile, $Q1$, is the smallest value of X larger than 25% of the values of X. The third quartile, $Q3$, is larger than 75% of the values of X. The interquartile range (IQR) is $Q3 - Q1$. The second quartile is the median, so $Q1, Q2$, and $Q3$ divide X into four equally-sized pieces. More generally, **quantiles** divide a variable into k roughly equally-sized pieces.

possibility that the choice of bins changes the meaning of a histogram. Density plots are also helpful in probability theory (Section 4.3.4). Similarly, a **violin plot** (Figure 4.2b) is like a box plot, but with information on the density of the data. A **bubble chart** (Figure 4.2f) is like a scatterplot, but allowing dots to have different sizes corresponding to some other variable of interest (e.g., population, if one were plotting GDP vs. time). This idea of layering one additional aesthetic (size, in this case) on top of an existing plot, to capture one more variable of interest, is the core concept behind the **grammar of graphics** that one would learn in a course on data visualization [509].

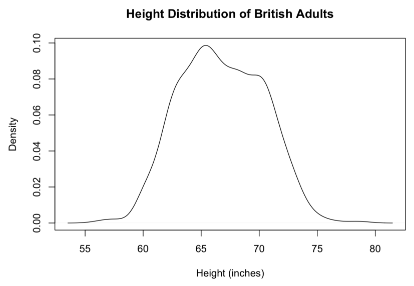
Another example of adding an aesthetic to an existing graphic is a **choropleth map** (Figure 4.2d), which shades regions in a map according to some variable, e.g., shading countries according to their military spending. A **heat map** (Figure 4.2c) works similarly, but ignores existing boundaries and instead shades (x, y)-coordinates (such as latitude and longitude) based on some quantitative variable of interest (e.g., the crime rate). Advanced data visualizations can also be used for categorical data or textual data. For example, a **word cloud** (Figure 4.2e) displays how frequently each word appears in a document, via the size of the word in the word cloud. Advanced visualizations can also be used to mislead, e.g., by reversing the typical shading convention in a choropleth map, so that countries with a higher density are shaded lighter rather than darker. Making choices about aesthetics is another area where a mathematical background can be very helpful in data science. For example, in a bubble chart, most statistical software will use the specified "size" variable to scale the bubbles, effectively making bubble radius directly proportional to the size variable. However, humans tend to perceive the significance of an object proportional to the space it takes up in our visual field, so using our data to impact radius may mean that the *squares* of our data are what the viewer is perceiving. If we do not wish this, we may need to apply a square root transformation to our size variable before giving it to the statistical software.

The article [364] contains an excellent summary of data visualization, and assignments to guide the reader through creating and interpreting visualizations. We turn now to a different kind of summary information that can be extracted from a dataset.

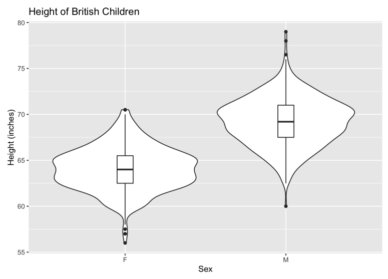
4.2.1 Descriptive statistics

In addition to summarizing data through visualizations, exploratory data analysis also entails extracting numerical and categorical summaries of a dataset. For example, one can and should extract the number n of individuals in the dataset, what kinds of information is present regarding each individual (e.g., summaries of the columns of a spreadsheet), and how much of the data is missing. Identifying missing data is an important part of thinking about issues of bias, and how such issues might limit the scope of analysis (Section 4.5.6).

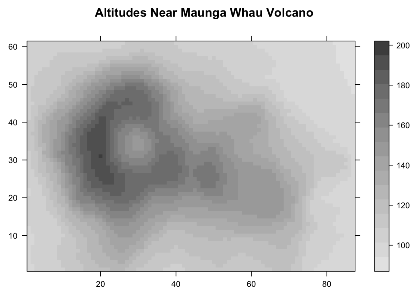
For a quantitative variable X, one often desires a measurement of the central value of X. For symmetric data, we usually use the mean, $\overline{x}$, for the



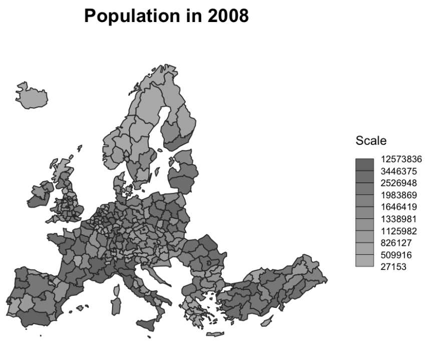
(a) Density Plot



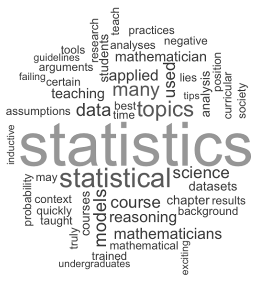
(b) Violin Plot



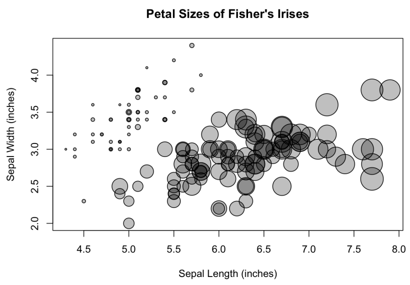
(c) Heat Map



(d) Choropleth Map



(e) Word Cloud



(f) Bubble Chart

FIGURE 4.2: Advanced visualizations.

central value of X. However, if X is skew, it is more appropriate to use the median value (which is less affected by outliers).

A crucial concept in statistics is the difference between a **population** (i.e., the set of individuals we want to infer information about) and a **sample** (i.e., the set of individuals we have data on). For example, a phone survey asking individuals about their voting preferences represents a sample of the population of all potential voters, whereas a census tries to obtain data on every individual in a country. The value of a numerical summary on an entire population is called a **population parameter**, and we often denote such a value using Greek letters. For example, the mean height of every American man would be denoted by μ. It is often prohibitively expensive to survey every individual in a population, so we will almost never know the true value of a population parameter. Instead, we compute a **statistic** from a sample, e.g., the sample mean, $\overline{x}$. If our sample is representative of the general population then the sample statistic will be close to the population parameter, and we can compute a confidence interval for the population parameter (Section 4.4.2).

In addition to a measurement of centrality, we often desire a measurement of the inherent variability in a quantitative variable X. For this, we often use the standard deviation of X,[2] defined in the formulas below (one for a population of N individuals, and one for a sample of n individuals):

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}} \text{ and } s = \sqrt{\frac{\sum(x_i - \overline{x})^2}{n-1}}.$$

The summation above is over all individuals under study, so runs from 1 to N for the population, and from 1 to n for the sample. Clearly, if X has no variability (i.e., if $x_i = \overline{x}$ for all i) then $s = 0$. When X has a larger variance, this means the histogram for X is fatter, more spread out. When X has a smaller variance, it is more tightly concentrated around its mean, so the histogram is thinner. With software, modern statistics courses can de-emphasize rote memorization of formulas for these quantities, and focus instead on conceptual topics such as how to identify when one histogram represents more variability than another. There is no longer any reason to ask students to compute these quantities by hand.

The reason that the denominator for s is $n-1$ rather than n is to make s an unbiased estimator for σ (defined in the next section). A deeper reason has to do with the concept of **degrees of freedom**. The idea here is that, because the formula for s depended on $\overline{x}$, we expended one degree of freedom in estimating σ. That is, given $n-1$ of the data points, plus $\overline{x}$, we could solve for the n^{th} data point, so we really only have $n-1$ bits of information going into our estimation for s. A mathematically rigorous explanation for the phenomenon of "losing a degree of freedom" is that our estimation procedure requires projecting from an n dimensional space of data onto an $n-1$ dimensional space. We will return to this linear algebra approach to statistics in Section 4.7. For now, we delve deeper into the difference between a sample and a population, and the notion of bias.

[2]The IQR can also be used to measure variability.

4.2.2 Sampling and bias

Before attempting to fit statistical models to data, or make inferences about an unknown population based on a sample, it is important to think about where the data came from (i.e., the **data-generating process**), and ways the sample may be systematically different from the population. The first step is correctly identifying the population under study, and thinking about whether the sample is representative of the population. For example, if one were to analyze data from the dating website OkCupid, can this data be considered representative of all individuals in the United States? Of all individuals who are single and actively seeking to date? Or, should any conclusions drawn be restricted to *only* talk about individuals using OkCupid? These are hard questions for which there is not always an obvious correct answer. Effective ways of teaching students to think about these kinds of issues include case studies, discussions in class, and short persuasive writing exercises. When teaching statistics in a project-driven way, as described in [498], students can continue practicing these skills in their lab reports.

Many results in classical statistics assume that data is obtained as a **simple random sample**, i.e., all individuals in the population have an equal chance of being in the sample. However, in the practice of data science, many important datasets are not random. For example, data from social media and Twitter is created when individuals opt to use these platforms, rather than being randomly assigned to use them. There are statistical tests to determine if a dataset behaves like random data for the purposes of statistics (Section 4.6.4), but these tests cannot detect every failure of randomness, and they are not a substitute for carefully thinking about where the data came from. As a first step, one should identify how a study gathered data, whether random sampling was used, and whether variables were controlled for (Section 4.8). Sometimes, if a statistician desires a sample guaranteed to be representative (e.g., to contain both men and women, members from each racial group, etc.), then **stratified random sampling** is used. This means the population is first divided into groups (e.g., white females, Latino males, etc.), and then simple random sampling is used to sample individuals from each group, usually respecting the group's size relative to the population size.

To illustrate the importance of stratified random sampling, consider **Simpson's paradox**. This is the phenomenon where the decision to group data can reverse a trend seen in ungrouped data. This phenomenon is best illustrated with an example, such as Table 4.1 (from [11]), which compares the batting average of two baseball players (Derek Jeter and David Justice).

TABLE 4.1: Batting averages of two baseball stars in 1995 and 1996.

Player	1995	1996	Combined
Derek Jeter	12/48 or .250	183/582 or .314	195/630 or .310
David Justice	104/411 or .253	45/140 or .321	149/551 or .270

The batting averages of Justice were higher than those of Jeter in both 1995 and in 1996, but the combined average of Jeter was higher over those two years. This happens because of the unequal sample sizes. Jeter's .314 in 1996 included many more at bats than Justice's .321, and this is reflected in the combined average. Where Justice had more at bats, in 1995, he did not do as well as Jeter in 1996. Examples such as this one can help clarify for students why they might want to use stratified random sampling if they ever ran an experiment, to guarantee equal sample sizes in all cells. Of course, for the example of baseball data, one cannot assign an equal number of at-bats for each of the two hitters. Indeed, most modern data analysis is conducted on data that did not arise from a randomized, controlled experiment.

When doing data science using "found data" rather than data obtained from random sampling, it is essential to think about issues of bias. For example, evaluations on RateMyProfessor do not represent a random sample of student evaluations, since they are created only when students decide to take the time to navigate to RateMyProfessor and write such an evaluation. Studies have shown that extreme evaluations are overrepresented on this platform. A similar phenomenon occurs with ratings on Yelp, perhaps because only individuals with strong opinions feel sufficiently motivated to write a review. Bias can creep in even when data is gathered randomly. The standard example is gathering data by calling people at home, since this procedure leaves out anyone who doesn't own a home phone.

Another subtle issue is deciding if a dataset represents a sample at all, or whether it is in fact the entire population. For example, suppose we have a dataset consisting of information about every person who died of a drug overdose in Ohio in 2017. If we view this dataset as a sample from the entire United States, we are implicitly assuming Ohio is representative (note that a random sample of overdose deaths in the United States would not need this assumption). If we view the dataset as a sample from the period of time 2014–2019, we are again assuming 2017 was a representative year. However, if we view the dataset as the entire population of individuals who died of an overdose in Ohio in 2017, then many of the tools of statistics (such as confidence intervals and inference; Section 4.5) are not available to us. We can still make data visualizations, extract numerical summaries, and find models to describe the population (Section 4.3), but it does not make sense to assess the statistical significance of such models.

Sometimes, we can get around the conundrum above by considering the dataset as representative of some hypothetical population (an idea we return to in Section 4.6.4). For example, we might have reason to believe that our dataset of overdose deaths is incomplete (due to unreported overdose deaths, coroner errors, etc.) and in that case we could view our dataset as a non-random sample of all overdose deaths in 2017 in Ohio. For another example, consider a dataset of all plays in the 2019 National Football League season. When statistics is used in sports, it is common to face questions like: "Is it better to run or pass against the 2019 Patriots on third down?" If we view the dataset as a population, we can only answer the question "*was* it better," and our answer will involve a summary of all plays made against the 2019 Patriots on third down. If we view the dataset as a sample from a hypothetical population (of how all plays *could* have gone), then we can attempt to use statistical models (perhaps using many more variables than just what happened on third-down plays) to answer this question based on traits of the opposing team. This is related to how video games simulate football games between historical teams, based on statistical models built from the historical data.

Moving forward, we will assume that our dataset is a sample taken from an unknown population. In general, we say that a statistic is **biased** if it is systematically different from the population parameter being estimated. Bias can arise from how the data was gathered (e.g., only surveying individuals who have phones), from certain individuals' systematically declining to participate in a survey (this is called "non-response bias"), or from how survey questions are phrased. Another source of bias comes from measurement error, also known as observational error. **Measurement error** occurs when a device for gathering data does so in a flawed way, e.g., if a device is incorrectly calibrated, or if a method of collecting individuals to study (e.g., turtles, or rocks) has a tendency to oversample larger individuals. Once students can identify bias and measurement error, they are empowered to argue against research that fails to take these concepts into account, they will be more likely to design good study protocols (another area where short writing exercises can help students put theory into practice), and they can begin to think about ways to correct for bias in their data. For example, if one recognizes that a measurement device was incorrectly calibrated, and if one can measure the average error, then one can correct data gathered by the incorrect device.

Since most data in the world is not produced by randomized controlled experiments, it may be important for a first course in statistics to teach students the basics of getting data from different sources into a form suitable for analysis. Alternatively, such a topic could be relegated to a first course in data science, which might come before a formal introduction to statistics, as recommended in [486]. Either way, we delay discussion of data wrangling and data cleaning until Section 4.10.1, so that we can get to the core of statistics more quickly.

4.3 Modeling

The power of statistics derives from its ability to fit models to data, and to use those models to extract information, e.g., predictions. Some mathematicians have a visceral negative reaction to the idea of modeling, but it is worthwhile to remember that a tremendous amount of the mathematics we love was, in fact, inspired by attempting to model real-world phenomena. For example, much of Newton's work in creating calculus was inspired by physical applications. Euler invented graph theory to model locations and the bridges between them. Much of geometry, e.g., the notion of a Calabi-Yau manifold, or the types of properties we prove for Riemannian manifolds, was inspired by theoretical physics and the shape of the universe. Much of dynamical systems, e.g., chaos theory, ergodic theory, and coupled differential equations, was inspired by applications in biology and in particle physics. The point is that, when mathematicians take inspiration from the real world, we come up with models that are intrinsically interesting, aesthetically pleasing, and worthy of study in their own right (not to mention actually being useful to the disciplines who rely on us).

The models that arise in statistics are no less interesting than the models discussed above. Many statistical models are simple (e.g., lines and hyperplanes), but the inherent randomness in data means there are real subtleties when it comes to fitting the model to the data. Often, this fitting process aims to minimize the error, i.e., the differences between the model and the data (which are called **residuals**). However, care must be taken not to overfit a model to data, as we will discuss shortly. Furthermore, different choices for the norm to minimize lead to different models, which are suitable for different kinds of data situations. The field of statistical modeling benefits from the involvement of mathematicians, because we deeply understand the formulas used in such models, we can think through when one model is or is not a good fit, and we can understand subtleties that arise in the fitting process (e.g., trying to invert a singular matrix). Many data scientists use statistical models as a black box, and this can lead to flawed analyses. This is an area where a little bit of mathematical understanding goes a long way.

Fundamentally, a statistical model takes the form `data = model + error`, where `error` is random noise. There are many different types of models, including discrete models (clustering data points into a fixed number of groups, as in Chapter 5), linear models (fitting a linear function that predicts the response variable from the explanatory variable(s)), polynomial models, and computational models (as seen in a machine learning course, covered in Chapter 8). In addition to knowing the common types of models, it is important for practitioners of data science to know how to select a model, how to fit a model, how to validate a model (e.g., to avoid overfitting), how to assess a model, and how to draw conclusions from a model. The idea of modeling should be introduced early in a first statistics course, and spiraled back to as the course unfolds. Indeed, it is possible to frame an entire semester-long course in terms of modeling [256].

4.3.1 Linear regression

Linear regression is the most common model in all of statistics. It is used to model a linear relationship between two quantitative variables. When

TABLE 4.2: A portion of the data in Galton's survey of the heights (in inches) of people in Britain in the 1880s.

Row	x = Father's Height	y = Child's height
1	70	66
2	70.5	68
3	68	61
4	66	61
5	67.5	64
6	65	60.5
7	70	60
8	69	64
9	70	63
10	68.5	60
$\vdots$	$\vdots$	$\vdots$

we fit a linear model, we will often begin with a spreadsheet (or CSV file) of data of the form shown in Table 4.2.[3]

A linear model states that $y = \beta_0 + \beta_1 x + \epsilon$, where ϵ is some random error term whose expected value is zero. For example, on the entire Galton dataset, statistical software can easily compute that the best-fitting linear model is approximately $y = 39.11 + 0.4x + \epsilon$ (the code for which is in Figure 4.3). In addition to explaining the relationship between the **explanatory variable**, x, and the **response variable**, y, the model can be used to predict values of y associated to given values of x. We use the notation $\hat{y}$ for a predicted value of y, obtained by substituting a value of x into the regression equation. For example, the model predicts that for a father of height 70 inches, the child is expected to have height $\hat{y} = 39.11 + 0.4 \cdot 70 = 67.11$ inches. Here we are using the assumption that the expected value of ϵ is zero. The first row in the table above represents a father of height 70 inches and a child of height 66 inches. For row 1, the residual is $\epsilon_1 = 66 - 67.11 = -1.11$. For row 7 above, the residual is $\epsilon_7 = 60 - 67.11 = -7.11$.

We reserve the notation β_1 for the true slope in the population (here, families in Britain in the 1880s), and we use $\hat{\beta}_1$ for the slope in the sample, since it depends on the specific random sample of 898 families Galton studied. The R code in Figure 4.3 demonstrates how easy it is to fit and analyze a linear regression model. Note that in R, the notation $y \sim x$ means y as a function

[3]This data represents a subset of the Galton dataset, a dataset representing a random sample of 898 families from the population of families in Britain in the 1880s. This dataset comes with the mosaic package of R [394], and hence can be ported to any statistical computing language. As discussed in Section 4.10.2, it may be appropriate to give students a dataset like this with some impossible data, e.g., a father whose height is 7 inches instead of 70, so that students can think about omitting such rows of dirty data.

```
require(mosaic)
data(Galton)
xyplot(height ~ father, data = Galton)
mod = lm(height ~ father, data = Galton)
summary(mod)
xyplot(height ~ father, data = Galton, type = c("p","r"))
```

FIGURE 4.3: R code that fits a linear model to the data from Table 4.2, prints the model summary, and creates a scatterplot with a regression line.

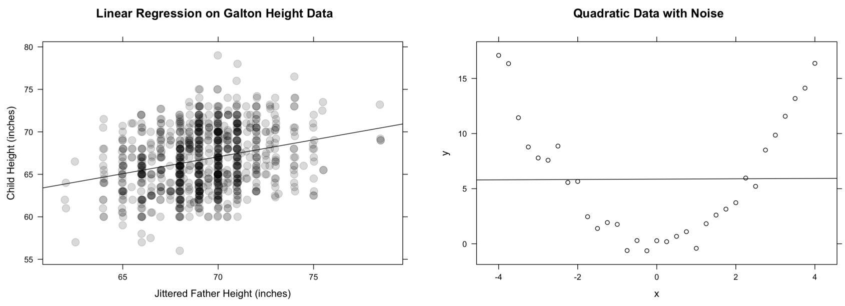


FIGURE 4.4: Sample scatterplots and regression lines.

of x, `lm` is short for "linear model," and `mosaic` is a package developed for teaching statistics and creating attractive visualizations [394].

The code in Figure 4.3 shows how to load the Galton dataset, how to make a scatterplot to check that the relationship between a father's height and a child's height is linear, how to fit the model, how to extract a regression table (to assess the model), and how to plot the scatterplot with the regression line overlaid. The resulting plot displays how the line captures the trend.

Scatterplots, like the examples in Figure 4.4, provide a valuable opportunity to get students thinking about nonlinear relationships, to foreshadow more complicated regression models (such as quadratic regression), transformations (e.g., taking the log of y), and even topological models (e.g., if the scatterplot represents a circle), should the instructor desire.

At this point in the course, it is not essential for students to know the formulas for computing $\hat{\beta}_1$ and $\hat{\beta}_0$ from a table of values (x_i, y_i) as above. It is sufficient for students to know that there *are* such formulas (so that, for a fixed dataset, the regression line is unique), that these formulas are referred to as the **ordinary least squares (OLS)** formulas, and that the purpose of the formulas is to minimize the **residual sum of squares** (RSS), i.e., the following sum, over all data points $(x_1, y_1), \ldots, (x_n, y_n)$.

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2$$

If students have taken calculus, this is a natural moment to remind students of the fact that optimization problems (such as the minimization problem above) can be solved with calculus. An alternative way to derive the equations for the regression coefficients proceeds via projections in linear algebra [100]. Mathematicians may recognize that minimizing the RSS is the same as minimizing the ℓ_2-norm of the vector of residuals, over all choices of $\hat{\beta}_0$ and $\hat{\beta}_1$.

Actually carrying out these derivations is best left for a second or third course in statistics. In such a course, one could also prove the Gauss-Markov Theorem, which states that, under the classic regression assumptions, the formulas for $\hat{\beta}_0$ and $\hat{\beta}_1$ obtained from OLS are the "best" linear unbiased estimators for the model (i.e., minimize variance). The meaning of "unbiased estimator" will be explained in Section 4.3.4. The classic regression assumptions state that the relationship between x and y is linear, the errors ϵ have mean zero, the errors are homoscedastic (i.e., all of the residual errors have the same variance), the data points are independent (which can fail if X represents time, where the past tells us something about the future), and, if there are multiple explanatory variables, that they are linearly independent. In Section 4.6.4, we discuss what happens when these assumptions fail. We will return to the Gauss-Markov Theorem in Section 4.7, after introducing the variance of the $\hat{\beta}$ coefficients. We discuss alternatives to the ℓ_2-norm in Section 4.7.2, which yield different estimates for β_0 and β_1 that are appropriate in certain situations.

This unit on regression is also an excellent time to introduce correlation. Just as with the mean or standard deviation, there is a population version of correlation (denoted ρ) and a sample version (denoted r or R). Below, we discuss the sample correlation, and an analogous discussion holds for population correlation (based on population means and covariance). While students should not be made to compute correlation coefficients by hand, the formula for correlation is worthy of inspection.

$$r = \frac{\sum((x_i - \overline{x})(y_i - \overline{y}))}{\sqrt{\sum(x_i - \overline{x})^2 \sum(y_i - \overline{y})^2}}$$

The numerator is the **covariance**, a measurement about how two quantitative variables, x and y, vary together. In the denominator, students see formulas for the variance of x and y. Taken together, we learn that **correlation is a scaled version of covariance**, scaled to take values between -1 and 1. An example of data measured in different units (e.g., in kilometers vs. meters) can demonstrate to students why it is important to scale the covariance. One can often get an idea of the correlation by looking at scatterplots. If the data falls perfectly on a line of nonzero slope, then the correlation is 1 or -1 (corresponding to the sign of the line's slope). If the scatterplot is a cloud, where the best-fitting line has slope zero, then the correlation is zero. It is also important to teach students that correlation measures the strength of a *linear* relationship only, and it is possible to have two very closely related variables that still have a correlation of zero (as in Figure 4.4).

4.3.2 Polynomial regression

As the previous section demonstrates, when data fails to satisfy a linear relationship, a linear model is not appropriate.[4] We must therefore choose a different statistical model, based on the scatterplot. For example, if the scatterplot suggests a cubic relationship, we would choose a model of the form $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$. By minimizing the residual sum of squares, as in the previous section, we could find estimates $\hat{\beta}_i$ for the population parameters β_i. There is no reason to stop at polynomial models: if one wished to fit a sinusoidal model to the data, the same considerations apply. Again, this is an area where a little bit of mathematical knowledge goes a long way.

The ability to fit any model desired raises the danger of overfitting, and the need to discuss the bias-variance trade-off. **Overfitting** occurs when a model fits a dataset "too well," meaning that the model is not suitable for generalizing from the dataset at hand to the unknown population of interest. For example, the Galton dataset above has 898 individuals. If desired, one could perfectly fit a 897-dimensional polynomial $y = \beta_0 + \beta_1 x + \cdots + \beta_{897} x^{897}$ to the dataset, resulting in an 898-dimensional residual vector of zero, $\vec{\epsilon} = \vec{0}$.[5] The problem is that this model would be terrible for predicting heights of individuals not in the dataset. For example, just after the tallest father, the graph would veer steeply towards infinity, and would therefore predict giant children taller than skyscrapers.

Our best tool to prevent overfitting is **cross-validation**. This means we hold out some fraction of the data to be used for testing our model rather than fitting it. For example, we might hold out 10% of the data above, or 90 data points. We then fit a linear model based on the remaining 808 data points, then check the residuals for the remaining 90 data points. If these 90 residuals look basically the same as the 808 residuals from the model, then we have reason to believe the model is a good fit, and that residuals from unknown population data points would look similar. Readers interested in more detail on cross-validation can see Sections 6.3.3.2 and 8.2.2. The crucial mindset for a statistician is the idea that more data is going to come later, and we need our model to be a good fit for that data, rather than just the data we began with.

When deciding between a complicated model and a simple model, like deciding which k to use if fitting a k-dimensional polynomial, it is valuable to think in terms of **sensitivity analysis**. The **bias-variance tradeoff** is a framework for such considerations. In this context, the **bias error** represents

[4] At least, not unless transformations are applied; see Section 4.6.1.

[5] Technically, to do this one should assume there are no repeated x observations, but this can be fixed by adding a very small amount of random noise to the x values in the data, a process known as **jittering**.

error due to the choice of model, e.g., the error that would arise if you fit a cubic polynomial but should have fit a quartic. The **variance error** represents the error due to the sensitivity of the model to small changes in the data, e.g., how much the cubic polynomial would change if one of the y values had a slightly larger random error. Models with high variance may inadvertently be fitting β coefficients based more on the random noise than on the relevant "signal" (i.e., the true relationship between x and y in the population).[6] One of the joys of statistical modeling is that there is no one single right answer when it comes to the choice of model, and this leads to spirited debate. However, for the purposes of a first course in statistics, we generally err on the side of choosing simpler, rather than more complicated, models, as long as doing so does not ignore crucial aspects of the phenomenon we are studying (such as the effect of gender on salary). This choice is sometimes called the **principle of parsimony** (also known as the Law of Parsimony, or Occam's razor). We illustrate these ideas with another example in the next section.

4.3.3 Group-wise models and clustering

A **discrete model**, also known as a **group-wise model**, is a particularly simple model based on data that assigns each individual to a group. This means the explanatory variable is a discrete variable. For example, one can understand the wage gap via a scatterplot with salary on the y-axis, and where x values are 0 for men and 1 for women, as in Figure 4.5. In this example, the model assigns to each male the average salary of the men $\overline{x}_{men}$, and assigns to each female the average salary of the women $\overline{x}_{women}$. This model is a special case of a linear regression model of the form $y = \beta_0 + \beta_1 x + \epsilon$, where x takes values of 0 or 1, and ϵ is the error term (how badly the model misses reality).

This example foreshadows the main idea behind the two-sample t-test, that of studying the difference $\overline{x}_{men} - \overline{x}_{women}$. When doing inference for regression (i.e., testing the null hypothesis $H_0 : \beta_1 = 0$, as we will cover in Section 4.5), a class can revisit this example to see that the two-sample t-test is a special case of regression inference.

The study of discrete models also allows for a discussion of how to choose a model. Students can see that choosing one group (i.e., ignoring gender, so assigning each individual to the overall average salary $\overline{x}$) explains less variability than choosing two groups. On the other extreme, choosing n groups—one for each individual—explains all the variability, because all residuals are zero, just like fitting an n-degree polynomial above. In this context, the bias-variance trade-off is about the bias error (i.e., systematic modeling error) in choosing too few groups vs. the variance error caused by choosing too many groups (i.e., how radically clusters will change when data points change by a little bit). The study of groupwise models leads naturally into clustering,

[6]We have chosen the terms "bias error" and "variance error" to avoid confusion with the statistical terms "bias" and "variance." For more on the bias-variance trade-off, we recommend [247]. This important issue will also appear in many later chapters of this text.

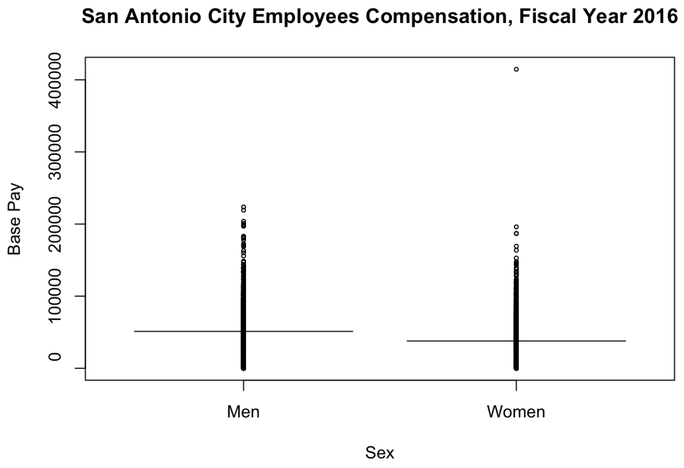


FIGURE 4.5: A scatterplot of compensation by sex. Data provided on the website data.world by the City of San Antonio for 11,923 employees. The mean for each sex is shown with a line segment.

and so we will not say more about about it here, but direct the reader to Chapter 5. Instead, we pivot to probability models, i.e., trying to determine which underlying probabilistic process might have generated the data.

4.3.4 Probability models

A **probability model** is a choice of probability distribution that may have generated the data. For example, if the data represents counts (e.g., the number of people buying a mattress each day of the week), then a Poisson model may be appropriate. When we speak of "fitting" a probability model, we mean estimating the parameters of the model based on the dataset. As a Poisson model is determined entirely by its mean, μ, we fit the model by computing the sample mean $\overline{x}$. The study of probability models is essential to statistics, because many results in statistics rely on the **Central Limit Theorem**, which states that the average (or sum) of many repeated experiments[7] is asymptotically normally distributed as the number of trials goes to infinity. This theorem is the reason that the normal distribution is ubiquitous in statistics and is used when statisticians compute confidence intervals and p-values (Section 4.5).

It is challenging to speak precisely about probability theory. A careful, technical definition of the probability of an event requires a detour into measure theory. Therefore, in statistics, we must learn to communicate less

[7]By which we mean independent and identical random trials.

formally about probability theory. For the purposes of statistics, randomness is taken to be inherent in the world, and a **random variable** is a variable whose values change according to some random process, which students can understand intuitively via example.[8] For example, the temperature outside at any given moment is a random variable. Commonly, random variables are generated by random processes or experiments; e.g., the top face on a standard six-sided die is a random variable generated by the process of rolling the die. If we repeat an experiment, we soon encounter sums of random variables. Another source of random variables is statistics computed from random samples; e.g., Galton's $\hat{\beta}_1$ (Section 4.3.1) was a random variable, because it depended on the specific 898 individuals in his study, obtained by random sampling.

Of all topics in statistics, probability theory is the most mathematical. It is therefore easy for a mathematician teaching statistics to enjoy spending too much time in a unit on probability theory. Browsing textbooks that teach statistics for engineers, it is common to find lengthy sections on probability theory, including formal definitions of sample spaces, Venn diagrams (for event unions, intersections, De Morgan's laws, conditional probability theory, and independence), expected value, and variance. Such books often contain results such as Chebyshev's theorem, the method of moments, and the law of the unconscious statistician. It is common to find definitions of (joint) probability mass/density functions, cumulative density functions, and a plethora of distributions, including the Bernoulli, binomial, geometric, hypergeometric, Poisson, uniform, and exponential. (A few of these are illustrated in Tables 6.1 and 6.2 on pages 264 and 265.)

I argue that none of this is required in a first course in statistics. The unit of probability is often the hardest for students, because many will lack probabilistic intuition. The GAISE guidelines are clear that probability theory should be de-emphasized in a first course in statistics, whether or not a calculus prerequisite is in play [13]. As hard as it is, we must resist delving into probability theory more deeply than is absolutely necessary. The essential topics from probability theory include the following.

- The idea that a **probability model** assigns a probability to an event [256]. Here an **event** can be formally defined as a subset of the set of all possible outcomes (e.g., the event of rolling an even number when rolling a standard six-sided die), but for the purposes of teaching it is better to avoid a digression into set theory or Venn diagrams.

[8]Formally, a random variable is a measurable function from a probability space to a measurable space. The probability space has a set (actually, a σ-algebra) of events, and a function (a probability measure) assigning each event a probability. The codomain of a random variable is a Borel space, and is almost always $\mathbb{R}^k$ in our settings. However, general measurable spaces are allowed, e.g., the space of finite graphs, if the random variable in question assigns random graphs $G(n, p)$ in the sense of Erdös and Rényi.

- An informal, frequentist, definition of the probability of an event. This builds the law of large numbers into the definition. For example, the probability of rolling a 2 on a standard 6-sided die is 1/6 because if we conducted a huge number of trials, about 1 in 6 would come up with a 2 on the top face. Formally, this is a statement that

$$\lim_{n\to\infty} \frac{\text{Number of times a 2 is rolled}}{\text{Number of Trials}} = \frac{1}{6}.$$

- The normal distribution, introduced as a picture (Figure 4.6) rather than via the formula for its probability density function. Students should also be given examples of non-normally distributed phenomena, such as the uniform distribution (e.g., illustrated with the example of a die roll).

- The notion that probability is an area under a curve (Figure 4.6), and examples of cutoff x values that leave 5% of the area in the tails.[9] Students need to know that the interval $[\mu - 1.96\sigma, \mu + 1.96\sigma]$ contains 95% of the area under a $N(\mu, \sigma)$ distribution. Similarly, 99% of the area is within 2.33 standard deviations from the mean. Students should be able to find cutoffs like 1.96 and 2.33 using software or using tables of values of the standard normal distribution.

- A visual understanding of variance as it relates to how "fat" the distribution is, as we saw for histograms in Section 4.2, and observation that a larger variability leads to larger tail probabilities.

- Simulation or applets [300, 411] so that students can see how a normally distributed random variable is more likely to take values near its mean, rather than in the tail, and how taking the average of many trials tends to produce a number even closer to the mean.

- The Central Limit Theorem and many examples where it does and does not apply. The use of simulation to visualize sums and averages of repeated trials converging to the normal distribution, and to see that a sample size of $n \geq 30$ is usually enough to rely on the Central Limit Theorem, unless the data is strongly skewed.

Students should get into the habit of checking if a given distribution is normal, e.g., using histograms or density plots to see if it "looks" normal, or **quantile-quantile (q-q) plots**. A q-q plot displays the quantiles (Section 4.2) of the given distribution along the x-axis, and the quantiles of a normal distribution along the y-axis. If the plot fails to be linear, then this is evidence that the distribution is not normal. There are also formal hypothesis tests for normality that we will discuss below. In many cases, we will not need our

[9]Given a probability density function Z and a cutoff z, the **right tail** is the set of values of Z larger than z. The **left tail** is defined similarly. When we say **tails**, we mean the union of the left and right tails.

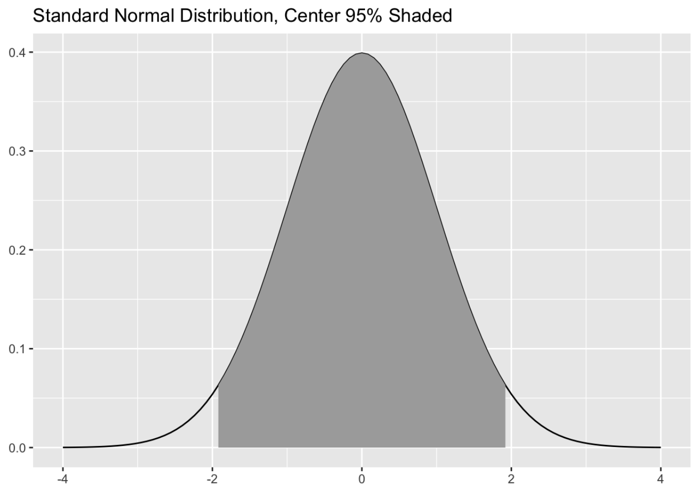


FIGURE 4.6: Standard normal distribution ($\mu = 0, \sigma = 1$) with the shaded region corresponding to $[-1.96, 1.96]$, covering approximately 95% of the area under the curve.

random variables to be normal, because the Central Limit Theorem applies even to sums and averages of samples from any distribution. If a dataset represents a sample of size $n < 30$, from a non-normal distribution, then n is not large enough for the Central Limit Theorem to tell us the average is approximately normal. Instead, the average is distributed according to a ***t*-distribution**, with $n-1$ degrees of freedom (Section 4.2). The t-distribution represents an infinite family of distributions that converges to the normal distribution as $n \to \infty$, and is close enough if $n \geq 30$ so that the normal distribution can be used as an approximation.

De-emphasizing overly theoretical discussions of the properties of probability distributions allows a first course in statistics to devote more time to simulation. Simulation is an important topic in applied statistics, an essential topic in data science, and a valuable way to give students probabilistic intuition. As students gain facility with simulation, it is possible to teach them how to use Monte Carlo simulations to approximate solutions to elementary probability theorem problems (e.g., the Kobe Bryant Open Intro lab [61]). That said, the core learning outcome for Monte Carlo simulations will still be achieved via a discussion of randomization-based inference and bootstrap resampling (Section 4.4.3).

Similarly, Bayes' Formula can be delayed until students need it, if you have time later in the course (or in a future course) to introduce the ideas of Bayesian statistics (Section 4.9). An upper-level course in mathematical statistics or probability theory will usually include analytic derivations of model parameters (such as mean and standard deviation), as well as proofs of relationships between distributions, such as the relationship that a squared t-distribution is an F-distribution. An upper-level course on survival analysis (the branch of statistics that models how long something will last) usually includes more distributions, such as the Weibull distribution, that are used to model wait times.

We conclude by mentioning that linear regression models (Section 4.3.1) are special cases of probability models. The Central Limit Theorem guarantees that the collection of $\hat{\beta}_1$, over all possible random samples of size n, is approximately normally distributed. Thus, finding estimates for $\hat{\beta}_1$ and its standard deviation $SE_{\hat{\beta}_1}$ can be viewed as fitting a normal distribution. This type of fitting procedure is accomplished via maximum likelihood estimation (which we discuss next), and the resulting formula for $\hat{\beta}_1$ agrees with the ordinary least-squares estimation from Section 4.3.1.

4.3.5 Maximum likelihood estimation

The primary theoretical tool used for fitting probability models to data is **maximum likelihood estimation (MLE)**. This procedure finds the optimal estimates for the parameters of a probability distribution to maximize the likelihood of the data that was observed. Analytically solving the optimization equations requires calculus, and this is another area where instructors are warned not to get bogged down. One example usually suffices to give students the idea of MLE, especially if the example involves finding the maximum intuitively or graphically, rather than analytically. In the example that follows, we use a common statistical notation that sometimes trips up newcomers. The notation $Pr(A; p)$ refers to the probability of an event A, as a function of an unknown parameter p. In this notation, the data (hence the event A) is fixed, and the parameter is allowed to vary, since we don't know the true population parameter in practice. When a statistician writes $f(x; \theta_1, \ldots, \theta_k)$, this represents a multivariable function where x denotes the data (or a quantity computed from it) and θ_i denotes an unknown population parameter. Sometimes, all but one of the parameters θ_i are fixed, and the goal is to solve for the remaining θ_i, by turning the multivarable function into a single-variable function. The reader is advised to read notation in statistical texts carefully, and clarify which quantities are fixed and which are variables. We have endeavored to avoid confusing notation wherever possible.

We now give an example of maximum likelihood estimation. Suppose our data consists of a sequence of heads and tails produced by flipping a coin over and over again, and our goal is to find the optimal parameter, p, such that a biased coin (that comes up heads with probability p) would be most likely to produce the data. For example, if there were 62 heads out of 100 flips, then a biased coin with $p = 0.62$ would have been more likely to produce the data than an unbiased coin (with $p = 0.5$). To prove this, one writes down the **likelihood function**, which tells us the probability of the data observed as a function of possible values of p. If A is the event of seeing 62 out of 100 heads then the likelihood function is

$$L(p) = Pr(A; p) = \binom{100}{62} p^{62}(1-p)^{38}.$$

We then find the p that maximizes this function. As it is difficult to differentiate a product, and the natural log is a monotonic function, we apply it before differentiating, and call the resulting function the **log-likelihood** $l(p) = \ln(L(p)) = c + 62\ln(p) + 38\ln(1-p)$, for some constant c. The derivative $l'(p) = 62/p - 38/(1-p)$ is zero when $62/p = 38/(1-p)$, i.e., when $p = 62/100$. As this is an estimate of the true p, obtained from a random sample, we use the notation $\hat{p} = 62/100$. Different datasets may give rise to different estimates $\hat{p}$, so we can view $\hat{p}$ as a random variable, generated by the random process of taking a random sample of data. In general, we call an estimate $\hat{\theta}$ of a population parameter θ **unbiased** if the expected value of $\hat{\theta}$ equals θ. This implies the average of the values $t\hat{h}eta$, over many random samples, approaches θ.[10] It is a theorem that maximum likelihood estimation produces unbiased estimators.

Other examples of MLE include finding the optimal parameters for Poisson or exponential distributions based on arrival data, or re-deriving the OLS formula[11] for $\hat{\beta}_1$ to maximize the likelihood of paired data (x, y), if the two variables are related linearly by $y = \beta_0 + \beta_1 x + \epsilon$. The mathematics behind maximum likelihood estimation is beautiful, but it is unwise to carry the method out in front of students. Many students would struggle to connect such a lecture to the core topics of the course, and would struggle with the shift from likelihood to log-likelihood. Rather than carrying out such a procedure in class, the method can be introduced informally, at a high level, with elementary examples, and foreshadowing how the method will come up again in future statistics courses. That way, the method will not be quite so foreign when students see it properly, in a mathematical statistics course. The idea of trying many different parameter values and picking the "best" one (here, the one that maximizes the likelihood) is also crucial for operations research (Chapter 6) and machine learning (Chapter 8). The formula for $\hat{p}$ from the previous example could also be obtained from a simulation, computing the likelihood function over a range of possible values of p (roughly, a concave down quadratic), and then selecting the value of p that maximizes the likelihood.

An alternative to maximum likelihood estimation is the **method of moments**, where one writes formulas for the moments of a probability distribution[12] in terms of the unknown parameters, and then solves these equa-

[10]There is also a notion of **asymptotically unbiased estimators**, where this statement is only true in the limit as the sample size approaches ∞.

[11]This justifies our use of the term "unbiased estimator" when discussing the Gauss-Markov Theorem above.

[12]The k^{th} **moment** of a probability distribution X is defined to be the expected value $E[X^k]$. The first moment is the average value, the second moment is related to the variance, etc. Two probability distributions are equivalent if and only if all their moments are equal.

tions to find values for the parameters in terms of the sample moments computed from the data. Discussion of this method is best left for a mathematical statistics course. In a first course in statistics, a high-level discussion about techniques for finding the best-fitting model is an excellent place to mention cross-validation (Section 4.3.2).

Now that we have a collection of techniques for fitting models to data, we turn to the important task of making inferences about an unknown population from a random sample.

4.4 Confidence intervals

The study of confidence intervals is core to a first course in statistics, as it allows one to make claims about the unknown population, based on sample data. A confidence interval for a population parameter is an interval, constructed from a sample, that will contain the parameter for a predetermined proportion of all samples. We find it best to illustrate with an example before delving into a formal definition. In the sections below, we discuss two procedures for producing confidence intervals, one related to the *sampling distribution*, and one based on *bootstrap resampling* (both defined formally in the sections below). The example below is based on the sampling distribution, introduced in the next section. We provide R code to make confidence intervals in Figure 4.9.

Suppose we are studying the heights in inches of all men in the United States, and we have measured the heights of a random sample of n men. A 95% confidence interval for the population mean height μ will be centered on the sample mean $\overline{x}$ (because that is our best estimate for μ from what we know), and its width will depend on n and on the variability present in the sample. For the purposes of the discussion below, let Z be a random variable distributed according to a $N(0,1)$ distribution (i.e., normal with mean 0 and standard deviation 1). We write $Z \sim N(0,1)$ as a short-hand notation. If n is sufficiently large, the confidence interval is given by the formula $[\overline{x}-1.96\cdot\frac{s}{\sqrt{n}}, \overline{x}+1.96\cdot\frac{s}{\sqrt{n}}]$, where s is the sample standard deviation, and 1.96 is (approximately) the value r of the normal distribution Z such that $Pr(-r \leq Z \leq r) = 0.95$. For a given sample, the confidence interval may or may not contain the unknown μ, depending on how lucky we were in gathering the sample. For example, a random sample containing a disproportionately large number of tall men would result in $\overline{x} > \mu$, and the confidence interval might fail to contain μ. The procedure for constructing confidence intervals (i.e., the choice to use 1.96 and $\frac{s}{\sqrt{n}}$) guarantees us that, if we took a large number, k, of random samples of n men, then we would expect 95% of the resulting confidence intervals to

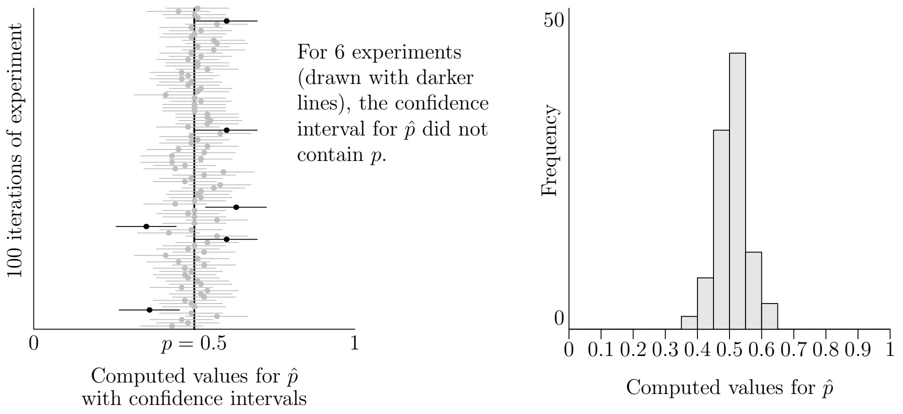


FIGURE 4.7: Results of 100 repetitions of a simulated experiment, each experiment involving flipping a fair coin 100 times and recording the number of heads. In each case, an estimate of the coin's probability p of heads is computed as $\hat{p}$, and a 95% confidence interval around $\hat{p}$ is shown. In 94% of cases, it contained p.

contain μ, and would expect the other 5% of those intervals to fail to contain μ. This is illustrated in Figure 4.7.

If we wanted to be 99% confident, i.e., to expect 99% of samples to yield intervals that contain μ, then the intervals would naturally need to be wider. Specifically, we would use 2.33 instead of 1.96, because $Pr(-2.33 \leq Z \leq 2.33) \approx 0.99$. Note that, as $n \to \infty$, confidence intervals get narrower, meaning we become more and more confident that $\overline{x}$ is close to μ. Similarly, if a dataset has less inherent variability (measured by s), then our confidence intervals get narrower, i.e., we become more confident about the range of values where μ can likely be found.

Formally, given a random sample X, a **confidence interval** for a population parameter θ, with confidence level γ (which was 0.95 and 0.99 in our examples above), is an interval $[a, b]$ with the property that $Pr(a \leq \theta \leq b) = \gamma$.

4.4.1 The sampling distribution

Recall that a **sample statistic** is any quantity computed from a sample. For simplicity, we will assume we are studying the mean, and we will let μ denote the population mean, and $\overline{x}$ the sample mean. Analogous considerations could be used to study other statistics, such as the standard deviation or median. For a fixed population, fixed sample size n, and fixed statistic to study, the **sampling distribution** is the distribution of the sample statistics over all samples of size n from the population. We can visualize the sampling distribution as a dotplot, which is like a histogram but with vertically stacked dots denoting the number of items in each bin (Figure 4.8). For the sampling

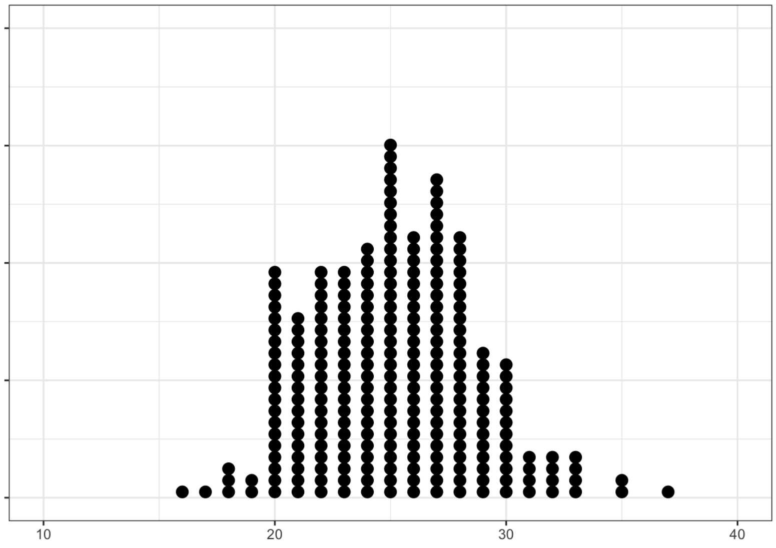


FIGURE 4.8: An approximately normally distributed sampling distribution.

distribution, each dot represents the information you would obtain from collecting a simple random sample of size n, and then computing the sample mean (or other sample statistic) of that sample. Hence, in practice, you will not have access to the sampling distribution, because it is prohibitively expensive to conduct so many samples. However, the Central Limit Theorem tells us that the sampling distribution of values of $\overline{x}$ (over many samples drawn *at random* from the population), will be approximately normally distributed, with mean of μ, and standard deviation $\sigma/\sqrt{n}$, where σ is the population standard deviation.

Of course, in practice, one would effectively never know the constant σ (especially if one doesn't even know μ). Consequently, in practice, one estimates σ by the sample standard deviation s, and uses $s/\sqrt{n}$ to approximate the standard deviation of the sampling distribution. We use the term **standard error (SE)** for the standard deviation of the sampling distribution. The sampling distribution is a challenging topic for students, so the use of applets is recommended [300, 411] to show students the procedure of taking many different samples from the unknown population. It is also important to stress how the formula for the standard error implies that taking larger samples (increasing n) implies a reduction in variability (decreasing the standard error).

```
marginOfError <- qnorm(.975)*sd(Galton$height)
                 / sqrt(nrow(Galton))
mean(Galton$height) - marginOfError
mean(Galton$height) + marginOfError
```

FIGURE 4.9: R code to compute a 95% confidence interval for the mean height.

4.4.2 Confidence intervals from the sampling distribution

Generalizing our example from the start of the section, we note that the sampling distribution is approximately a $N(\mu, s/\sqrt{n})$ distribution (for sample sizes $n \geq 30$, at least). Thus, 95% of sample means are contained in the interval

$$\left[\mu - 1.96 \cdot \frac{s}{\sqrt{n}},\ \mu + 1.96 \cdot \frac{s}{\sqrt{n}}\right].$$

It follows that we can be 95% confident that μ is contained in the interval

$$\left[\overline{x} - 1.96 \cdot \frac{s}{\sqrt{n}},\ \overline{x} + 1.96 \cdot \frac{s}{\sqrt{n}}\right],$$

and hence this interval is a 95% confidence interval for the parameter μ. Half the width of the interval, i.e., $1.96 \cdot \frac{s}{\sqrt{n}}$, is called the **95% margin of error**.

This strategy for making confidence intervals described above depends on the use of the normal distribution to approximate the sampling distribution. In practice, it is better to use a t-distribution with $n-1$ degrees of freedom, since we use s to estimate σ. When $n > 30$, there is little difference between a t-distribution and a normal distribution. If $n < 30$, then it is essential to use the t-distribution, and we replace 1.96 with the value on the t-distribution that leaves 5% in the tails. For example, if $n = 13$, we would use a t-distribution with 12 degrees of freedom. Statistical software, or a t-table, tells us that a 95% confidence interval for the mean μ is given by

$$\left[\overline{x} - 2.179 \cdot \frac{s}{\sqrt{n}},\ \overline{x} + 2.179 \cdot \frac{s}{\sqrt{n}}\right].$$

The R code in Figure 4.9 illustrates how easy it is for R to create a confidence interval. We again use the built-in Galton dataset from Figure 4.3, and we make a 95% confidence interval for the average height in this dataset. Since the Galton dataset has $n = 898$ rows, we use the normal approximation for the sampling distribution. In the code shown, `qnorm(.975)` gives us the value z of the normal distribution $Z \sim N(0,1)$ such that $Pr(Z \leq z) = 0.975$. We have seen above that $z \approx 1.96$. The function `sd` returns the standard deviation, and `mean` returns the mean. When we write `Galton$height`, R gives us the `height` column of the `Galton` dataframe.

When run, this code returns a confidence interval of (66.52, 67). The sample mean was 66.76. Assuming Galton obtained his data as a random sample from the population, we are 95% confident that the population mean height μ is in the interval (66.52, 67). The narrowness of the interval is due to the large sample size of $n = 898$.

This strategy for making confidence intervals works for many other population parameters, including:

1. the difference between two means $\mu_1 - \mu_2$, e.g., the difference in salary between men and women,
2. the proportion p of individuals with some property, e.g., the proportion of people with depression,
3. the difference between two proportions $p_1 - p_2$, e.g., the difference between the proportion of women with depression and the proportion of men with depression,
4. the true slope β_1 for linearly related data given by $y = \beta_0 + \beta_1 x + \epsilon$,
5. the average response value $\overline{y}$ associated to a given value x of the explanatory variable, e.g., the average salary for a $x = 47$ year old,
6. the median,
7. the standard deviation, σ,
8. the ratio of two standard deviations σ_1/σ_2,

and many more.

For (1)–(5), the sample statistic (e.g., $\overline{x}_1 - \overline{x}_2$, or the sample slope $\hat{\beta}_1$) will be approximately normally distributed, and there is a formula that can estimate the standard error of the sampling distribution from sample data. Once you have a standard error, confidence intervals are computed just as above. For the last three examples, other distributions are needed to compute confidence intervals. For example, σ (and the standard deviation of sample statistics s) is distributed according to a χ^2 distribution. There is a formula for the standard error, and instead of 1.96, a cutoff taken from the χ^2 distribution is used. Similarly, σ_1/σ_2 (and the sampling distribution of s_1/s_2) is distributed according to an F distribution. Descriptions of these tests, and formulas for the standard error, can be found in most standard statistical textbooks, e.g., [11, 409].

Case (5) is worthy of further discussion. In order to make a confidence interval around an average y value in a regression, a data scientist must know that the residuals (i.e., the differences $y_i - \hat{y}_i$) are independent and identically distributed. If the sample size is less than 30, one must also know that the residuals are normally distributed. Students can check this assumption using histograms or a q-q plot, as discussed in Section 4.3.4. A data scientist must

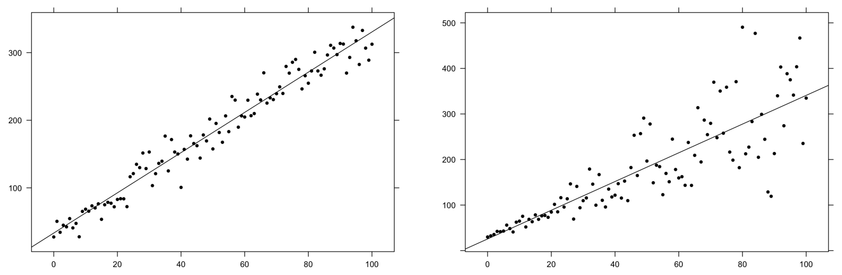


FIGURE 4.10: A regression line through data whose residuals exhibit homoscedasticity (left) and one where the residuals exhibit heteroscedasticity (right).

also know that the residuals are **homoscedastic** (also spelled homoskedastic). This means that the degree to which the residuals are spread around the regression line (i.e., the variance of the residuals) does not depend on the value of x, as shown on the left in Figure 4.10. Students can check this assumption using a scatterplot. In Section 4.6.4, we will see how to cope with a failure of this assumption, as shown on the right in Figure 4.10.

If a data scientist wants to predict future y values associated to a given x value, then it is not enough to predict the average such y value. A 95% **prediction interval** associated with x is expected to contain 95% of future y values associated with x. This will of course be wider than a 95% confidence interval for $\overline{y}$, since averaging reduces variability, as shown in Figure 4.11. For example, using our R code from Figure 4.3, for a fixed value of the father's height (e.g., $x = 65$), a confidence interval would tell us about the mean child height among all children of fathers who are 65 inches tall, whereas with a prediction interval we would be 95% confident that it would contain the height of *any* child of a father who is 65 inches tall. Formulas for prediction intervals can be found in standard statistical texts [11], and prediction intervals are easy to produce using software. Since averages are not being used, it is essential that residuals be normally distributed, or else prediction intervals cannot be produced using values like 1.96 (from the normal distribution). Figure 4.11 is based on a regression for predicting stopping distance as a function of the speed a car is going [150]. The dotted lines show prediction intervals for every x, and the dark gray region shows a confidence interval. Note that we are more confident about the mean response of y for x values near $\overline{x}$, because the dark gray region is narrower for those x values.

Nowadays, it is not essential to possess an encyclopedic knowledge of formulas for standard errors. Software can easily compute confidence intervals of any given level of significance (e.g., 95% or 99%), and so it is much more important to know how to interpret and use confidence intervals than it is to know how to compute their margins of error by hand.

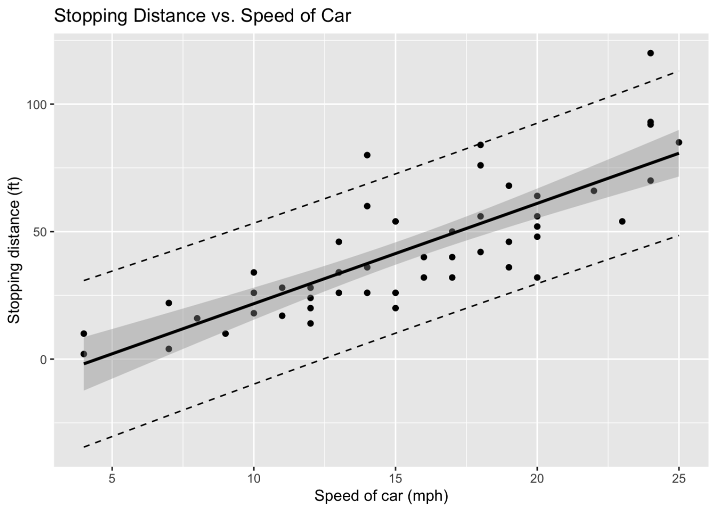


FIGURE 4.11: Confidence and prediction intervals for regression.

4.4.3 Bootstrap resampling

Suppose X is a quantitative variable and we have a sample of n values $x_1, \dots, x_n$. When we represent the sample graphically as a density plot (Section 4.2), we call it the **empirical distribution**. Dotplots and histograms can serve as approximations to the empirical distribution. We will use a dotplot below (as we do when teaching this material) because it clarifies the connection to the sampling distribution, and because it makes p-values intuitive and easy to compute (by simply counting the number of dots in the tail). We assume that our sample is representative of the population, so that the empirical distribution is an approximation to the actual distribution of X over the entire population. For example, if n is 100, and we have two individuals taller than 6 feet, then we can assume the probability an individual in the population is taller than 6 feet is about 2%. This means that each individual in our sample represents potentially many individuals in the population. If we take a sample *with replacement* from our data $x_1, \dots, x_n$, we can treat this new sample like a sample taken from the overall population (we call this new sample a **bootstrap resample**). If we sampled the same tall person twice, this would represent getting two different people of this height from the population.

The upside of this point of view is that we can build the **bootstrap distribution** by taking many bootstrap resamples of our dataset $x_1, \dots, x_n$, and writing down the sample mean of each resample. To avoid confusion, let z_i denote the sample mean of the i^{th} resample. We use a dotplot to visualize the bootstrap distribution, as in Figure 4.12, and each dot represents some z_i. On average, the center of the dotplot will be $\overline{x}$, the mean of our dataset. The standard deviation of the bootstrap distribution is a good approximation to the

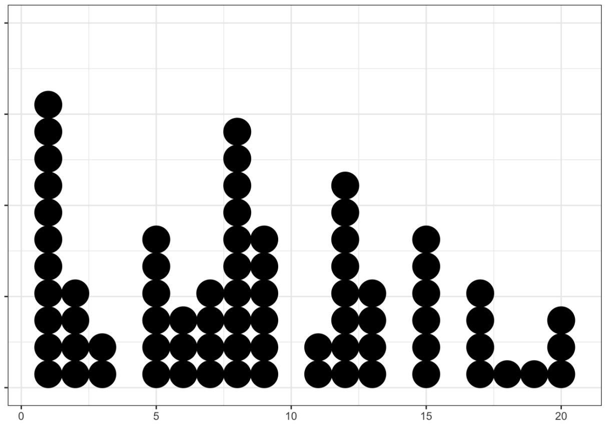


FIGURE 4.12: Dotplot of simulated, one-dimensional data. Each stack of dots represents repeated instances of the same x value.

standard error, if a sufficiently large number of resamples (e.g., about 10,000) is used. The mathematical theory justifying bootstrap resampling is detailed in [217]. The original idea of the bootstrap goes back to Ronald Fisher, one of the giants of statistics, but at that time it was impractical, due to a lack of computing power [217, Section 4.4].

Once you have the bootstrap distribution, you can make confidence intervals in two ways. First, you could write down the standard error SE (using the usual formula

$$s = \sqrt{\frac{\sum(z_i - \overline{z})^2}{n-1}},$$

where each z_i is a mean of a bootstrap resample) and then use the method from the previous section to make a confidence interval $[\overline{z} - 1.96SE, \overline{z} + 1.96SE]$, using the fact that the bootstrap distribution is approximately normally distributed (and that $\overline{z} \approx \overline{x}$). The second option is to simply find the cutoff points L, R in the bootstrap distribution, such that 2.5% of the z_i are less than L, and 2.5% of the z_i are greater than R. One is then 95% confident that the true mean, μ, is in the interval $[L, R]$.

The R code in Figure 4.13 demonstrates how to carry out this procedure to produce a 95% confidence interval for the mean height in the Galton dataset with $k = 1000$ bootstrap resamples. We demonstrate both methods from the previous paragraph. Both agree with the confidence interval produced in Figure 4.9. The code below creates an $n \times k$ matrix (recall that $n = 898$), fills it by sampling from `Galton$height` *with replacement*, and then applies the `mean` function to each column. This results in $k = 1000$ means, which form the bootstrap distribution `boot.statistics`. The two lines that start with `#` are comments in the code.

```
height.mean = mean(Galton$height)
k = 1000
n = nrow(Galton)
boot.samples = matrix(sample(Galton$height,
    size = k * n, replace = TRUE), k, n)
boot.statistics = apply(boot.samples, 1, mean)
densityplot(boot.statistics)

# First method of extracting a confidence interval
bootstrap.SE = sd(boot.statistics)
height.mean - 1.96 * bootstrap.SE
height.mean + 1.96 * bootstrap.SE

# Second method of extracting a confidence interval
quantile(boot.statistics, c(0.05, 0.95))
```

FIGURE 4.13: R code for computing a bootstrap confidence interval.

Bootstrap resampling is a **non-parametric method**, because it does not require assumptions about the distribution of the population or data to work (especially the second method, which avoids value 1.96 and the appeal to the Central Limit Theorem). Notice how the second method above does not require the numbers 1.96 obtained from the normal distribution. As many datasets in the real world (and many in my class) are not normally distributed, non-parametric methods are critical for applied data analysts. Rather than mathematical approximations using the classical probability distributions, resampling leans on modern computing power, much like a Monte Carlo simulation. Many examples of bootstrapping in non-normal settings may be found in [316], which also comes with a fantastic and freely available software called StatKey that allows students to carry out the resampling process. Techniques to carry out resampling in R are provided in [256].

If the dataset is not representative of the population (e.g., if it is biased), then bootstrap resampling will not produce confidence intervals that contain the true mean 95% of the time. However, in this case, the classical approach to computing confidence intervals (described in the previous section) will also face difficulties. If the sample size, n, is small, then both methods result in wider intervals, but in theory these intervals should be correct 95% of the time. If the sample size is small and the data is skewed, then both methods will be wrong. There is an alternative method, called the Bootstrap T, that can cope with such a situation [217].

An additional benefit of teaching with bootstrap methods is that one can delay probability distributions until later in the semester, and can de-emphasize the sampling distribution, one of the more conceptually challenging topics for students. It is possible to arrange a course that teaches bootstrap resampling before (or even in lieu of) parametric inference [317]. For more about the pedagogical advantages of teaching via randomization-based methods, see [99, 465, 466, 507, 508, 513].

4.5 Inference

Confidence intervals are not our only tool to reason about an unknown population from a random sample. Another core area of statistics is inference. **Inference** refers to a set of procedures to assess claims about a population parameter, based on sample data. The claim about the parameter (that we wish to test) is called the **null hypothesis**, and the sample data is used to produce a **test statistic** and then a ***p*-value**. The p-value tells how likely the test statistic would be for a different random sample of size n, if the null hypothesis were true. A sufficiently low p-value represents evidence that the null hypothesis is unlikely to be true, and leads the researcher to reject the null hypothesis.

This is analogous to a proof by contradiction, where one assumes a statement is true, then applies logical deduction until a contradiction is reached. That contradiction demonstrates that the initial assumption must have been wrong. As statistics is intimately tied to random error, one will never be certain that the null hypothesis is wrong. Nevertheless, a low p-value represents a probabilistic contradiction, i.e., a strong departure from what we would expect, based on the null hypothesis. After introducing the basics of inference in Sections 4.5.1 and 4.5.2, in Section 4.5.3 we will discuss the types of errors that can occur when doing statistical inference.

4.5.1 Hypothesis testing

In statistical inference, we wish to test a hypothesis about a population, based on a random sample. In this section, we will carefully carry out one example of statistical inference related to the mean of a population (and we provide R code later in Figure 4.16), and then we will explain how to do inference for other population parameters. Throughout, H_0 will denote the null hypothesis one wishes to test.

4.5.1.1 First example

Often, the null hypothesis will be a hypothesis based on a previous study. For example, many of us grew up with the idea that a healthy human body temperature is 98.6 degrees Fahrenheit. For a given population, e.g., all college students in the United States, a natural null hypothesis to test would be $H_0 : \mu = 98.6$. Before testing this hypothesis, we should also decide on an **alternative hypothesis**. The default choice is a **two-sided test**, where the alternative hypothesis is $H_a : \mu \neq 98.6$. If we have some additional information, e.g., that thermometers from the last century tended to output measurements that were too high, then we might do a **one-sided test** and test $H_a : \mu < 98.6$.

To test H_0, we would gather a sample of n people, measure the temperature of each, and compute the sample mean, $\overline{x}$, and sample standard deviation, s. In a two-sided test, if $\overline{x}$ is sufficiently far away from 98.6, we would take this as evidence that the null hypothesis is unlikely to be true. In the one-sided test above, the *only* way we could reject H_0 is if the sample mean in our data was sufficiently *below* 98.6. The critical question is: how far away does $\overline{x}$ need to be from 98.6 before we can conclude the null hypothesis is probably false?

The answer to this question depends on both n and on the variance in our dataset, just as the width of a confidence interval did. In fact, we can use a confidence interval to carry out a hypothesis test. If 98.6 is not in the confidence interval

$$\left[\overline{x} - 1.96 \cdot \frac{s}{\sqrt{n}},\ \overline{x} + 1.96 \cdot \frac{s}{\sqrt{n}}\right],$$

then it means one of two things. Either 98.6 *is* the population mean, μ, and we are in the unlucky 5% of samples that produce bad confidence intervals, or 98.6 is not the population mean. Of course, there is nothing magical about the number 95%. As we have seen, working with 95% confidence intervals means that we will be wrong 5% of the time, and similarly this test will mistakenly reject a true null hypothesis 5% of the time. In general, it is important to fix a **significance level**, α, before starting a hypothesis test. This α will be the probability that we will mistakenly reject a true null hypothesis, with a random sample of data. So, if $\alpha = 0.05$, we work with a 95% confidence interval, and in general, we work with a $(1 - \alpha) \times 100\%$ confidence interval.

Another way to carry out a hypothesis test is to write down a **test statistic** based on the data and on the null hypothesis. In our present example, the test statistic would be

$$z = \frac{\overline{x} - 98.6}{s/\sqrt{n}}.$$

Under the null hypothesis (i.e., assuming it is true), and assuming n is large enough so that we can apply the Central Limit Theorem, the sampling distribution is $N(98.6, \sigma/\sqrt{n})$ distributed. The formula above converts a point in the sampling distribution, $\overline{x}$, to the corresponding point z on a $Z = N(0, 1)$ distribution, by shifting and scaling (using s as an estimator for σ). The number, z, therefore tells us how many standard deviations $\overline{x}$ is away from 98.6. In any normal distribution, about 95% of observations are contained within two standard deviations from the mean (technically, within about 1.96 standard deviations from the mean), and 99.7% of the observations are within 3 standard deviations of the mean. If we are doing a two-sided test, we define $p = 2Pr(Z > |z|)$, the probability that Z takes a value as extreme as z.

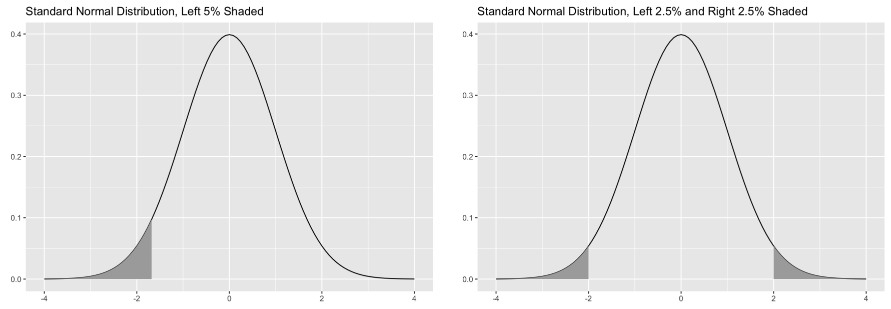


FIGURE 4.14: Standard normal distribution with 5% of its area shaded in one tail ($\alpha = 0.05$, left) and the same distribution with 5% its area shaded again, but split evenly between two tails (right).

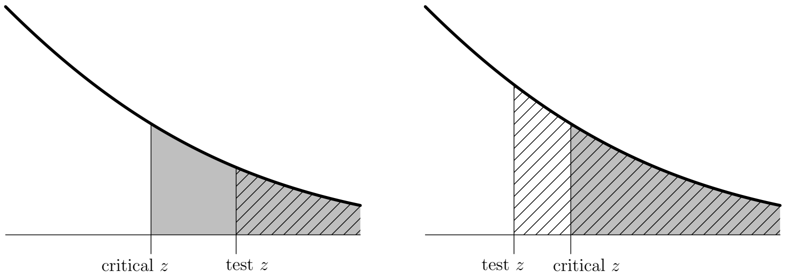


FIGURE 4.15: Comparing the test statistic to the critical value in a one-sided hypothesis test. In the image on the left, the null hypothesis would be rejected, but in the image on the right, we would fail to reject.

Equivalently, this is the probability of getting a value $\overline{x}$ as far away from 98.6 as we observed, if the null hypothesis were true. We refer to p as the **two-sided tail probability**, and display an example on the right of Figure 4.14.

If we were doing a one-sided test, with an alternative hypothesis $H_a : \mu < 98.6$, then we would define $p = Pr(Z < z)$, and refer to it as a **one-sided tail probability** (as on the left of Figure 4.14). Either way, we call p the **p-value** of the test. If $\overline{x}$ is close to 98.6, then the test statistic z will be close to zero, and so this probability p will be large. However, if $\overline{x}$ is far from 98.6, then p will be small.

If the p-value is less than α, we reject the null hypothesis. This situation occurs on the right of Figure 4.15, since the area for p, shaded in gray, is contained in the area for α, shaded with lines (i.e., the tail probability past the cut-off z value). Of course, it could be the case that the null hypothesis really is true, and we were simply unlucky in getting a sample that yielded an extreme value of $\overline{x}$. If the null hypothesis were true, we would expect to be unlucky in this way with probability α. Hence, α represents our willingness to

```
alpha = 0.05
mu0 = 67.5
xbar = mean(Galton$height)
s = sd(Galton$height)
n = nrow(Galton)
t = (xbar - mu0) / (s/sqrt(n)) # This is -6.18
p = 2*pt(t, df = n-1)      # pt gives the left tail probability

xbar; t; p
```

FIGURE 4.16: R code to carry out a null hypothesis significance test.

mistakenly reject a true null hypothesis. This may sound as if we should choose α to be zero, and never make such a mistake. However, if α were zero, then we would never reject the null hypothesis, even if it were wrong, so we would be making a different mistake. We will talk about these two types of errors in Section 4.5.3. Note also that there was nothing special about the value 98.6 in the discussion above. The same technique could test the hypothesis $H_0 : \mu = \mu_0$ for any fixed value μ_0.

The R code in Figure 4.16 carries out a two-tailed hypothesis test, again on the Galton dataset. We test the null hypothesis, $H_0 : \mu = 67.5$, that the true population height is 67.5 inches. We work at the $\alpha = 0.05$ significance level. The code computes a t-statistic $t = -6.18$ according to the formula discussed above. The code computes the p-value using the function `pt`, which gives the tail probability to the left of t. We multiply by 2 because we are doing a two-tailed test. The last line in the R code carries out three commands, printing each of the three quantities $\overline{x}, t$, and p.

4.5.1.2 General strategy for hypothesis testing

What were the ingredients of the procedure we have just laid out? We needed:

1. to know our population and which parameter we wanted to study,
2. to state our null and alternative hypotheses,
3. to be aware of any assumptions, e.g., that the Central Limit Theorem could apply so that the sampling distribution is normally distributed,
4. to know the formula for the standard error,
5. to compute the test statistic,
6. to get a tail probability from the $N(0, 1)$ distribution (which is easy to do with software or with a table of statistical values), and
7. to make a decision, rejecting the null hypothesis if and only if $p < \alpha$.

This same procedure works for every single parametric hypothesis test in statistics, though sometimes the sampling distribution is not normally distributed. For example, if the sample size, n, is less than 30, then the Central Limit Theorem tells us to use the t-distribution with $n-1$ degrees of freedom, just as we did for confidence intervals. When testing a hypothesis of the form $H_0 : \sigma = \sigma_0$, the sampling distribution is χ^2-distributed, and when testing if two populations have the same standard deviation, the sampling distribution is F-distributed.

4.5.1.3 Inference to compare two populations

Sometimes, the null hypothesis is not based on a previous study, but is instead based on comparing two populations. To compare the means of two populations (e.g., the mean salary of men vs. the mean salary of women), a **two-sample t-test** can test the hypothesis $H_0 : \mu_1 = \mu_2$. First, we restate the null hypothesis as $H_0 : \mu_1 - \mu_2 = 0$. Then we test H_0 using a test statistic based on $\overline{x_1} - \overline{x_2}$. The same formula for the standard error is used for confidence intervals in the setting of two populations. To compare two proportions, we test $H_0 : p_1 = p_2$. Rejecting the null hypothesis in such tests means concluding the populations are statistically significantly different.

Sometimes, the data consists of pairs (x_1, x_2), and each pair represents two observations of the same individual, under different conditions. For example, x_1 might be the amount of weight an individual can lift with his/her right hand, and x_2 the amount for the left hand. In this case, the paired structure is important for explaining some of the variability in the dataset (because different people have different baseline strengths). Another example would be if x_1 represented a score on a pre-test, and x_2 a score on a post-test, to measure the impact of an educational innovation. A **paired t-test** can test the hypothesis $H_0 : \mu_1 = \mu_2$, but with a different formula for the standard error than for a two-sample t-test, that can be found in standard statistical texts [11, 409]. For a first course in statistics, using statistical software, it is not essential that students memorize formulas for standard errors, and all the traditional formulas work in the same way (that is, they depend on the standard deviation in the dataset, and the standard error shrinks as n grows, as discussed in Section 4.5.1.1). It is far more important that students be able to identify which test to use for a given situation.

One of the more important null hypotheses we can test is whether or not two quantitative variables are correlated. Let ρ denote the population level correlation, and let r denote the sample correlation. There is a test for the hypothesis $H_0 : \rho = 0$, based on the standard error of the sampling distribution of sample correlations. Rejecting the null hypothesis means that there is statistically significant evidence of a relationship between the variables.

There is an equivalent hypothesis test (meaning, with the same t-statistic and p-value) for the hypothesis $H_0 : \beta_1 = 0$, based on the standard error of the regression slope β_1. In order to conclude the sampling distribution is normal (or t-distributed for small sample sizes), one needs a few more assumptions. As always, the data must represent a random sample. Additionally, the residual errors (i.e., the distances $y_i - \hat{y}_i$) must be independent and identically distributed, with finite variance. The independence part of this assumption says there cannot be **autocorrelation** in the data. This can fail in, for example, data drawn from a time series, in which values depend on values from previous moments in time, or in data drawn from different geographic positions, if data from one position depends on data at nearby geographic positions. It is not essential for students to be able to prove that the sampling distribution for regression coefficients is approximately normal, but students should get used to checking the regression assumptions (as in Section 4.6.4). One can still do linear regression if these new assumptions fail, but one should not attempt to interpret t-statistics or p-values. We will discuss analogous tests for multivariate linear regression in Section 4.6.3.

4.5.1.4 Other types of hypothesis tests

There are many more statistical tests that data scientists may encounter, but all work according to the principles laid out in this section. For instance, there are hypothesis tests for the median, or to test hypotheses on non-normal populations based on small sample sizes, such as the Wilcoxon, Mann-Whitney, and Kruskal-Wallis tests [11]. For examples of a different sort, there are statistical tests to test whether a given dataset comes from a normally distributed population (e.g., Shapiro-Wilk, Anderson-Darling, or Kolmogorov-Smirnov tests) [463], whether or not there is autocorrelation (e.g., Durbin-Watson test) [514], whether or not there is heteroscedasticity (e.g., Breusch-Pagan test) [514], and whether or not a dataset appears random (e.g., the runs test for randomness, as in Section 4.6.4) [409]. For these four situations, the researcher is hoping to fail to reject the null hypothesis, to have more confidence in assumptions required for various models and tests. It is important to note that one can never *accept* the null hypothesis as a result of a significance test. One can only *fail to reject* the null hypothesis. Each test considers one way that the null hypothesis could fail, but no test can consider all possible ways. Students should get into the habit of making an argument that the requisite assumptions are satisfied.

All statistical inference tests result in p-values. We have seen that p-values represent tail probabilities based on the test-statistic, and the choice of a one-sided or two-sided test determines whether or not we consider one tail or both tails of the sampling distribution.[13] Because the threshold α must be set before the test is carried out, one should not try to interpret the magnitude of a p-value,[14] other than to determine whether or not $p < \alpha$. Furthermore,

[13]There are also one-sided variants of confidence intervals, that reject the null hypothesis if and only if a one-sided significance test has a p-value below the cutoff (e.g., $p < 0.05$) [11].

[14]That is, one should not get into the habit of interpreting a p-value of 0.0001 as more significant than a p-value of 0.001.

when reporting the results of a statistical analysis, the American Statistical Association strongly recommends that a researcher report more than just the p-value [494]. Ideally, one should also report the sample size, how the data was gathered, any modifications made to the data (including cleaning and removal of outliers), issues of bias in the data, and confidence intervals. Ideally, data should be made publicly available to aid efforts to reproduce research. We will return to the issue of reproducibility and error in Section 4.5.3.

We conclude this section with a word regarding the teaching of inference. The logic of inference can be difficult for students, and even experts sometimes have difficulty pinning down precisely the population one has data about. It takes practice to speak correctly about inference, and the reader is encouraged to discuss inference with colleagues before first teaching it, both to get into the habit of speaking correctly, and to begin to recognize difficulties students will face. In the next section, we will look at a randomization-based approach to inference, which is useful for test statistics that fail to be well-approximated by the Central Limit Theorem. An additional benefit of the next section is that it solidifies the logic of the previous section. We find that including both approaches helps students gain an intuition for inference.

4.5.2 Randomization-based inference

Just as bootstrap resampling provides a non-parametric approach to confidence intervals, randomization-based inference leverages modern computing power and resampling to provide a non-parametric approach to hypothesis testing. We describe the procedure below, and carry it out in R in Figure 4.17. As in Section 4.4.3, we must assume our sample is representative of our population. We then use a Monte Carlo simulation to compute a p-value as the number of successes divided by the number of trials.

For example, if given paired data $(x_1, y_1), \ldots, (x_n, y_n)$, we can test the hypothesis that the correlation ρ is zero as follows. (One could also make minor modifications to the discussion that follows to consider a difference of means $\overline{x} - \overline{y}$.) First, we write down the sample correlation r of this dataset, and select a p-value cutoff α (let's say $\alpha = 0.05$). If the true correlation were zero, then the pairing didn't matter. So we use our dataset to generate thousands of other datasets by randomly pairing x_i values to y_j values (i.e., breaking the given pairings). For example, thinking back to the Galton dataset of Section 4.3.1, Table 4.2 gave a snippet of the data, with $(x_1, y_1) = (70, 66), (x_2, y_2) = (70.5, 68)$, etc. The thousands of new datasets $(X^1, Y^1), \ldots, (X^k, Y^k)$ obtained from the procedure just described (bootstrap resampling) might look like Table 4.3.

For example, in the first resample, the new first pairing is $(x_1, y_1) = (70, 60)$, obtained by randomly pairing $x_1 = 70$ with $y_7 = 60$ in the original dataset. The second new pairing is $(x_2, y_2) = (70.5, 63)$, and y_2 is y_9 in the original dataset. Similarly, in the last resample (X^k, Y^k), the first pairing

TABLE 4.3: k bootstrap resamples.

X^1	Y^1	$\dots$	X^k	Y^k
70	60		70	64
70.5	63		70.5	68
68	60		68	60
66	66		66	60.5
67.5	64		67.5	60
65	64		65	64
70	61		70	66
69	61		69	63
70	60.5		70	61
68.5	68		68.5	61

is $(x_1, y_1) = (70, 64)$, obtained by randomly pairing $x_1 = 70$ with $y_5 = 64$ in the original dataset.

If the null hypothesis were true, then the original pairing did not matter, because the xs and ys are unrelated. Thus, under our assumption that the sample is representative of the population, we can pretend that these new datasets are drawn from the population, rather than created from the sample. With thousands of datasets, we can make an empirical sampling distribution, now called the **bootstrap distribution**. To do so, for each random pairing, we write down the correlation obtained, and we gather all the correlations together in a dotplot. In this distribution, some correlations will be positive, and some will be negative, because each is obtained by a random pairing of the original data. The average of all these correlation values approaches zero as the number of bootstrap resamples increases. Hence, the mean of the bootstrap distribution is close to zero.

The standard deviation of the bootstrap distribution will be based on how much variability there was in our original dataset, and also the sample size n. This standard deviation approximates the standard error of the sampling distribution. To obtain the p-value, we simply count the number of bootstrap samples whose correlation was larger than our original sample correlation r. This count is the number of successes in our Monte Carlo simulation. For example, if we created 3000 bootstrap samples (from 3000 random pairings), and if 36 had a correlation as large as our original sample correlation r, then the *bootstrap p-value* of a one-tailed test would be $36/3000 = 0.012$. This is lower than the threshold $\alpha = 0.05$ that we selected, so we reject the null hypothesis, and conclude that the true population correlation is nonzero.

The R code in Figure 4.17 carries out the procedure just described, again on the Galton dataset. We test the null hypothesis that the true correlation between a mother's height and a father's height is zero, $H_0 : \rho = 0$. We work at the $\alpha = 0.05$ significance level. We use the function `cor` to compute the sample correlation. We use the `shuffle` command for the permutations as in

```
alpha = 0.05
corGalton = cor(mother~father,data=Galton)
corGalton   # The correlation is 0.07366461

bootstrap.dist2 = do(1000)*(cor(shuffle(mother)~father,
                                data=Galton))
bootstrap.dist2 = as.data.frame(bootstrap.dist2)
dotplot(~result, data=bootstrap.dist2)
# Several bootstrap correlation values are > 0.0737
ladd(panel.abline(v=corGalton, col="red"))

# p-value
numBigger = pdata(corGalton,bootstrap.dist2)
upperTail = numBigger/nrow(bootstrap.dist2)
pvalue = 2*upperTail
pvalue
```

FIGURE 4.17: R code for randomization-based inference.

Table 4.3. We use the `do(1000)` command to repeat the procedure "shuffle then compute correlation" $k = 1000$ times. The `ladd` command lets us add a vertical line to display the p-value, which we then compute as the number of bootstrap correlations larger than the original correlation $r = 0.07366461$, divided by 1000.

In the end, the code produced a p-value of 0.028 when we ran it. Because of the random shuffle command in the code, different runs of the code above can yield different p-values. Thankfully, bootstrap p-values asymptotically converge. Because of the reliance on shuffling the data, tests like this are sometimes called **permutation tests**. The code above should be compared with Figure 4.13 to solidify understanding of bootstrap procedures. Figure 4.18 shows one possible result of running the code. Each dot represents a correlation obtained via the `shuffle` command. The p-value is obtained by counting the number of dots in the tail, and multiplying by 2. An alternative approach would be to count dots above r and below $-r$.

Empirical studies have shown that, if the dataset really is distributed according to a classical distribution, then the bootstrap p-value (computed via either of the approaches just described) is usually close to the p-value one would obtain via parametric methods [217]. Randomization-based inference avoids the need for the sampling distribution (witnessed by the lack of `pnorm` or `pt`, unlike Figure 4.16), and can be used for all the hypothesis tests of the previous section. These tests can be carried out using the website StatKey [315], and many examples of randomization-based tests can be found in [316].

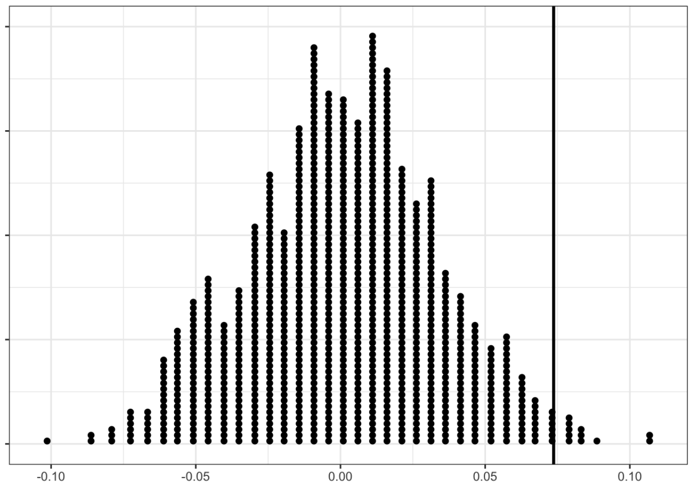


FIGURE 4.18: Bootstrap approximation to the sampling distribution, obtained via randomization-based inference.

4.5.3 Type I and Type II error

In Section 4.5, we saw that the parameter α controls the probability of wrongly rejecting a true null hypothesis. This kind of error is called a **Type I error**. It occurs if the data is not representative of the population, which can occur due to chance when a random sample is collected. A Type I error occurs when the test statistic is far from what would be expected under the null hypothesis, even though the null hypothesis is true. The other kind of error, a **Type II error**, is when the null hypothesis is false, but the researcher fails to reject it. We denote the probability of this type of error by β. Unlike α, the researcher can never know β, but thankfully β decreases as the sample size increases. Furthermore, there is a trade-off between α and β: if the researcher decreases α, this makes it harder to reject the null hypothesis, and consequently increases β. If a researcher has access to a previous study (and hence some idea of how badly the null hypothesis might fail, as well as a sense of the variance), there are techniques for choosing α to affect a good trade-off [110]. In practice, one often cannot control β, and hence the most important learning goal for students is simply to be aware of it.

4.5.4 Power and effect size

The **power** of a statistical study is defined to be $1-\beta$, i.e., the probability that it does *not* make a Type II error. Equivalently, this is the probability that the statistical study correctly detects an effect (i.e., that the null hypothesis is false). The terminology comes from medical science, where an effect means a treatment is working. The power rises with both n and α. It also depends on the variability in the population and on the **effect size**, i.e., the difference between the hypothesized mean μ_0 and the true mean μ. For a two-sample test, the effect size is the true difference between the population means, $\mu_1 - \mu_2$. In practice, we do not know the effect size. However, researchers sometimes know how large of an effect would be practically significant. This is especially true in medical studies, where changing medication can cause complications, but may be worth it depending on how much improvement one would expect from the change. For researchers seeking funding, it is important to know how to obtain a sample size, n, to achieve a given effect size (equivalently, a given power). For problems like this, the confidence interval formulas from Section 4.4.2 can be used. For example, if a researcher wanted to detect an effect size of 2 (e.g., a reduction of 2 on a 1–10 pain scale), then the width of the confidence interval would need to be 4, so that the null hypothesis would be rejected whenever the sample mean was farther than 2 from the hypothesized mean. If a researcher had an estimate, s, of the standard deviation (e.g., from a previous study), and was willing to accept a 5% probability of a Type I error ($\alpha = 0.05$), then this would mean solving for n in $1.96 \cdot \frac{s}{\sqrt{n}} \geq 2$. Such a calculation could be used to determine how many individuals would need to be in the study, and hence how much money to request in a grant application.

Historically, statistics often faced the problem of not having enough data. However, in the world of big data, it is possible to have *too much* data. Giant sample sizes mean that statistical tests have power very close to 1. Consequently, statistical tests can detect very small effect sizes. For example, if one had data on 100 million Americans, it might be possible to detect a difference in salary smaller than one penny. In situations like this, statistical significance does not imply practical significance. (That is, analyzing differences of \$0.01 in salary is not worth the effort.) If one has a large amount of data, and knows the desired effect size, it may be prudent to take a random sample of the data of a size n, chosen so that being statistically significant (i.e., rejecting the null hypothesis) means the same thing as being practically significant (i.e., detecting the given effect size).

4.5.5 The trouble with p-hacking

It is an unfortunate fact that academic publishing tends to bias in favor of statistically significant results, such as when a new treatment *does* result in a difference from an old treatment. Consequently, there are market forces pressuring researchers to find significant results. At times, this has led researchers to test many hypotheses, until finding a p-value less than 0.05. This practice is known as **p-hacking**, and is a large part of the reason for the current Reproducibility Crisis, wherein modern researchers are unable to reproduce results from previously published papers [447]. Even if no p-hacking ever occurred, one would expect that about 5% of papers would falsely reject a true null hypothesis, by the definition of α and the common choice of $\alpha = 0.05$. However, in some fields, a proportion as high as 54% of papers cannot be replicated [426].

Let us discuss the reason that p-hacking leads to mistakes. Imagine a dataset with one response variable (say, whether or not a person developed cancer), and many explanatory variables, such as how much this person consumed bananas, yogurt, etc. Suppose that none of these substances actually contribute to an increased risk of cancer. Nevertheless, due to chance alone, a dataset might show a correlation between risk of cancer and consumption of some innocuous food. If just one hypothesis test were conducted, e.g., to see if consumption of bananas is associated with increased risk of cancer, then one would expect to correctly fail to reject the null hypothesis 95% of the time, and would expect that in 5% of datasets the null hypothesis would be (wrongly) rejected. Now suppose twenty different foods were tested for a possible association with cancer. With a probability of 0.05 of being wrong, and 20 tests, one would expect to be wrong on one of the tests. Even if there is no effect present, doing sufficiently many tests can always lead to a statistically significant p-value.

It should be stressed to students that they should decide which precise hypothesis test(s) to conduct before actually carrying any tests out, that they resist the temptation to add new tests if prior tests were not significant, and that they maintain ethical standards throughout their careers. It may help to remind them that reputation is more important than whether or not one single experiment leads to a publishable result. Furthermore, the sheer volume of research going on today means that it is very likely someone will attempt to replicate their results, and that any oddities in the analysis will eventually come out. If a student insists on testing more than one hypothesis, a Bonferroni correction (Section 4.6.3) can control the overall probability of a Type I error.

4.5.6 Bias and scope of inference

Before conducting a hypothesis test, it is essential to know what population one wishes to study. In the case of "found data" (rather than data a researcher collected), it is often the case that the population represented by the data is not exactly the population the researcher hoped to study. For example, if one has a dataset consisting of all individuals who died from a drug overdose in Ohio, then it is not possible to make statistical inferences related to other states. It is also not possible to make conclusions about what factors might help an individual overcome addiction, since none of the individuals in the dataset did. In a case like this, one is arguably not doing statistics at all, since the data represents the entire population of individuals who died from a drug overdose in Ohio. Since there was no random sampling, the classical uses of confidence intervals and hypotheses tests do not apply. Instead, one can conduct exploratory data analysis, find summary information and visualizations, and fit statistical models. If one wishes to use the data to predict future observations, special techniques from the field of **time series analysis** are required. These will be discussed briefly in Section 4.6.4.

4.6 Advanced regression

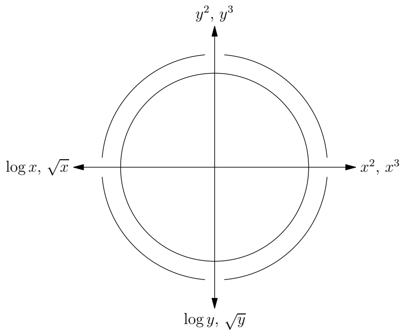
In Section 4.3, we introduced some basic statistical models, including ordinary least-squares regression for relating two quantitative variables. In this section, we will discuss more advanced techniques for regression, how to factor in more than one explanatory variable, and what to do when regression assumptions fail to be satisfied. All of these are essential skills for a data scientist working with real-world data.

4.6.1 Transformations

We saw in Section 4.3.1 that linear regression should be used only when there is a linear relationship between the variables x and y. In the real world, many variables exhibit more complicated relationships. For example, if y represents salary data, then the distribution of y values is probably skewed by extremely wealthy individuals. In such situations, it is often wise to apply the log function to y. Many times, a scatterplot will reveal a nonlinear relationship between x and y, and transformations, such as taking the log, square root, or square of either x or y can result in linear relationships. Ideally, the decision of which transformation to apply will be based in some theory of how the two variables should be related, or based on previous practice (e.g., the use of log in econometrics). In less ideal situations, a researcher can use Tukey's Bulging Rule [471], as shown in Figure 4.19, to try different transformations until a linear relationship is found. The idea is to transform x or y according to Figure 4.19 (i.e., based on the shape of the scatterplot), iteratively, until the relationship is linear (keeping in mind the Principle of Parsimony that models should not be made more complicated than necessary). For examples, we refer the reader to [471].

Trying many transformations is dangerous, as one runs the risk of overfitting the model to the sample data, rather than to the true relationship in the model. A similar problem was discussed in Section 4.5.5. Cross-validation (Section 4.3.2) can be used to prevent overfitting, by finding the transformations based on a random subset of the data, and then verifying that the remaining data points fit the transformed model.

Sometimes, the relationship between x and y is a polynomial but not linear. In such a situation, **polynomial regression** should be used, as shown in Figure 4.20. For example, we might use a model of the form $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$ when the relationship is quadratic. The decision to fit such a model should be based on the scatterplot and attributes of the variables that the data



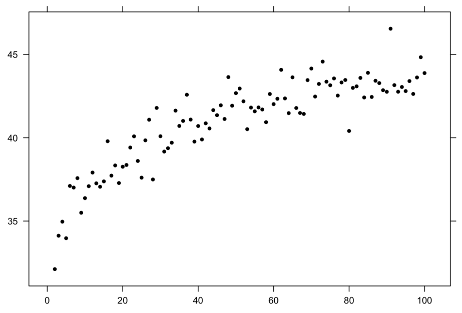


FIGURE 4.19: Tukey's Bulging Rule (left) and a scatterplot in need of transforming (right).

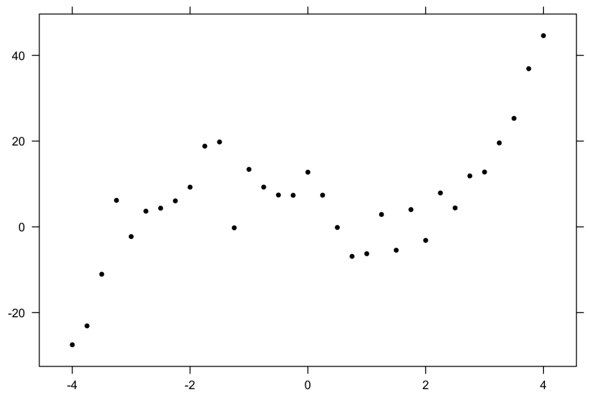


FIGURE 4.20: Data that might be well fit by polynomial regression.

scientist knows from familiarity with the application domain. The procedure for finding the β estimates and assessing the model is a special case of multiple linear regression (discussed in Section 4.6.3), where we create a new variable x^2 in our dataset by simply squaring the value of x in every row.

4.6.2 Outliers and high leverage points

An **outlier** is a point that is far away from the other points, and there are many ways to quantify "far away." The most popular approach, due to Tukey and mentioned earlier in Section 4.2, is to use the **inter-quartile range (IQR).** Formally, this means computing the first quartile Q_1 (with 25% of the data below this point), median, and third quartile Q_3 (with 75% of the

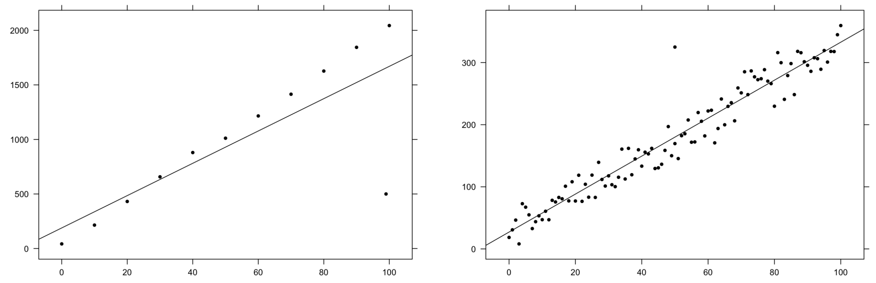


FIGURE 4.21: A small dataset with a linear model fit to it, and a high-leverage data point on the bottom right of the figure (left). A large dataset with a linear model fit to it, and an outlier near the top center of the figure that is not a high-leverage data point (right).

data below this point), then defining the IQR as $Q_3 - Q_1$. An outlier is then defined as any point that is less than $Q_1 - 1.5IQR$ or more than $Q_3 + 1.5IQR$.

With this definition, it is clear that outliers can occur due to chance alone. An outlier is not the same as an impossible point or a misrecorded point. Blindly throwing away outliers will reduce the variance in the data, and may lead to wrong conclusions. However, sometimes an outlier can skew a statistical model, as seen on the left of Figure 4.21. Because the regression coefficients are based on the residual sum of squares, large residuals (e.g., due to outliers) have an outsized impact on the regression coefficients. Another way to combat this impact is to work with **studentized residuals** (also known as jackknife residuals), computed with the outlier removed. This prevents a single outlier from causing overly large confidence intervals, but removing outliers should be done with caution. Another way to combat the same problem is explained in Section 4.7.2.

Note that the regression line $y = \beta_0 + \beta_1 x$ always passes through $(\overline{x}, \overline{y})$, by definition of β_0. Consequently, adding a data point (x, y), where y is an outlier and x is close to $\overline{x}$, does not change the regression coefficients by much (Figure 4.21). The slope will not change at all if $x = \overline{x}$. However, adding a new point (x, y) far away from $(\overline{x}, \overline{y})$, even if neither x nor y is an outlier, can radically change the regression coefficients, as the same figure shows.[15]

When a point (x, y) causes a large change in the regression coefficients, as shown on the left of Figure 4.21, we call it an **influential point**. A technique for detecting influential points will be explained in Section 4.7. In the case of an influential point, a researcher can consider reporting models with and without the point in question. It is important to resist the temptation to purge the dataset of points that don't agree with the model, knowing that

[15]Adding a new point far away from $(\overline{x}, \overline{y})$ could also leave the regression coefficients unchanged, e.g., if the new point is on the old regression line.

we as researchers face implicit pressure to find a model that fits well. Often, influential points or outliers are worthy of further study, as individual cases. Sometimes, they represent a subpopulation, and can be omitted by restricting attention to a different subpopulation. For example, in height data, an outlier might represent an individual with pituitary gigantism, so that their height is far outside what would normally be expected. Finding the outlier helps the researcher realize that this condition exists, but makes it difficult to create a model that fits the dataset at hand. By setting the outlier aside for further study, the researcher can get a model that fits the remaining data, and can be used for predictions for individuals without pituitary gigantism. This situation is similar to the left of Figure 4.21.

4.6.3 Multiple regression, interaction

Many interesting real-world response variables cannot be explained by a single explanatory variable. For example, the value of a stock in the stock market is a function of many variables, not just time. For a case such as this, an appropriate statistical model would be $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \epsilon$, where ϵ is some random error term. Given a dataset of tuples, where the i^{th} tuple is denoted $(y_i, x_{1i}, x_{2i}, \ldots, x_{ki})$, it is easy to find the best-fitting β coefficients using statistical software. We denote these coefficients, $\hat{\beta}_j$, to distinguish them from the population-level coefficients β_j. These estimated coefficients will minimize the sum of squared residuals $(y_i - \hat{y}_i)^2$, where $\hat{y}_i$ is the y value predicted by the model, i.e., $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \cdots + \hat{\beta}_k x_{ki}$. Formulas for the $\hat{\beta}_j$, known as the **normal equations**, could be obtained from a multivariate optimization problem, but this derivation is not essential for introductory students. (See Sections 3.2.2.1 and 8.5.3.) The process of finding the estimated coefficients $\hat{\beta}_j$ is known as **fitting the model**. With statistical software, it is also easy to produce a regression table, like the one shown in Table 4.4, containing information about how well the model explains the variability in y, and which variables appear to be significant. In this example, we use the length and gender of a turtle to predict its height.

In such a model, we interpret the slope, β_j, as the average change in y resulting from increasing x_j by 1, and *holding all the other variables constant.* In other words, β_j is the partial derivative of y with respect to x_j, assuming that all variables are related linearly (i.e., the true population can be modeled as a hyperplane plus random noise). Table 4.4 tells us that we would expect a change of length of 1 cm to be associated with a change in height of 0.34446 cm, on average. Similarly, there is a difference between the heights of male and female turtles of -3.52558 cm, on average. Both β coefficients are statistically significant, as is the model as a whole (with an F-statistic of 517.6).

In practice, we often have more variables than we need, and we need to determine which collection of x_j to include in the model. There are many procedures for this. One, **backwards selection**, first fits all possible values x_j, then iteratively removes variables that are not significant, until only significant

TABLE 4.4: Example linear regression table output by the statistical software R.

```
Call:
lm(formula = height ~ length + gender, data = turtle)

Residuals:
    Min      1Q  Median      3Q     Max
-3.3384 -1.0071 -0.0643  1.1902  4.4280

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.18082    2.06826   2.505   0.0159 *
length       0.34446    0.01498  23.001  < 2e-16 ***
gendermale  -3.52558    0.60743  -5.804  6.1e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.745 on 45 degrees of freedom
Multiple R-squared:  0.9583,    Adjusted R-squared:  0.9565
F-statistic: 517.6 on 2 and 45 DF,  p-value: < 2.2e-16
```

variables remain. Another, **forwards selection**, starts by fitting all possible models with only one explanatory variable, picks the best (e.g., whichever maximizes R^2, a quantity we will define shortly), and then tries all ways to extend this model with a second variable, then with a third, etc. As with any model selection procedure, to avoid overfitting the model to the data, it is recommended to use cross-validation (Section 4.3.2).

In practice, we often prefer simpler models, and there are good theoretical reasons to avoid including unnecessary variables. For instance, if two variables x_1 and x_2 are both effectively measuring the same thing, then including both in the model will result in **multi-collinearity**. This inflates the variance in the model and makes it difficult to correctly interpret the regression coefficients, t-statistics, and p-values. Fitting a model of the form $y = \beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_k x_k+\epsilon$ is equivalent to fitting a k-dimensional hyperplane to our data. Ideally, the k explanatory variables would all be orthogonal. If we include a variable that is mostly explained by other variables in the model, then this makes the model more susceptible to variation, since a small change in the data can lead to a big change in the estimated coefficients. This situation is often referred to as a “tippy plane.” If the explanatory variables are linearly dependent, then the algorithm for finding the coefficients $\hat{\beta}_j$ will fail, because a certain matrix will not be invertible (Section 4.7). To avoid fitting a model where explanatory variables are too close to being linearly dependent, statistical software can compute **variance inflation factors (VIFs)**, and it is desirable to leave out variables with VIFs larger than 5 [101].

The **coefficient of determination**, R^2, is defined to be the fraction of the variability in y explained by all the explanatory variables x_j. We will see below that R^2 can be defined formally as SSM/SST, where SSM is the variability explained by the model, and SST is the total variability in y. Hence, adding more explanatory variables will never decrease R^2, because it only gives the model more ability to explain variance. However, adding unnecessary variables inflates the variance in the model (e.g., making the model more likely to fit to the random noise than to the true signal) and makes the model unnecessarily complicated. For this reason, researchers often use **adjusted** R^2, which penalizes models that contain unnecessary variables. (See Section 8.3.4.2 for its definition.) When selecting which variables to include, researchers often refer to adjusted R^2 rather than R^2.

Sometimes, understanding the response variable requires understanding two explanatory variables, *and their interaction*. For example, we know that drugs and alcohol interact in the human body, and this frequently leads to drug overdoses. When a statistician desires to model interaction, the most common model is to allow the variables to interact multiplicatively.[16] To model such an interaction, we can include an **interaction term** in the model as follows: $y = \beta_0+\beta_1 x_1+\beta_2 x_2+\beta_{12}x_1 x_2+\epsilon$. Thinking in terms of hyperplanes, including an interaction term means allowing for curvature. We will talk in Section 4.6.5 about another way to think about and teach interaction terms, in the context of categorical variables. When one of the explanatory variables x_1 is binary, then a model of the form $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \epsilon$ represents fitting two parallel lines (with β_1 as the distance between the lines), whereas a model of the form $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \epsilon$ represents two possibly non-parallel lines. For the group where $x_1 = 0$, the line is $y = \beta_0 + \beta_2 x_2$. For the group where $x_1 = 1$, the line is $y = (\beta_0 + \beta_1) + (\beta_2 + \beta_3)x_2$. We will return to this in Figure 4.24.

The astute reader may have noticed that the variable selection techniques described earlier in this section involved looking at p-values for a large number of statistical tests, each of the form $H_0 : \beta_j = 0$, and this violates the best practices described in Section 4.5.5. Indeed, if we tested 20 useless variables at the 5% significance level, we would expect to see one variable appear useful even if it is not. One way to fix this is to use a **Bonferroni correction**. (Another way, that works in the context of ANOVA, is Tukey's HSD, which we discuss in Section 4.6.5.) Briefly, the idea of the Bonferroni correction is to insist on a smaller p-value for each individual test, so that the overall probability of a Type I error is bounded by 5%. If one intends to conduct k hypothesis tests, the p-value threshold should be α/k rather than α. It is then an exercise in probability theory to see that the overall probability of a Type I error (i.e., that at least one of the individual coefficient tests is wrong) is bounded above by α.

[16]Multiplicative interaction is also common in applied mathematics, e.g., it arises in the SIR model in epidemiology, and in the Lotka-Volterra equations modeling populations of wolves and moose on Isle Royale.

An alternative approach is to test all of the regression coefficients at once. Here, the null hypothesis is $H_0 : \beta_1 = \beta_2 = \cdots = \beta_k = 0$. The idea of R^2 can be turned into a test statistic, which is distributed according to the F-distribution, if the regression assumptions are met.[17] The F-statistic and p-value are reported by standard statistical software (in Table 4.4, they are in the bottom row).

The test statistic, F, is based on an analysis of variance. Our goal with a linear model is to explain the response variable, y, and we use residuals to tell how well the model succeeds in this goal. It is therefore natural to study the amount of variability in y that is explained by the model. The following three sums all run from $i = 1$ to n, i.e., over all of our data points. The total variability in y is the **total sum of squares (SST)**: $\sum(y_i - \overline{y})^2$. The residual sum of squares, or **sum of squared error (SSE)**, is $\sum(y_i - \hat{y}_i)^2$. The remaining variability is the variability explained by the model, **SSM**: $\sum(\hat{y}_i - \overline{y})^2$. These three sums of squares are related by one of the most fundamental formulas in all of statistics:[18]

$$SST = SSM + SSE.$$

This sum of squares decomposition immediately allows us to calculate $R^2 = \frac{SSM}{SST}$, which we introduced informally above. It is a theorem, easily proven in a mathematical statistics course, that the square of the correlation, r, is R^2 [409]. Assuming the null hypothesis, that the β coefficients are zero (so that the x_i variables do not explain the response variable), we would expect that the model does no better than the error term at explaining y, i.e., that SSM is close to SSE. Hence, we would expect that SSM/SSE is close to 1. Since SSE and SST are biased estimators for population-level parameters (as in Section 4.3.4), we replace them by unbiased estimates MSE (Mean Squared Error) and MST (Total Mean Square). Similarly, SSR has a corresponding unbiased estimate, MSR (Regression Mean Square). This is analogous to how we needed to divide by $n - 1$ when computing s, but divide by N when computing σ, in Section 4.2. Formally:

$$MSE = \frac{SSE}{n-k-1}, \; MST = \frac{SST}{n-1}, \text{ and } MSR = \frac{SSR}{k}.$$

The F-statistic is defined as $F = MSR/MSE$. A p-value for a given F-statistic can be obtained from a table of values of the F distribution (with degrees of freedom $n - k$ and k), or from statistical software. If the p-value is significant, we conclude that the model as a whole has merit, i.e., some combination of the explanatory variables do explain a statistically significant amount of the response variable. An F-test can also be done using randomization-based inference (Section 4.5.2), by shuffling the data thousands of times, creating a distribution of F-statistics from each shuffle, and then comparing the

[17]In particular, the residuals should be independent and identically distributed, and should either be normally distributed or the sample size should be sufficiently large. In addition, we must assume there is no multi-collinearity or heteroscedasticity.

[18]The reader can try proving this formula by recalling that the β_i were chosen to minimize SSE, so we may assume $\frac{\partial SSE}{\partial \beta_0} = \frac{\partial SSE}{\partial \beta_1} = 0$. With those assumptions, algebra and persistence suffice to complete the proof.

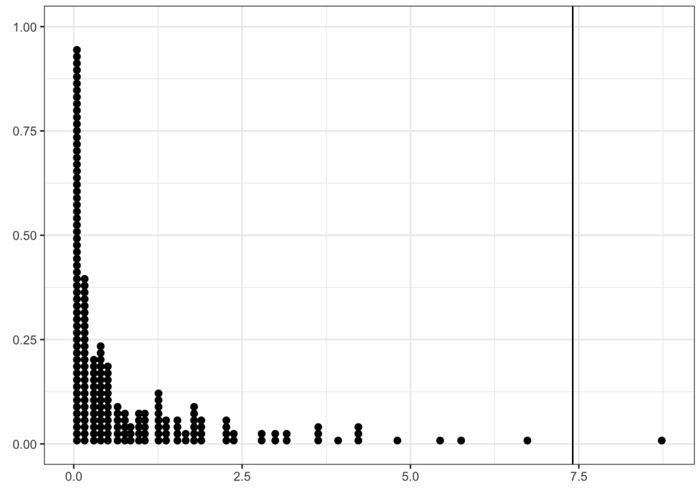


FIGURE 4.22: Result of a randomization-based F-test.

original F-statistic to the distribution created in this way. Figure 4.22 displays the result of such a test (created analogously to the code in Figure 4.17) with a very low p-value.

Another null hypothesis that can be tested analogously is the hypothesis that the full model (featuring all k explanatory variables) has additional value beyond some reduced model (featuring $q < k$ variables). Such a test is called a **nested F-test**, and the test statistic is

$$F = \frac{\frac{\mathrm{RSS}_{\mathrm{Full}} - \mathrm{RSS}_{\mathrm{Reduced}}}{\mathrm{df}_{\mathrm{Reduced}} - \mathrm{df}_{\mathrm{Full}}}}{\frac{\mathrm{RSS}_{\mathrm{Full}}}{\mathrm{df}_{\mathrm{Full}}}},$$

where RSS is the residual sum of squares (determined from either the full model or the reduced model), and df stands for "degrees of freedom" [100]. In this case, a significant p-value means we reject the null hypothesis that the two models have equal merit, and we conclude that the full model has additional merit over the reduced model.

4.6.4 What to do when the regression assumptions fail

The classic assumptions[19] required for linear regression inference state:

1. Linearity: The relationship between each explanatory variable x_i and y is linear.

[19]There does not seem to be any consensus from textbooks on how to phrase these assumptions or how to group them, so the number of assumptions is not well-defined.

2. No perfect multicollinearity: The explanatory variables x_i are linearly independent.

3. No autocorrelation: The residuals (errors) are independent from each other.

4. Homoscedasticity: The variance of the residual errors is constant as a function of the explanatory variables x_i.

5. Randomness: The data are obtained from a random process.

6. Normality: The residuals satisfy a normal distribution.

We have already discussed using scatterplots to check linearity (Section 4.3.1), and the need to avoid perfect multicollinearity (Section 4.6.3) to avoid trying to invert a singular matrix (more on this in Section 4.7). Several of the remaining assumptions are required only if one intends to do inference, e.g., to test the hypothesis $H_0 : \beta_1 = 0$. If one only intends to use the regression line to make predictions (by substituting in x values), not all of these assumptions are required. In this section, we discuss which assumptions are needed for which procedures, and what to do if an assumption is not satisfied.

The linearity assumption is needed for everything, because we are using a linear model. If it fails, we can sometimes use transformations (Section 4.6.1) until linearity is satisfied. The assumption of no autocorrelation is similarly needed for everything. Students should get into the habit of thinking about how the data was gathered, and possible sources of autocorrelation. Certain tests, such as the Durbin-Watson test (in case of dependence on time) [514], or the Moran test (in case of dependence on spatial location) [473], can be used, but should not be relied upon exclusively. There is no substitute for thinking through the relationships between data points. We will return to autocorrelation at the end of this section.

The homoscedasticity assumption is required only for confidence intervals, prediction intervals, and inference. Even if it is not satisfied, the fitted regression line $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$ provides an unbiased estimate of y values associated with a given x value. A common way to reduce heteroscedasticity is to take a log of y, or to take a log of both variables, as long as this does not break linearity [514]. An alternative approach (popular in econometrics) is to replace the standard errors by **heteroscedasticity-robust standard errors**, which are unbiased estimates of the true standard deviation of the sampling distribution. However, in this case, the convergence of the t-statistics to a normal distribution requires a much larger sample size than $n = 30$ [514]. Yet another approach, which works if the residual standard deviations σ_x are a function of x (e.g., if the scatterplot is "fan-shaped"), is to introduce a new variable and do **weighted regression**. In this approach, the dependence of σ_x on x is factored into the model, and the residuals of this more advanced model are homoskedastic, so inference works as usual [409].

The randomness assumption is required for confidence intervals, prediction intervals, and inference. If the data is not random, then it cannot be thought of as a random sample from a population. Hence, there is no sampling distribution. In this case, the regression line can still be used to make predictions, but we know predictions to be unbiased only if they are based on a random sample of data. Sometimes the data are being produced by a mathematical or physical process with little to no error (e.g., sunset times, or electromagnetic currents), and hence follow mathematical curves precisely. In such a situation, most of the residual error will be due to measurement error (Section 4.5.6), and if our measurement devices are sufficiently precise, this error could be close to zero. When this occurs, residual error and SSE will be close to zero, and so statistical tests will likely be statistically significant. This is an acceptable use of statistics, for a case when most of the data is signal, and very little is random noise. From this point of view, mathematical modeling can be thought of as a special case of statistical modeling.

In a world awash with data, it is increasingly common to gain access to data on an entire population. For instance, one might have unemployment data from each of the 50 U.S. states, or might be looking at census data (which is supposed to contain every person in the U.S.). Such data does not represent a sample, and so confidence intervals and inference should be used only if one is thinking of the population as a random sample from some "hypothetical population," possibly including future values. This should be done with caution, to avoid extrapolation. Even if inference does not make sense, regression is a powerful tool for explaining the relationship between two variables, and one can use the standard deviation of the β coefficient to measure the effect size and inherent variability in the data.

Sometimes, it is difficult to determine whether a dataset represents a random sample. If there are no clear issues of bias, then one can test if a sample exhibits statistical randomness, e.g., using the **runs test for randomness**. In this test, the null hypothesis is that the data (a sequence of numbers) is created by a random process. The test statistics is created based on the number of runs, that is, monotonic subsequences. If the number of runs is far off from what you would expect from randomly generated data (using the framework of Section 4.5.1), then we reject the null hypothesis that the data is random [409]. If we fail to reject the null hypothesis, then we have slightly more confidence that the data is random, but of course, the data could be non-random and simply behave like a random dataset from the point of view of runs. As with autocorrelation, the runs test for randomness is not a substitute for thinking carefully about the data.

The normality assumption is not required in order to use a regression model for prediction. The assumption is required for inference if the sample size is less than 30, though a t-distribution can soften this need. For a large sample size, the sampling distribution for the estimates $\hat{\beta}_1$ is asymptotically normally distributed, and hence we can do inference for the slope (or, more generally, an F-test for multivariate regression). Similarly, we can do confidence intervals for the slope and for predicted values $\hat{y}$. However, prediction intervals require normally distributed residuals even if $n > 30$. All in all, this assumption is the least important of the regression assumptions.

We conclude this section by returning to the issue of autocorrelation. If there is autocorrelation, a more sophisticated model is required, even to know that the model is useful for predicting y values. Sometimes the autocorrelation is due to time, e.g., if observations (x, y) are drawn at different moments in time, and if earlier tuples can affect later tuples. In such a situation, a time series model is appropriate. Such a model attempts to predict future values of y based on past values of y, x, and residual error ϵ. For example, to predict the value of y at time t, one model might state

$$y_t = \beta_0 + \beta_1 x_t + \beta_2 x_{t-1} + \beta_3 x_{t-2} + \beta_4 x_{t-3} + \alpha_1 \epsilon_{t-1} + \alpha_2 \epsilon_{t-2} + \gamma t + \epsilon_t.$$

Note that y_t is allowed to depend on time, t, as well as the **lagged explanatory variables** x_{t-1}, x_{t-2}, and x_{t-3}, and the **lagged errors** ϵ_{t-1} and ϵ_{t-2}. Of course, y_t can also depend on previous values of y, e.g., y_{t-1}, but if the model specification above holds for all t, then the model above already captures the dependence on y_{t-1} through the dependence on the other lagged variables.

Time series models are complicated enough even in the one-variable case, i.e., determining how x_t depends on lags x_{t-i} and on lagged errors ϵ_{t-i}. A standard course in time series analysis discusses how to decompose a time series x_t into its trend component (e.g., $\gamma t + \beta_0$), its seasonal components (predictable deviations above and below the trend), and its periodicity (longer term cycles, like the housing market). After subtracting these predictable features, one attempts to fit a model to what's left, i.e., to the residuals. If this new time series is not **stationary** (in the same mathematical sense from ergodic theory), then iterative differencing is required, i.e., shifting to a new time series of the form $\nabla x_t = x_t - x_{t-1}$. Once the time series is stationary, one fits a linear ARMA(p, q) model, where x_t depends linearly on p lags of x, and q is lags of ϵ. Even more complicated models are possible, e.g., allowing longer-term seasonal lags (such as a time series of months that depends on the previous two months, and also twelve months ago), or even models based on the Fourier transform of the time series. An excellent reference is [434]. If one has repeated measurements over time, from each individual in the sample, then the methods of **longitudinal data analysis** (also known as **panel data analysis**) may be appropriate. Such topics are worthy of an advanced course, and for a first course in statistics, it is sufficient to get students thinking about detecting autocorrelation, and being savvy enough to avoid fitting simple linear models when more complicated models are required [215].

4.6.5 Indicator variables and ANOVA

In many real-world situations, we want to study explanatory variables that are categorical rather than quantitative. For example, we might want to quantify the effect of gender upon salary, or the effect of the governing

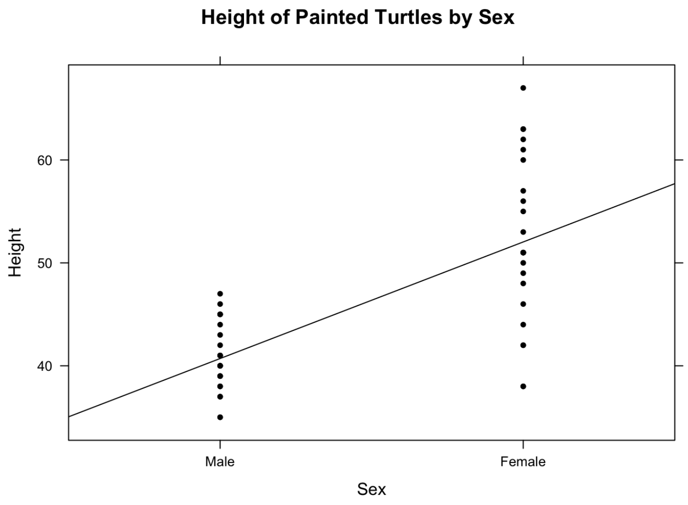


FIGURE 4.23: Because female painted turtles have a larger height on average, if we let $x = 0$ for male and $x = 1$ for female turtles, the regression line is not horizontal.

party upon the stock market. Thankfully, categorical variables can be included in multivariate regression models with ease. If a categorical variable has k possible values, then statistical software creates $k - 1$ dummy variables, also called **indicator variables**, each of which takes values 0 or 1 for every case in the data. For example, a categorical variable that takes values `male` or `female` will turn into a dummy variable `isMale` that takes the value 1 if the individual is a male, and 0 otherwise. If a categorical variable represented a traffic light, with values `green`, `yellow`, and `red`, then two categorical variables would be created: `isGreen` and `isYellow`. If the light was green, then `isGreen` will be 1. If the light was red, then both `isGreen` and `isYellow` will be 0. More generally, the category represented when all $k - 1$ variables are zero is the **reference category**, and can be set by the user of the software, or can be set automatically by the software.

In the first example above, the slope coefficient for the variable `isMale` tells us how much, on average, being male causes the salary to increase. The t-statistic and p-value for this slope coefficient tell us whether or not gender has a statistically significant effect on salary, holding the other variables in the model constant. If there are no other explanatory variables, then in the regression model `salary` $= \beta_0 + \beta_1 \cdot$ `isMale`, testing the hypothesis $H_0 : \beta_1 = 0$ (i.e., testing whether the line in Figure 4.23 is significantly different from zero) is equivalent to testing if the average salaries for men and women differ, i.e., a two-sample t-test of $H_0 : \mu_{\text{men}} = \mu_{\text{women}}$. Note that β_0 is the average salary for women (the case `isMale = 0`, and $\beta_0 + \beta_1$ is the average salary for men).

To study the (linear) impact of hair color on salary, one would study a model of the form $\texttt{salary} = \beta_0 + \beta_1 \cdot \texttt{isBlonde} + \beta_2 \cdot \texttt{isBrown} + \beta_3 \cdot \texttt{isBlack}$, where here we are assuming there are four hair colors, and the fourth is a reference category (e.g., including red hair, dyed hair, silver hair, no hair, and everything else). We interpret the β_i similarly to the previous example. The regression F-statistic and p-value tests the hypothesis $H_0 : \beta_1 = \beta_2 = \beta_3 = 0$, i.e., whether there is any difference in salary based on hair color. Now β_0 is the average salary for the reference category, and β_i is the difference in salary between the reference category and the i^{th} group. Equivalently, this tests the hypothesis $H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4$, where μ_i is the average salary in the i^{th} group (e.g., the average salary for blondes). Such tests could be studied independently of the regression framework, and often appear in statistics texts under the name **analysis of variance (ANOVA)**.

The assumptions required to carry out an ANOVA test are identical to those required for a regression model (Section 4.6.4). Note that homoscedasticity is now the hypothesis that different groups have the same standard deviation, and we have seen a test for this in Section 4.5. The ANOVA F-statistic is computed from the decomposition of variance into $SST = SSM + SSE$, as $F = MSR/MSE$ just as in Section 4.6.3. Now it is measuring the variance between groups (i.e., variance explained by the assignment of data points into groups) divided by the variance within groups (i.e., variance not explained by the grouping). The ANOVA F-test tells us the utility of a groupwise model (Section 4.3.3). Another way to phrase the hypothesis test is as a test of a groupwise model of the form $y = \mu + \alpha_i + \epsilon$, where μ is the grand mean (the overall mean value of y), and α refers to the impact of an individual being in each group (e.g., gender, or hair color). ANOVA tests the hypothesis $H_0 : \alpha_1 = \cdots = \alpha_k = 0$. Equivalently, ANOVA is testing the hypothesis that the means of each group are equal (generalizing a two-sample t-test to more than two groups).

If an ANOVA test rejects the null hypothesis, then we know that there is some pair of means $\mu_i \neq \mu_j$. We can find which pair by using two-sample t-tests, here called **post-hoc tests** because they occur after an ANOVA. Because the probability of a Type I error rises with the number of tests, a Bonferroni correction can be used to bound the probability of a Type I error below a given threshold α (often, $\alpha = 0.05$). Alternatively, one can use **Tukey's honest significant difference (HSD) test**, which has the benefit that the probability of a Type I error, for the entire collection of tests, is exactly α. The idea of Tukey's test is to compute a test statistic for each pairwise comparison (μ_i vs. μ_j) desired (using $|\overline{x_i} - \overline{x_j}|$ and the standard error from the ANOVA test), then compare these test statistics to cut-offs obtained from the ANOVA test as a whole (using a studentized range distribution, q). When there are only two means to compare, Tukey's cutoff is the same as the cutoff from a two-sample t-test. Unlike the Bonferroni correction, Tukey's method works only for comparison of means (not, say, testing multiple β coefficients in a regression).

A standard course in **categorical data analysis** would also include **two-way ANOVA**, which allows for overlapping groupwise models. For example, salary might depend on *both* gender and hair color, and every individual has both a gender and a hair color. There are now eight total groups, but instead of using one categorical variable with eight levels, we retain the grouping in terms of gender and hair color. Our model is then $y = \mu + \alpha_i + \gamma_j + \epsilon$, where α refers to the impact of gender, and γ refers to the impact of hair color. The ANOVA decomposition of squares is then $SST = SSA + SSB + SSE$, where SSA is the sum of squares explained by the gender grouping, and SSB is the sum of squares explained by the hair color grouping. There will now be two F-statistics (one based on SSA/SSE and one based on SSB/SSE, with adjustments for sample size as in Section 4.6.3), and two p-values. The first tests whether all α's are zero, and the second tests whether all γ's are zero.

Like a one-way ANOVA, a two-way ANOVA is a special case of a linear regression model of the form $\texttt{salary} = \beta_0 + \beta_1 \cdot \texttt{isMale} + \beta_2 \cdot \texttt{isBlonde} + \beta_3 \cdot \texttt{isBrown} + \beta_4 \cdot \texttt{isBlack}$, where we have implicitly replaced `hairColor` by indicator variables. The case where all indicator variables are zero corresponds to a female with hair color in the reference category (e.g., a red-headed female). The p-value for β_1 matches the F-test p-value based on SSA/SSE, and the p-value for testing the hypothesis $H_0 : \beta_2 = \beta_3 = \beta_4 = 0$ matches the F-test p-value based on SSB/SSE.

Just as in Section 4.6.3, it is important to check for possible interaction between the explanatory variables. If there is interaction, then the model would need a product variable $\texttt{isMale} \cdot \texttt{hairColor}$. Now, since the explanatory variables are categorical, instead of looking for curvature in a multivariate model, we are looking to determine if the effect of one categorical variable (say, `hairColor`) depends on the value of the other categorical variable (`isMale`). Such a determination can be made via an **interaction plot**. The idea is to see if a parallel slopes model (i.e., without an interaction term) is a better fit than a model with an extra term (which, remember, we want to include only if we need it) that allows the slopes to be non-parallel. An example is provided in Figure 4.24 below, showing the impact of running habits and gender on heart-rate. This dataset groups individuals by gender, and also by whether they are runners or sedentary. The plot shows that the lines are mostly parallel, so an interaction term is probably not required.

If a model contains both categorical and quantitative explanatory variables, most often the language of multivariate linear regression is used. However, if the quantitative variables, x_i, are not primarily of interest, sometimes they are referred to as **covariates**, and the process of fitting a linear model based on both the categorical and quantitative explanatory variables, is called **Analysis of covariance (ANCOVA)**. As the x_i are not of interest, inference for these variables is ignored. However, including covariates in the model allows for the analysis of the importance of the variables of interest, *while holding the covariates constant.* This will result in a lower SSE in the sum

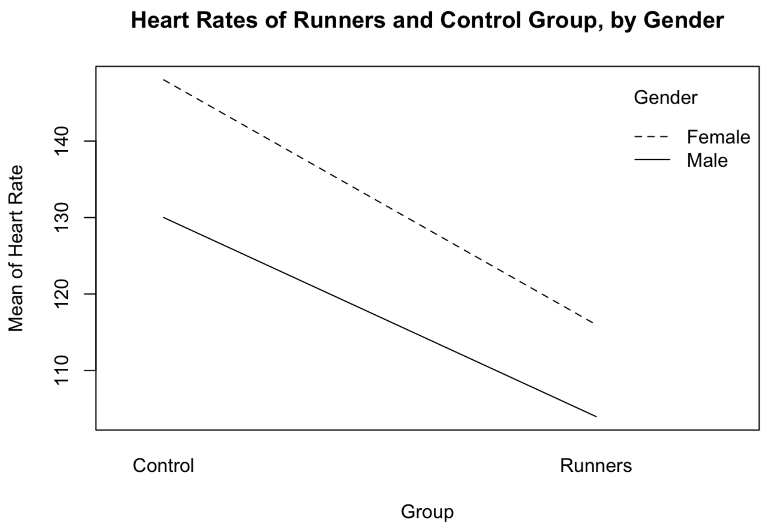


FIGURE 4.24: Interaction plot for heart rate as a function of gender and running habits.

of squares decomposition, and hence an improved F-statistic, at least, if the covariates actually explain variation.[20]

A more advanced concept, generalizing both covariates and interaction terms, is that of a **moderator** variable, i.e., a variable that influences the strength or direction of the relationship between the explanatory variable(s) of interest and the response variable. Thinking about moderator variables is also important to experimental design, and we will return to it in Section 4.8. However, a discussion of covariates and moderators could easily be omitted from a first, or even second, course in statistics. To learn more, we recommend [25].

4.7 The linear algebra approach to statistics

In this section, we simultaneously generalize both multivariate regression and ANOVA. We first generalize them to a model known as the *general linear model* (defined below), that often serves as a capstone to a second course in statistics focused on regression [100, 288]. The beauty of the general linear

[20]The use of degrees of freedom in defining MSE means that, if the covariates are useless, the F-statistic could actually be less significant.

model is that it brings linear algebra into statistics, and allows for a geometric understanding of many of the concepts introduced earlier in the chapter. This linear algebraic approach makes it easy to discuss ridge and LASSO regression (Section 4.7.2).

The next generalization involves introducing a link function to allow non-linear relationships between the explanatory variables and the response variable, and allowing non-normally distributed error terms. The simplest example of this more general setting is logistic regression, which we cover in Section 4.7.3. We then simultaneously generalize logistic regression and the linear algebra approach to linear regression, arriving at the *generalized linear model* in Section 4.7.4. This model is general enough to cover logistic regression, multinomial regression, Poisson regression, and negative binomial regression. The acronym GLM is often used for this model, but we avoid this acronym to minimize confusion between the general linear model and the generalized linear model.

A little bit more linear algebra allows us to introduce, at the end of Sections 4.7.1 and 4.7.4, even more general models known as the *general linear mixed model* and *generalized linear mixed model* (both often denoted GLMM in statistical texts), that are suitable for time series analysis, hierarchical data analysis, spatial data analysis, and longitudinal data analysis. All of these generalizations are made possible by the linear algebraic approach we are about to describe, and taken together, these general models yield almost all statistical models one comes across in practice. We conclude with a discussion of categorical data analysis in Section 4.7.5.

4.7.1 The general linear model

The **general linear model** states that $Y = XB + \epsilon$, where:

- Y is an $n \times o$ matrix of values of o different response variables (but for simplicity, we will stick to the case $o = 1$),
- X is an $n \times (k+1)$ **design matrix** whose columns correspond to explanatory variables (plus one extra column for the intercept term, β_0),
- B is a $(k+1) \times 1$ matrix containing the β coefficients to be estimated, and
- ϵ is an $n \times 1$ matrix of errors.

A concrete example is instructive. To predict salary, y, based on gender, x_1, and age, x_2, we gather data on n individuals, and store the y observations in the $n \times 1$ matrix Y. The matrix X is $n \times 3$, and the matrix B is 3×1, consisting of the unknowns β_0, β_1, and β_2. The first column of X consists of 1 in every entry. The second column of X is 1 for males and 0 otherwise. The third column of X consists of the age vector. The vector ϵ is $n \times 1$, and we can estimate it via the residuals $u_i = y_i - \hat{y}_i$, where $\hat{Y} = X\hat{B}$ and $\hat{B}$ is the vector of $\hat{\beta}$ estimates for the β coefficients. The regression assumptions are about ϵ. No autocorrelation asks that the entries, ϵ_i, are independent. Homoscedasticity asks that the standard deviations of all ϵ_i are equal. The normality assumption asks the vector ϵ to be distributed according to a multivariate normal distribution (which follows from the Central Limit Theorem, for sufficiently large sample sizes). We test these assumptions based on the vector U of residuals, but first we must fit the model.

One determines if a general linear model is an appropriate model just as one does with multivariate linear regression: by checking the regression assumptions, beginning with the assumption that the explanatory variables are related to the response linearly. This is automatic for categorical variables, because dummy variables take only two values. For quantitative explanatory variables, scatterplots reveal the relationship, and transformations should be used if necessary.

After deciding the general linear model is appropriate, a researcher must fit the model to the data. This involves finding estimates $\hat{\beta}$ for the β-coefficients, and can be done in two ways. The first involves computing the residual sum of squares as a function of the β_i ($k+1$ variables), setting partial derivatives to zero, and optimizing. The resulting equations for the β_i are called the **normal equations** (as in Sections 3.2.2.1 and 8.5.3). The second approach involves projecting the Y vector onto the k-dimensional hyperplane spanned by the x_i. The projected vector is $\hat{Y}$, and represents everything the X variables know about Y. Just as in linear algebra, there is a **projection matrix**, P, and $\hat{Y} = PY$. To do ordinary least squares regression, we must choose the vector $\hat{B}$ to minimize the residual sum of squares, i.e., the dot product $(Y - XB)^T(Y - XB)$. This dot product is the squared Euclidean distance, $||Y - XB||^2$, from the vector Y to the hyperplane XB. That distance is minimized by $\hat{Y} = X\hat{B}$, the orthogonal projection. Orthogonality guarantees that $X^T(Y - XB) = 0$.

The normal equations are now equivalent to the statement that $(X^TX)\hat{B} = X^TY$. It follows that $\hat{B} = (X^TX)^{-1}X^TY$, as long as X^TX is invertible. This is why the explanatory variables are required to be linearly independent, and why statistical software outputs an error when they are not. Since $\hat{Y} = X\hat{B}$, it follows that $P = X(X^TX)^{-1}X^T$. This matrix P is often called the **hat matrix**, since multiplication by P puts a hat on the vector Y. The estimates $\hat{\beta}$ obtained in this way match the estimates obtained via multivariable calculus. The Gauss-Markov Theorem states that $\hat{B}$ is the best linear, unbiased estimator of B.[21] The proof requires the regression assumptions, but not multivariate normality of the errors.

Already in the case of one explanatory variable, this approach sheds light on classical regression. In this context, x and y may be viewed as $n \times 1$ vectors, and we normalize them to vectors a and b obtained by $a_i = \frac{x_i - \overline{x}}{\sigma_x}$ and $b_i = \frac{y_i - \overline{y}}{\sigma_y}$. These new vectors have mean zero and standard deviation one, but point in the same directions as the original vectors. Let θ denote the angle between

[21]Here "best" means minimal variance among all linear, unbiased estimators.

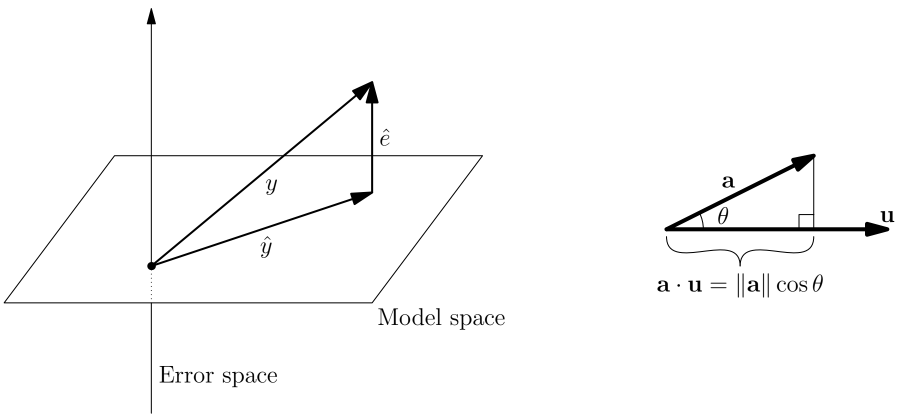


FIGURE 4.25: The linear algebra approach to regression: orthogonal projection and correlation as a dot product.

them. As students learn in a first course in linear algebra, the projection from b onto a is given by the formula $\frac{a \cdot b}{||a||}$. Elementary trigonometry then tells us that $\cos(\theta) = \frac{a \cdot b}{||a||||b||}$, as shown in Figure 4.25. Replacing a and b with their definitions in terms of x and y, one discovers that $\cos(\theta) = r$, the correlation between x and y. This provides students another reason that correlation is always between -1 and 1. We find it helpful to use the slogan **correlation is a dot product**. Figure 4.25 also demonstrates what regression is really doing. The model space is spanned by the explanatory variables x_i. The n-dimensional vector of fitted values $\hat{y}$ is the closest vector to y that can be obtained in the model space, i.e., is the orthogonal projection of y onto the model space. The n-dimensional residual vector $\hat{e}$ (which estimates the unknown error vector ϵ) is the orthogonal distance from y to the hyperplane spanned by the explanatory variables x_i.

In the general context of k explanatory variables, the **covariance matrix**, Σ, is defined to be $E[(X-\mu_X)(X-\mu_X)^T]$. The entry $\Sigma_{i,j} = E[(X_i-\mu_{X_i})(X_j-\mu_{X_j})^T]$ is the **covariance** of the variables X_i and X_j. The diagonal entries, $\Sigma_{i,i}$, represent the variances of each explanatory variable. The **correlation matrix** is a scaled version of Σ, with ones along the diagonal. As above, we obtain that each correlation is the cosine of the angle between the vectors $Y - \overline{Y}$ and $X_i - \overline{X_i}$.[22] Furthermore, the coefficient of determination, R^2, is $\cos^2(\theta)$, where θ is the angle between $Y - \overline{Y}$ and $\hat{Y} - \overline{Y}$.[23] Furthermore, the F-statistic (or nested F-statistic) can be obtained as the cotangent squared of the angle between $\hat{Y}_{\text{Full}} - \overline{Y}$ and $\hat{Y}_{\text{Reduced}} - \overline{Y}$ [100]. This is because the

[22]The context makes it clear here that $\overline{Y}$ represents an $n \times 1$ vector, where each entry is $\overline{y}$.

[23]It is an exercise to show that the mean of the vector $\hat{Y}$ equals the mean, $\overline{Y}$, of the vector Y.

fundamental equation of ANOVA, that $SST = SSM + SSE$, can be written as $||Y - \overline{Y}||^2 = ||\hat{Y} - \overline{Y}||^2 + ||Y - \hat{Y}||^2$, i.e., as the Pythagorean Theorem.

The linear algebra approach also allows us to write down equations for the standard errors of the $\hat{\beta}$, and hence to make confidence intervals and carry out inference. The residual vector, U, may be written as $Y - \hat{Y} = (I - P)Y$. Hence, multiplication by $I - P$ converts the response vector, Y, to the residual vector, U. The variance, σ_ϵ^2, of the unknown errors ϵ, is estimated by $s_\epsilon^2 = \frac{U^T U}{n-k}$, where the denominator is required in order to make this an unbiased estimator. The covariance of the residuals is obtained as $\Sigma_u = (I - P)^T \Sigma (I - P)$. If we have perfect homoscedasticity and no autocorrelation at all, then Σ_u is the diagonal matrix $\sigma^2 I_n$, where σ^2 is the variance of every ϵ_i.[24]

Assuming $\Sigma_u = \sigma^2 I_n$, elementary matrix algebra implies $E[(\hat{B} - B)(\hat{B} - B)^T] = \sigma^2 (X^T X)^{-1}$. One then obtains the standard error, $SE_{\hat{\beta}_i}$ of the ith estimated coefficient, as the (i, i) entry of this matrix. If the errors are distributed according to a multivariate normal distribution, $\epsilon \sim N(0, \sigma^2 I)$, then $\hat{\beta} \sim N(\beta, \sigma^2 (X^T X)^{-1})$. One uses this to make confidence intervals, and to compute t-statistics, p-values, and F-statistics just as in Section 4.6.3. Note that we do not know σ^2 in practice, but it is standard to estimate it by $\hat{\sigma}^2 = \frac{n-k}{n} s^2$, using s^2 from the previous paragraph [288].

Another powerful consequence of the linear-algebra approach to regression is to better understand leverage and influential points. First, one may compute **standardized residuals** as $(y - \hat{y})/\hat{\sigma}_\epsilon$, to see how extreme a residual is, relative to the underlying variation. Next, the **leverage** of the ith data point on the model, is $h_i = p_{i,i}$, the ith diagonal entry of the hat matrix, P. For simplicity, we stick to the case of one explanatory variable, but everything we say can be generalized to k explanatory variables. Unpacking the formula $h_i = p_{i,i}$ yields a formula relating the leverage to the geometric distance between x_i and $\overline{x}$, hence the term leverage, viewing $\overline{x}$ as the fulcrum. The ability to compute the leverage of each data point, via the hat matrix, allows us to compute the standard deviation of the leverages. If a data point has a leverage that is two or three standard deviations away from the average leverage, then that might be cause to consider that data point separately, as discussed in Section 4.6.2. This can even be formalized into a t-test, to test whether the ith point has statistically significant leverage [101].

More important than leverage is **influence**, which is leverage times discrepancy (where discrepancy is how far x_i is from the regression line). This is analogous to torque, from physics, as can be seen on the left of Figure 4.21. **Cook's distance** is one way to measure influence, and is computed by

[24] In some sources, this assumption will be phrased as saying the **errors are independent and identically distributed (iid)**. Technically, to be iid, one would also need to know that all the errors have the same distribution, e.g., that all are normal. That they all have mean zero follows from the linearity assumption, and the residuals have mean zero because we fit the model via orthogonal projection.

standard statistical software:

$$D_i = \frac{(\hat{\epsilon_i}^*)^2}{k+1} \cdot \frac{h_i}{1-h_i}.$$

The * indicates a studentized residual, that is, the i^{th} residual computed from a model fit without data point i. Therefore the formula above calculates how far the predicted values $\hat{y}$ would move if your model were fit without the point (x_i, y_i). If D_i is larger than 1, the point (x_i, y_i) is very influential. If $0.5 < D_i < 1$, then the point (x_i, y_i) is somewhat influential. These designations can be used to help a modeler decide whether or not to further investigate (x_i, y_i), e.g., to determine if this point is from a different population than the rest of the points [101].

In this section, we have sketched a linear algebra-based approach to regression, that often forms the backbone of a second course in statistics [100, 288]. The beauty of this approach is in its elegant reliance on geometry. The power is in its wide range of applications and ease of generalizability. In addition to providing multivariate regression and ANOVA as a special case, this approach also gives MANOVA and MANCOVA, for when there is more than one response variable (e.g., predicting both salary and life expectancy). The approach of this section may also be made to apply to **hierarchical modeling**, which is useful when data comes naturally grouped into hierarchies that help explain variation (e.g., students grouped into schools, and schools grouped into school districts), or when building **multi-level models** (with different β's at different levels, or using one model to fill in missing data, and then feeding that into a subsequent model). For details, see [402]. As for generalizability, we will see in the next sections how the approach in this section may be modified to yield almost all statistical models one comes across in practice.

Dropping the assumption of homoscedasticity, but retaining the assumption of no autocorrelation, and letting σ_i^2 denote the variance of ϵ_i, we have

$$\Sigma_u = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n^2 \end{bmatrix}.$$

The OLS standard errors will now be wrong, i.e., $X^T \Sigma_u X \neq \sigma^2 X^T X$. To correct this, we can either do **weighted regression**, where we try to model Σ_u based on X, or we can use **robust standard errors** (a.k.a. **heteroscedasticity consistent standard errors**), which keep the same estimates, $\hat{\beta}_i$, but compute the standard errors $SE_{\hat{\beta}_i}$ using UU^T in the formula for $E[(\hat{B}-B)(\hat{B}-B)^T]$, rather than using $\sigma^2 I$. It is then a theorem, due to White and building on earlier work of Huber and Eicker, that the resulting standard errors will yield confidence intervals and p-values that are correct asymptotically, i.e., as $n \to \infty$ [514]. Weakening the "no autocorrelation" assumption slightly allows for off-diagonal entries of Σ_u, and hence for off-diagonal entries of $E[(\hat{B} - B)(\hat{B} - B)^T]$. Such entries give a formal definition of **variance inflation factors (VIFs)** from Section 4.6.3 [101].

If one drops the "no autocorrelation" assumption entirely, one must consider more complicated structure in the matrix Σ_u. This generalization necessitates the consideration of stochastic processes, i.e., the consideration of both fixed effects and random effects (while the general linear model allows only fixed effects). Following this path leads to the **general linear mixed model**, which builds on the idea of hierarchical models, and is often appropriate for panel data [215]. Along the same line of generalization, one finds ARIMA models, as described in Section 4.6.4. Note that the general linear model assumes linear relationships and a normal error term.

4.7.2 Ridge regression and penalized regression

In the previous section, we saw how the linear algebra approach to ordinary least-squares regression finds coefficients $\hat{\beta}$ that minimize the residual sum of squares. In this section, we briefly discuss alternative models that can do better in certain situations. Full details of the contents of this section may be found in [213, 247, 284]. The content of this section would rarely make it into a first or second course in statistics, but may be useful to the reader.

Ridge regression, also known as **Tikhonov regularization** or ℓ_2 **regularization**, is an alternative way to fit the $\hat{\beta}$ coefficients that reduces the variance (i.e., reduces $SE_{\hat{\beta}_i}$), but at the cost of introducing slight bias into the estimates. This phenomenon is representative of the **bias-variance trade-off** in machine learning. This trade-off cannot be avoided, as the Gauss-Markov Theorem shows. Ridge regression is a powerful tool when building a model where explanatory variables are tightly correlated, e.g., a model with large VIFs. The process of fitting the $\hat{\beta}_i$ begins by standardizing the explanatory variables, so that $R = X^T X$ is the correlation matrix. One then solves a constrained optimization problem, using Lagrange multipliers, rather than the classic OLS minimization of residual squared error.

The idea is to add a small parameter, k, to the diagonal entries of the matrix R (this diagonal is a "ridge"), then define $\hat{B}^{\text{ridge}} = (R + kI)^{-1} X^T Y$, which solves the constrained optimization problem. The resulting coefficients, $\hat{\beta}^{\text{ridge}}$ have less variance than $\hat{\beta}$ but are slightly biased, depending on k. They are also more stable, in the sense that small perturbations of the data lead to less severe changes in $\hat{\beta}^{\text{ridge}}$ than in $\hat{\beta}$. There is an optimal choice of k (which can lead to improved prediction error relative to OLS regression), but knowing it would require knowing the true regression coefficients, β. A **ridge trace** can be used to help the researcher decide on a value of k, and the methods of machine learning should be used to avoid overfitting the model to the data. The value of k is a Lagrange multiplier penalizing β-estimates that are too large in ℓ_2-norm,

$$\sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2 + k \sum_{i=1}^{n} \beta_j^2.$$

As in the previous section, this cost function can be rephrased in terms of dot products, to obtain a more geometric intuition for ridge regression. This approach reappears when studying **support vector machines** in machine learning (Section 8.8), where one transforms a nonlinear separation problem into a linear separation problem by using **kernel functions** (Section 8.8.3) to shift the problem into a larger dimensional linear space. In that context, the regularization term (Lagrange multiplier) is to guarantee a solution exists.

LASSO regression (Least Absolute Shrinkage Selector Operator) conducts a procedure analogous to the above, but for the ℓ_1-norm,

$$\sum_{i=1}^{n}(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2 + k\sum_{i=1}^{n}|\beta_j|.$$

Again, a tuning parameter, k, determines how much we penalize solutions that have a large ℓ_1-norm. However, unlike ridge regression, LASSO regression can set certain coefficient estimates to zero. Consequently, it can be used for variable selection (i.e., if the estimate for β_i is zero, that is equivalent to leaving x_i out of the model). The downside of this is that there is now no closed-form expression, and no uniqueness result, for the new estimated coefficients $\hat{\beta}^{\text{lasso}}$. Numerical optimization techniques, such as gradient descent (covered in Chapters 6, 8, and 9), must be used. Further generalizations are possible, such as elastic net regression, which simultaneously penalizes both ℓ_1 and ℓ_2 norms.

4.7.3 Logistic regression

Classically, linear regression is used when both the explanatory and response variables are quantitative. If the explanatory variables are categorical, then we are doing ANOVA, but the generalized linear model shows that the procedure works the same whether the explanatory variables are quantitative, categorical, or both. In this section, we will discuss what to do when the *response variable* is categorical. We provide R code later in Figure 4.27. We focus first on the simplest kind of categorical response: a binary (yes/no) response. For example, in the Whickham dataset (which comes with the R package `mosaic`), we have data on individuals including their age and whether or not they smoked when data was collected in 1974,[25] and we know whether or not they were alive 20 years later in a follow-up study. In this case, the response variable is whether or not they were alive.

In this setting, it is not appropriate to use a model of the form $Y = XB+\epsilon$, where ϵ is normally distributed. Indeed, if the response variable takes values in $\{0,1\}$, then the error term should be distributed according to a Bernoulli distribution (i.e., a binomial distribution with only one trial). Fundamentally, what we are interested in is the probability that the person survived, based

[25]Technically, this data was collected in 1972–1974, but we will simplify for the sake of exposition.

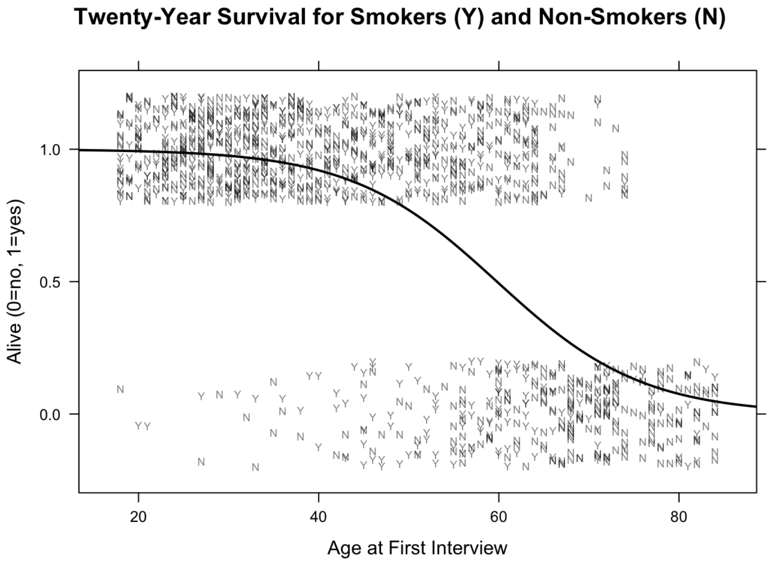


FIGURE 4.26: A logistic model fit to the smoking survival rate described in the text, here using only the age variable, not the smoking variable.

on the values of the explanatory variables, X. A standard linear model is not appropriate, because it could lead to predicted probabilities smaller than 0 or larger than 1. The general solution to this kind of situation is the **generalized linear model**, but for ease of exposition, we work it out first for the case of a simple logistic regression where the response variable is p, the probability that an individual survived, and the explanatory variable is x, their age in 1974. (Obviously, the smoking variable is the more interesting one, but we focus on age first and add the second variable in the next section.) The **logistic regression curve** is shown in Figure 4.26.

This graph demonstrates that older people were less likely to survive till the follow-up study in 1994. Note that the response variable, p, is always between 0 and 1, as it should be. To fit the logistic model, we first transform the response variable to $y = \log(p/(1-p))$. This transformation is called the **logit function**, also known as the **log-odds** function, because the odds associated with a probability of p are $p/(1-p)$.[26] The logit function maps $(0, 1)$ bijectively and continuously to $\mathbb{R}$. We next fit a simple linear regression model of the form $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$. Lastly, we transform back using the inverse of the logit function:

$$\hat{p} = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x}}.$$

[26]For example, if $p = 0.75$, then $p/(1-p) = 3$ and the odds are 3-to-1.

```
logm = glm(sex ~ height, data=Galton, family=binomial(logit))
summary(logm)
exp(confint(logm))
```

FIGURE 4.27: R code to carry out a logistic regression.

This is the equation for the logistic model, which predicts p given x. The curve slopes downwards if $\beta_1 < 0$ and slopes upwards if $\beta_1 > 0$. In the former case, a larger β_0 means the curve slopes upward at a lower x value. In the latter case, a larger β_0 means the curve slopes upward at a higher x value. The larger the value of $|\beta_1|$, the steeper the curve.

In order for this model to be appropriate, we must know that y is a linear function of x. This can be checked with a scatterplot. If x were binary (like the other explanatory variable, `smokerYes`, in the Whickham dataset), then linearity would be automatic, as we will see in the next section. We also need to know that the data is random and that different data points are independent.

As with classical linear regression, β_1 represents the average change in y for a unit change in x (e.g., the impact of being one year older). However, the response variable we are interested in is p, not y. We interpret β_1 in terms of p via the **odds ratio**, which tells us how the odds of survival would change (on average) if age were increased by one. For example, if $\beta_1 = -0.12368$, then the model tells us that a change in age by one year (from A to $A+1$ say) leads to an odds ratio $odds_{A+1}/odds_A = e^{-0.12368}$. The mantra is "change in odds equals e to the slope" [101], where "change" means "multiplicative change." We interpret the odds ratio as follows. The odds of survival for a person aged $A+1$ are the odds for a person aged A, multiplied by $e^{-0.12368} \approx 0.88$. Hence, the odds of survival go down, as we would expect. If the expected probability of being alive at age A was p, so that $O = \text{odds}_A = p/(1-p)$ and $p = O/(O+1)$, then for age $A+1$ the probability becomes $p' = 0.88O/(0.88O+1)$. This causes $p' < p$, as one would expect, but of course the impact of age on p is not linear, as the logistic curve demonstrates.

The R code in Figure 4.27 carries out the logistic regression procedure described above, again on the Galton dataset. Here our explanatory variable is `height` and our response variable is `gender`. Even though `gender` comes to us as a sequences of characters M and F, R is able to automatically convert this into a binary numeric variable. The `summary` command produces the logistic regression table, which we display in Table 4.5. The command `exp(x)` applies the function e^x, and we use this to find a confidence interval for the odds ratio. The code returns a confidence interval of $(2.016, 2.464)$, centered on $e^{\hat{\beta}_1} = 2.218$.

Having discussed how to fit the model and how to interpret the model, we turn to how to assess the significance of the model. We wish to test the hypothesis $H_0 : \beta_1 = 0$, to determine if age has a statistically significant effect on the probability of surviving. The test statistic is $z = \hat{\beta}_1/SE_{\hat{\beta}_1}$ and is called the **Wald statistic**. The theory of maximum likelihood estimation implies z is normally distributed, and hence we can use a normal table (or software) to find the p-value to assess our model's significance. Note that, unlike simple linear regression, we do not need the t-distribution or degrees of freedom. This is because we don't need to estimate the standard deviation, because it can be derived from the mean, using the familiar formula that the variance of a binomial(n, p) distribution is $np(1 - p)$. The key point is that the errors are Bernoulli distributed [101].

If we have multiple explanatory variables, then to assess model significance, we need a test analogous to the ANOVA F-test. There, we compared the best-fitting horizontal hyperplane (all slopes being zero), whose residual sum of squares was SST, with the best-fitting hyperplane, whose residual sum of squares was SSE. The same idea works for logistic regression, but instead of comparing sums of squares, we compare the two models by comparing the **drop in deviance** in shifting from the null model to the full model.[27] Our presentation follows [9]. The **deviance** of a model is defined as the sum $\sum d_i$ (from $i = 1$ to n) where d_i is the deviance of the i^{th} residual, that is, how far the model is off for the i^{th} data point (for a formula for d_i, see [9]). Analogous to an ANOVA table, we may obtain an **analysis of deviance** table, keeping track of the deviance explained by each variable (just like the sum of squares decomposition).

Equivalently, the deviance for a model can be defined as $-2(L_m - L_T)$, where L_m is the maximum of the log-likelihood function for the model (that is, based on the fitted coefficients $\hat{\beta}$), and L_T is the maximum possible log-likelihood (achieved by the **saturated model**, which has a separate parameter for each data point, and hence no residual error). The **null model** has only an intercept term (all slopes are zero). Both the null model and the full model have a deviance. The resulting test statistic,

$$G = \text{Null deviance} - \text{Residual deviance},$$

is χ^2-distributed for large sample sizes n, and its p-value is a standard output of statistical software. The relationship between G and the Wald statistics of the individual variables is precisely analogous to the relationship between the F-statistic and the individual t-statistics. The odds ratio e^G is called the **likelihood ratio**. Doing the drop in deviance test when there is only one explanatory variable recovers a test known as the **likelihood ratio test**, an improvement on the Wald test. In general, the likelihood ratio is the likelihood of seeing the data (if the null hypothesis were true) divided by the maximum likelihood (allowing parameters to range over the whole parameter space, rather than just the values allowed by the null hypothesis).

A reader who has studied entropy or information theory may wonder if there is a connection to the appearance of $-2 \log L$ above. Such a connection

[27]The same procedure works in comparing a reduced model to the full model, just like a nested F-test for regression. Simply replace "Null" by "Reduced" in the discussion.

TABLE 4.5: Sample logistic regression table.

```
Call:
glm(formula = sex ~ height, family = binomial(logit), data = Galton)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5425  -0.5068   0.0789   0.5046   3.2188

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -52.98422    3.40240  -15.57   <2e-16 ***
height        0.79683    0.05117   15.57   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1243.75  on 897  degrees of freedom
Residual deviance:  626.22  on 896  degrees of freedom
AIC: 630.22
```

```
fmod <- glm(sex ~ height, data=Galton,family = "binomial")
nmod <- glm(sex ~ 1, data=Galton, family = 'binomial')
anova(nmod, fmod, test = 'Chisq')
```

FIGURE 4.28: R code for analysis of deviance table.

indeed exists, and is used to define the **Akaike information critrerion (AIC)**, which is the logistic regression version of adjusted R^2 (and hence commonly appears in logistic regression tables). AIC is defined as

$$AIC = -2(\text{maximum log likelihood} - \text{number of model parameters}).$$

Observe that AIC penalizes models for having unnecessary parameters, just like adjusted R^2 does. For details, see [101]. All of the quantities just discussed appear in the logistic regression table produced by the code in Figure 4.27, and displayed in Table 4.5.

The table demonstrates that `height` is a statistically significant predictor for `gender`. The table also displays the AIC, the deviance of the null model, the deviance of the residual model, and (via subtraction) the drop in deviance. The analysis of deviance table, including the G-statistic and p-value, can be obtained via the code in Figure 4.28. This code fits the full model `fmod`, then fits the null model `nmod`, then uses the `anova` command to get the analysis of deviance table.

4.7.4 The generalized linear model

Just as linear regression can be extended to the general linear model via a linear algebra approach, so too can logistic regression be extended to the **generalized linear model**. This model states that $g(Y) = XB + \epsilon$, where ϵ is some random variable (not necessarily normal) with expectation $E[\epsilon] = 0$, and the **link function** g allows for nonlinear relationships between the explanatory variables and the response. The general linear model is a special case, where g is the identity and ϵ is normally distributed. In the case of logistic regression, $g(p) = \log(p/(1 - p))$, and the errors, ϵ, are Bernoulli distributed. The general procedure is to first transform Y to $g(Y)$, then find the coefficients $\hat{\beta}$, often using maximum likelihood estimation or a linear algebra approach, and then conduct a drop in deviance test to determine if the model is significant. For details, we refer the reader to [288].

If we wished to study a categorical response variable with more than two categories (for example, to predict one of five disease types based on explanatory medical data), then we should do **multinomial logistic regression** (also known as **softmax regression**). In this case, we can use the same link function (because we are trying to predict probabilities), but now the errors will be distributed according to a multinomial distribution (like a binomial, but with more options than just "success" or "failure").

A related type of regression is if our response variable represents count data, e.g., for a female crab, taking y to be the number of satellites (males residing nearby) she has. In this case, the link function will be log rather than log(odds), because the response is not a probability, but takes only non-negative values. The log function maps $\mathbb{R}_{>0}$ to $\mathbb{R}$, and we can fit a simple linear regression to $g(Y)$ without fear of predicting impossible values. We transform back to count data via the exponential function. The errors, ϵ, should now be distributed according to the Poisson distribution, which is appropriate for count data. Consequently, this form of regression is known as **Poisson regression**. Note that the Poisson distribution has the property that its variance is equal to its mean. If this is not supported by the distribution of residuals, a quasi-Poisson model can be fit (which introduces an extra parameter to allow the variance and mean to be different), or even a negative binomial model for ϵ. In this case, one is doing **negative binomial regression**.

Just as one can extend the general linear model to the general linear mixed model, by allowing both fixed and random effects (i.e., choosing different structure for the matrices Σ_u in Section 4.7.1), so can one extend the generalized linear model to a **generalized linear mixed model**. Equivalently, the generalized linear mixed model is just the general linear mixed model plus a link function g and a non-normal error term ϵ. Such models arise in longitudinal data analysis when the response variable is binary, categorical, or represents count data [215].

4.7.5 Categorical data analysis

In the previous section, we saw how to predict a categorical response variable from a quantitative explanatory variable. Of course, it is not terribly surprising that age is a strong predictor for failure to survive. A more interesting model would be to predict survival in the Whickham dataset based on the variable `smokerYes`. In this case, we have both a binary predictor and a binary response. We can do logistic regression and obtain a Wald statistic and p-value for whether or not the model is statistically significant,[28] and it turns out the p-value will be exactly the same as we would obtain from the hypothesis test of Section 4.5, to determine if the probability of survival is the same for smokers and non-smokers, $H_0 : p_{\texttt{smokerYes}} = p_{\texttt{smokerNo}}$.

Now suppose the explanatory variable has more than two levels. For example, if we wish to predict whether or not someone has a job based on one of four possible hair colors, then the test statistic G, and its associated p-value, are testing the hypothesis that hair color does not have a significant impact on the probability of having a job. This is equivalent to a classical **χ^2-test for association**, whose test statistic is the sum over all cells (in this example, there are $2 \times 4 = 8$) of

$$\frac{(\text{Observed} - \text{Expected})^2}{\text{Expected}}.$$

If the counts in all 8 cells are the same, then the test statistic is zero. If there really is an association between hair color and whether or not someone has a job, then some cells will have a much larger fraction of success than others. If both the explanatory and response variables have more than two levels, then the test statistic from the generalized linear model can be viewed as conducting a **χ^2-test for homogeneity**, which again sums

$$\frac{(\text{Observed} - \text{Expected})^2}{\text{Expected}}$$

over all cells.

It is possible to teach an entire course on categorical data analysis [9]. For a first-year sequence in statistics, it is probably sufficient if students see a χ^2-test for association, and it is a bonus if they realize the connection to logistic regression.

[28]Recall from the previous section that we do not need to check that a linear model is appropriate, because our explanatory variable has only two values, so is linear automatically.

4.8 Causality

A large fraction of questions begin with "why." It is human nature to wonder why something occurs and whether it is caused by something else. Statistics can be used to answer such questions, but the uncertainty inherent in the field makes it difficult. Until now in the chapter, we have seen how to tell when two variables are correlated, how to use R^2 to quantify how much uncertainty in the response variable is explained by the explanatory variables, and whether or not explanatory variables are statistically significantly better than nothing when it comes to modeling the response. None of this tells us if a change in an explanatory variable *causes* a change in the response, and correlation can be very misleading. For example, if one studies a dataset of children, one might discover a strong correlation between shoe size and score on a reasoning exam. Neither of these variables causes the other, but both are caused by a lurking variable, age. Spurious correlations are not even rare, but are often hilarious, as [487] demonstrates. Causation is also easy to get backwards, e.g., the phrase "rain follows the plow" from the late 1800s, expressing the belief that rainfall was caused by human farming [256]. In this section, we will briefly discuss how statistics can make causal inferences, and avoid these fallacies. Our treatment is inspired by [256].

4.8.1 Experimental design

The gold standard for an experiment to demonstrate causality is the **randomized controlled trial**. To carry out such an experiment, a researcher has to consider all variables that might cause y, then collect a simple random sample of data where all of those variables have been controlled for, except for the explanatory variable of interest, x. Control here refers to the experimenter accounting for these variables in such a way that it can be known that they are not causing y. For example, to determine if a particular drug helps patients sleep, one might choose y to be the number of hours of sleep, and x to be 1 for patients taking the drug, and 0 for patients not taking the drug.[29] Since age also has to do with how well a person sleeps, the experimenter could start with a population where everyone is the same age, or could do stratified random sampling to be sure that both the treatment group (where $x = 1$) and the control group (where $x = 0$) have the same distribution of ages. That way, if the treatment group gets more sleep, we know it can't be because they are younger while the control group is older. Similarly, we could control for gender by making the two groups gender balanced, or we could control for living circumstance in the same way.

[29] Actually, patients not taking the drug should still be given a placebo, and should not know which group they are in, to control for the placebo effect.

Sometimes, it is impossible or unethical to control for every variable or to randomly assign individuals to the treatment or control group. For example, to construct an airtight statistical argument that smoking causes lung cancer, we would need to assign individuals to degrees of smoking. To argue that a rise in greenhouse gases causes global warming, we would need to assign atmospheric greenhouse gas levels (and, since we have only one planet, a control group would be impossible). In some situations (such as social science), there may be an extremely large number of variables to control for. Different researchers could think up different variables that might matter, and for this reason it is unlikely any one statistical experiment will be taken as definitive proof of causation. It is essential for students to understand that **correlation is not causation**, that controls are needed to argue in favor of causation, and that different researchers can disagree, based on the same data. Students should also know about placebo treatments, and about the importance of blindness in experiments. An experiment is **double-blind** if neither the subjects nor those they interact with know whether or not an individual is receiving the treatment or a placebo. This information is known only to the statistician analyzing the data, to control for the possibility that such knowledge would cause different interactions between experimenters and subjects (giving a possible alternative reason for differing outcomes, and impacting the ability to draw a causal inference).

There is much more that can be said about causation, and may be appropriate for a second course in statistics [54], but we will content ourselves with a very brief survey, and we refer the reader to the textbook [123, 256] for more details.

First, to conduct a causal statistical experiment, an experimenter must create a **hypothetical causal network**, i.e., a directed graph whose nodes are variables and where a directed edge denotes a causal relationship. An example is provided in Figure 4.29, and many more examples may be found in [256]. Links are drawn as dotted lines if there is a non-causal relationship. Nodes can be marked (e.g., written in different shapes) to denote that certain variables were not measured. (These are called **latent variables**, and are often impossible to measure, like quality of life.) Sometimes, an experimenter can use a **proxy variable** to stand in for a latent variable, e.g., using wealth to stand in for quality of life. To be useful, a proxy variable must be closely correlated with the variable of interest. For example, the widths of tree rings, and information from ice cores, are common proxy variables for historical environmental conditions (on which we can't gather data because they happened in the distant past).

In the hypothetical causal network, a **correlating pathway** between variables x and y is a path that includes a source (i.e., a variable with in-degree zero). Such a path does not have to start at either x or y, and does not even need arrows pointing to x or y. It should be understood as an undirected path. A **non-correlating pathway** is any other undirected path. See Figure 4.29 for examples of both types of paths. To prove that the treatment variable, x, causes change in the response variable, y, we must **block** all correlating

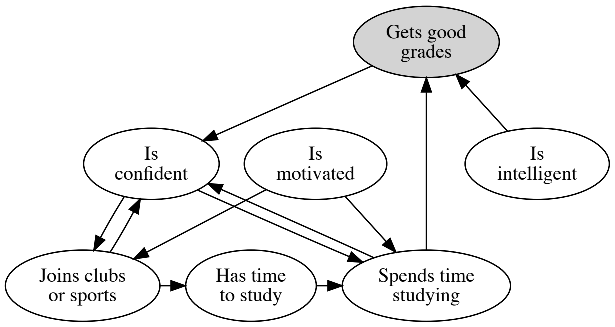


FIGURE 4.29: A hypothetical causal network of some factors in the lives of high-school students, where perhaps the variable of interest is "Gets good grades."

pathways except for the one we intend to argue proves x causes y. To block a pathway, we must include some variable in that path as a covariate in the model, so that its influence can be held constant. In a simple linear regression $y = \beta_0 + \beta_1 x + \epsilon$, controlling for a covariate, z, means including it in the model, i.e., using the model $y = \beta_0 + \beta_1 x + \beta_2 z + \epsilon$. In this second model, the coefficient β_1 describes the relationship of x on y while *holding z constant.*

If a variable is outside of our model, but still affects our variables of interest (or the relationship between them), it is called an **exogenous variable**. Such a variable E can be included when we draw a hypothetical causal network (Figure 4.29), but should be denoted in such a way that it is clear we do not have data on E. If we cannot set up an experiment to avoid an exogenous variable, then we should find a proxy variable for it that we can include in our model. Variables with at least one arrow pointing to them are called **endogenous**, and they can create relationships between x and the error term, ϵ. Again, it can be valuable when drawing a hypothetical causal network (Figure 4.29) to denote such variables with a new symbol.

One way to correct for the presence of endogenous variables is via the use of **instrumental variables**. These are variables that are correlated with the treatment variable but orthogonal to all the covariates. By projecting both the treatment and response onto an instrumental variable, one avoids the need to include the covariates in the model, and eliminates the impact of endogenous variables. Equivalently, one can do a **two-stage least-squares regression**, where the first stage predicts values for endogenous variables based on exogenous variables, and then the second regression uses these predicted values as inputs. In this way, everything in the new model is based on exogenous, rather than endogenous, variables. If time is a variable, but if endogenous variables are fixed in time, then a **difference in differences** approach can be used to remove the effect of endogenous variables.

There are even more special variable names in a hypothetical causal network. In a correlating pathway of the form $X \leftarrow T \rightarrow Y$ (where T is the source), then T is called a **confounder**, just like how age confounded the relationship between foot size and test score. In a non-correlating pathway of the form $X \rightarrow T \leftarrow Y$, then T is called a **collider**. One should not adjust for such a variable. Doing so inadvertently builds Y into the explanatory variables. For example, both depression and risk-taking cause drug use, and one should not include drug use as a variable when exploring the relationship between depression and risk-taking. In a correlating pathway of the form $X \rightarrow T \rightarrow Y$ (where X is the source), we call T a **mediator variable**. For example, in many people, age causes weight gain, and weight gain causes knee pain. Age also causes knee pain. In this case, weight gain is a possible variable that mediates the relationship between age and knee pain. Other variables matter, too, such as changes in how the body heals. A **moderator variable** affects the direction and/or strength of a relationship. An example is an interaction term in ANOVA. Mediator variables *explain* part of the relationship, whereas moderator variables *affect* the relationship.

Even when it is possible to measure all variables (or find suitable proxy variables), it is still challenging to come up with a study design that controls for the covariates. Certainly individuals must be assigned randomly to either the treatment or control group, but an entirely random assignment gives away control over the covariates. Often, a **randomized block design** is used, where individuals are divided into blocks in such a way as to reduce the variability between blocks (e.g., a block could consist of individuals of the same age, gender, and race), then within each block, individuals are randomly assigned to receive either the treatment or the placebo. For more on design, see [256].

4.8.2 Quasi-experiments

We conclude this section with a discussion of what can be done with data that is not obtained by a random experiment. One approach is to use a **quasi-experiment**, which is like an experiment, but where the assignment to treatment and control groups is not random [343]. In such a setting, it is important to argue that the explanatory variable *behaves* randomly with respect to all other variables in the system. One approach is to build a regression featuring the explanatory variable and the covariates. Additionally, one needs a theoretical justification for why no other variables could have been involved in explaining the causation the hypothetical causal network seeks to explain. For example, an extensive literature in economics studies the causal relationship between weather variables and the economy, using the fact that the weather is random, and hence can only be randomly related to the economy [125]. A similar approach is a **knockout experiment**, where a researcher abruptly removes an element from a system (e.g., a gene in a mouse) to determine that element's causal effect on other variables [341]. Lastly, even when experimental design is not available to achieve balanced blocks, one can use **propensity score analysis** on non-experimental data to determine the probability that an individual will appear in the treatment group, as a function of the covariates [256]. When such a probability can be determined, one can focus on a subset of a sample where there is balance across groups.

4.9 Bayesian statistics

Until now, our treatment of basic statistics has proceeded following a **frequentist perspective**, where probability is defined as a limit, representing the proportion of successes as the number of trials approaches infinity. Our use of the sampling distribution in Section 4.5 is a clear example of a frequentist approach, since to define a p-value, we had to think about potentially collecting many random samples. An alternative approach is the **Bayesian perspective**, where probability is thought of as a measurement of the plausibility of the event happening. In the Bayesian formulation, an individual can update a believed probability based on data, e.g., ceasing to believe a coin is fair after it comes up heads on 10 flips in a row.

It is possible to teach an entire course using a Bayesian approach,[30] even a first course [281]. The Bayesian philosophy is that one has a prior belief (perhaps based on a previous study, on a hypothesis about the world, or chosen in order to make all outcomes equally likely), then collects data, and then updates the prior belief to a posterior belief. Both beliefs can be modeled as probability distributions, so the language of a **prior distribution** and **posterior distribution** is common. If another experiment is conducted at some future date, this posterior can be taken as the prior for that next experiment. This philosophy lends itself well to computational, iterative models, and to machine learning, both extremely common in data science. We find it is best to give students a flavor of Bayesian thinking, and to teach Bayesian statistics properly as an upper level elective, due to the appearance of sophisticated probability distributions such as the beta distribution, and due to the reliance of the subject upon integrals.

4.9.1 Bayes' formula

Bayesian statistics takes its name from **Bayes' Formula**, from probability theory. For events A and B, we denote by $A \mid B$ the event that A occurs, given

[30]Indeed, it is possible to live an entire life as a Bayesian.

that B occurred. Bayes' Formula states

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)P(B \mid A)}{P(B)}.$$

An excellent application of this formula is to the study of false positives in medical testing. Many students will at some point receive a positive medical test for a frightening condition. Even if this test has a low probability of a false negative, often the probability of a false positive is larger than students realize. For example, suppose a disease affects 5% of the population, and, when people who have the disease are tested, 80% of the tests come back positive. Furthermore, when people who don't have the disease are tested, suppose 15% of the tests come back from the lab marked positive (a false positive result). Let A be the event that a randomly chosen person has the disease, and B be the event that a randomly chosen person tests positive for the disease. If a person has received a positive test, they are interested in $P(A \mid B)$, i.e., the probability that they actually have the disease. In our example, this can be computed as

$$\frac{0.05 \cdot 0.8}{P(B)} = \frac{0.04}{P(B)}.$$

We can find $P(B)$ using the **Law of Total Probability**, which states that

$$P(B) = P(B \mid A)P(A) + P(B \mid A')P(A'),$$

where A' denotes the complement of A. This means $P(B) = 0.8 \cdot 0.05 + 0.15 \cdot 0.95 = 0.1825$, so that $P(A \mid B) = 0.04/0.1825 \approx 0.2192$. This is higher than 0.05, the probability that an individual has the disease if we know nothing about their test status, but it shows that one positive test result is far from conclusive in determining if a person has the disease, in this hypothetical scenario.

4.9.2 Prior and posterior distributions

Another application of Bayes' Formula arises when updating a Bayesian probability. For this section, we follow the treatment of [281], which also contains a number of examples carrying out this procedure using R code. In this context, there is some parameter, θ, and we are interested in the probability density function, $p(\theta)$, telling us the likelihood of the values θ could take. (Compare to Section 4.3.4.) When we start, $p(\theta)$ is the **prior distribution** and can be any distribution we believe describes the probabilities of values of θ (e.g., a uniform distribution, a normal distribution, an empirical distribution, etc.). The distribution version of Bayes' Formula says

$$p(\theta \mid D) = \frac{p(\theta)p(D \mid \theta)}{p(D)},$$

where D represents the dataset that was observed (so, $p(D)$ is the **evidence**), $p(D \mid \theta)$ is the probability of observing the data if the prior were correct (this is called the **likelihood**), and $p(\theta \mid D)$ is the **posterior distribution**. In general, both $p(D)$ and $p(D \mid \theta)$ are very small numbers. For example, to test whether or not a coin is fair, the parameter of interest, θ, is the probability that the coin comes up heads. If we flip the coin 100 times and see an ordered sequence containing 60 heads and 40 tails, $p(D \mid \theta) = \theta^{60}(1-\theta)^{40}$. Given a prior distribution, $p(\theta)$, the Law of Total Probability tells us $p(D)$ is an integral, over all values of θ, of $p(\theta)p(D \mid \theta)$. Choosing a prior, $p(\theta)$, is not easy. If we knew nothing about the coin, we might choose a uniform distribution. If we were completely certain the coin was fair, we might choose a point mass of 1 on the value $\theta = 0.5$. In practice, the prior will be somewhere between these two extremes. It is a theorem that if the prior distribution has a **beta distribution**, then the posterior does, too. For this reason, the beta distribution appears all over a theoretical approach to Bayesian statistics. Nowadays, computers can relatively easily approximate the integral that computes $p(D)$, using a **Markov Chain Monte Carlo** algorithm, such as the **Metropolis algorithm** or **Gibbs sampling** [281, Chapter 7]. Consequently, it is less essential to choose a prior distribution from the family of beta distributions, when doing applied Bayesian statistics.

Bayesian statistics has its own version of model fitting, confidence intervals, and inference. Model fitting is the easiest, since this fundamentally comes down to estimating model parameters, just as in the previous paragraph. Instead of a single value $\hat{\theta}$ estimating θ, we get the entire probability distribution of θ values, which is strictly more information. We can think of $\hat{\theta}$ as the expected value of this probability distribution.

Recall that frequentist confidence intervals and inference are based on the sampling distribution (Section 4.5). The Bayesian version are quite different. The Bayesian version of a 95% confidence interval is a **highest density interval**, and contains the middle 95% of the posterior distribution $p(\theta \mid D)$. This interval spans the 95% most likely values of θ. Note the similarity to bootstrap confidence intervals (Section 4.4.3). For inference, the natural question is whether or not the prior distribution was correct. There are many ways to compare the prior distribution and the posterior distribution to measure how different they are, and to answer this question. One popular way is a **posterior predictive check**.

As [281] demonstrates, there are Bayesian versions of one and two sample tests for comparing proportions and means, for the slope of regression, for the utility of a GLM (like the F-test), for logistic and multinomial regression, and for ordinal and Poisson regression, among many other topics. There are R packages to carry out the computations described in this section, and R code is provided in [281]. Moving to a more complicated model is as easy as adding one more parameter. The Bayesian approach works especially well for hierarchical modeling. Once all parameters and dependencies have been identified, the Gibbs sampling algorithm makes it easy to estimate the parameters iteratively. There is much more to Bayesian statistics than can easily be fit into a single section, and we encourage the reader to add Bayesian techniques to your repertoire. Excellent books include [180, 181, 281].

4.10 A word on curricula

As mentioned in Section 4.1, statistical pedagogy has undergone a renaissance over the past few decades. It is now commonly agreed that the best way to teach statistics is to focus on statistical thinking as an investigative process, to focus on conceptual understanding, to integrate real data with a context and purpose, to use active learning approaches, and to use technology rather than rote hand calculations [13]. It is often advisable to use a project-driven approach, where students carry out weekly guided labs (which can be constructed in collaboration with colleagues from departments all over your campus), culminating in an independent final project [498]. This has the added benefit of giving students practice writing and speaking about statistical investigations, and thinking through some of the ethical considerations inherent to real-world data analysis. Additionally, applied statistics courses are therefore especially attractive to students who desire to see immediate applications of course content to real-world problems (which, in our experience, describes most students).

However, if students are to be required to work with real-world datasets, they will need additional skills with data wrangling and data cleaning. We discuss these topics below, in case the reader needs to teach them. An alternative approach is to require a first course in data science to come *before* a first course in statistics, so that students would begin their statistics course already familiar with a statistical software, and with data wrangling, cleaning, and curating. Such an approach was suggested in [486]. For more about pedagogical uses of statistical computing languages, and some considerations regarding which language to use, we recommend [231, 351].

4.10.1 Data wrangling

Data wrangling, also known as **data munging**, refers to the process of transforming data from its initial format into an easier-to-use format. An example is the process of parsing data from an HTML file (e.g., **web scraping**), or extracting data from a database (e.g., using **SQL**), and writing the data into a comma-separated value (CSV) file. Data wrangling often involves some familiarity with programming, though many statistical computing languages have built-in functions to carry out certain aspects of data wrangling, such as filtering data to consider only individuals with a desired property (e.g., women), grouping data according to some categorical variable (e.g., which state an individual lived in), and computing aggregate statistics such as the number of individuals satisfying the given properties. An example of one such built-in library of functions is the R package dplyr [505], or the pandas module in Python [340]. One can go much deeper into data wrangling. For example, the field of **natural language processing** is intimately tied to the wrangling of textual data.

It is not an exaggeration to say that one could spend an entire semester teaching students the intricacies of data wrangling. (For a sample curriculum, see [63] and the book [62], and for a shorter introduction, see Section 2.4.) In a first course in statistics, it is best to teach students just what they will need to be facile at basic transformations of their data, e.g., those required to work with tidy data [501]. In practice, for elementary statistics, we find that it is appropriate to work entirely with CSV files, but to include a discussion about other forms of data. When merging two datasets, aggregation is often required. For example, if one dataset represents drug overdoses per month, and another dataset represents daily police seizures of drugs, then the only reasonable way to study the impact of police activity on overdoses is to aggregate the second dataset so that it represents seizures per month, making the unit of measurement months rather than days. Such transformations involve applying a function (e.g., sum or count) to a column, and are easy to accomplish with statistical software [257]. If the reader wishes to avoid the need for such transformations, there are still many CSV files that can be analyzed without the need for data wrangling. However, if students are allowed to choose their own dataset for an independent final project, as suggested in [498], then some amount of data wrangling should be expected.

4.10.2 Cleaning data

In addition to data wrangling, many datasets one encounters are fundamentally **dirty** in some way. To get these datasets into a format where one can run statistical analyses, **data cleaning** is required. As with data wrangling, there is far more to cleaning data than can comfortably fit into a single semester course. A taxonomy of dirty data types is provided in [267], and a discussion of common scenarios with dirty data is provided in [521]. If students are to work with real-world data, and especially if they are going to do any independent projects, it is very valuable to teach them about certain common sources of data dirtiness and how to cope with it.

For instance, students are likely to face issues of statistical software interpreting integers as strings, impossible data (e.g., a large number of 0s in a list of blood pressure measurements, perhaps used by the person collecting the data to denote a lack of a measurement), and gaps in data (e.g., if data was kept only for certain blocks of years). Once students are comfortable spotting oddities in the data, statistical software can mutate data to a desired type (e.g., integer), replace impossible data by “NA” entries, filter or drop impossible entries, and group data from different ranges into different clusters, rather than trying to fit a single model to data that represents multiple different populations [257]. If there is time, it can be valuable to teach students about the idea of filling in missing data by joining datasets that feature the same individuals. Another common technique is **multiple imputation**, which fills in missing data by fitting a model (e.g., linear regression) based on the data that is present, and then using this model to predict expected values for the data that was missing. Note that a simpler version of imputation simply replaces missing values in a univariate dataset X with copies of $\overline{x}$. (Imputation was introduced in Section 2.4.3.1.) In the interest of time, such techniques are probably best delayed until a second course in statistics.

Both cleaning data and keeping data tidy are skills students will almost certainly need if they are to conduct an independent final project, or if they are to use the content from their statistics course(s) in subsequent research, consulting, or employment settings. If one uses a project-based approach to teaching, then students can practice these skills by being given successively less tidy and dirtier data for each lab. However, it is possible (and perhaps advisable, given time constraints) to teach a first course in statistics where students are provided clean and tidy data for each assignment, and to relegate the teaching of data cleaning and tidying to a data science course.

4.11 Conclusion

In this chapter, we have given a high-level overview of the essential topics from basic statistics required for a working data scientist. We have emphasized topics that should be included in a first or second course in statistics, and have tried to identify areas that historically give students difficulty. We have highlighted areas where abstract thinking skills and advanced knowledge of mathematics can be useful, and we have provided numerous references where a reader can explore these topics more deeply. A large part of the author's approach was inspired by the textbooks [256], [316], and [101], which together cover both a first and second course in statistics. For a more traditional approach to a first course, one might follow [384]. If the audience is future engineers, [225] (more theoretical) or [248] (more applied) may be appropriate. Other excellent choices for a second course include [285, 401], or for an audience with a strong mathematical background, [288]. An excellent resource for anyone teaching statistics at any level is [182].

4.12 Sample projects

In this section, we give a few sample projects where the reader can practice using the ideas in this chapter. All use built-in R datasets, because we intend the focus to be on the statistics rather than obtaining and cleaning data, but the reader who wishes to increase the challenge may feel free to find datasets in the wild instead. Readers who wish to do this work in another programming language may be able to access the same data; for example, Python and Julia provide "RDatasets" packages that can load the same data for analysis in those other languages.

1. Using the built-in dataset `SAT` that comes with the mosaic package of R [394], create a multivariate regression model to predict the average SAT total score in each state, based on expenditure per pupil, the average pupil/teacher ratio, the estimated average annual salary of teachers, and the fraction of eligible students taking the SAT. It may be helpful to refer back to the R code provided throughout the chapter. Use a model selection technique to find the best model you can, be sure to check the assumptions of regression, and consider using cross-validation to avoid overfitting. Be sure to assess the final model you find, reporting the results of statistical inference, confidence intervals, and any concerns about outliers and influential points. If you find outliers, consider using ridge or LASSO techniques (Section 4.7.2) to create alternative models.

2. Using the built-in dataset `Whickham` that comes with the mosaic package of R [394], create a logistic regression model for the probability that an individual had a positive `outcome` (i.e., was still alive after 20 years) based on their age and smoker status.

3. Using the built-in dataset `CPS85` that comes with the mosaic package of R [394], create a two-way ANOVA model to determine if there is a significant difference in `wage` among white and non-white individuals, and among men and women. You may wish to use interaction plots. If you wish, you can extend your model by including `age` as a covariate, or other covariates you desire.

4. Conduct exploratory data analysis on the `Alcohol` dataset that comes with the mosaic package of R [394]. Create at least three interesting graphics, as in Figures 4.1 and 4.2. If necessary, create additional categorical variables, e.g., for which continent each country is on. For a challenge, try to create a choropleth map shading each country in the dataset by alcohol consumption per capita.

5. Test your knowledge of confidence intervals and inference (Section 4.5) on the dataset `KidsFeet` that comes with the mosaic package of R [394]. Create confidence intervals (using both methods discussed above) for the mean of the variable `length`, and do inference (using both methods discussed above) to see if there is a statistically significant correlation between the length and width of a kid's feet.

6. (Challenge) The SAT problem implicitly treats states as independent. A more advanced statistical model would proceed via spatial data analysis,

mentioned in Section 4.6.4. If you feel inspired, teach yourself about spatial statistics and consider using the `spatial` package in R to formulate a model using the spatial aspect of the data.

7. (Challenge) Teach yourself about time series analysis (mentioned in Section 4.6.4) and fit an ARIMA model to the built-in `AirPassengers` dataset that comes with R. Use this model to forecast ahead, and predict the numbers of airline passengers in the three months after the dataset ended. You can also look up the real number of airline passengers in those months, to see how well your model performed.

Chapter 5

Clustering

Amy S. Wagaman

Amherst College

5.1 Introduction

5.1.1 What is clustering?

Clustering refers to a collection of methods which are designed to uncover natural groups, called clusters, in data. The idea is that the groups should contain objects similar to each other, and the groups should be as different as possible. Some mathematical attempts to quantify the degree of similarity and difference necessary for clusters to be distinct have been pursued, but no consensus exists. In short, we recognize clusters when we see them (and sometimes find them when we should not).

The groups are unknown prior to applying the method, so these are often called unsupervised techniques. Uncovering natural groups may lead to better understanding of the dataset, showing relationships between observations, or effectively creating a classification of the observations, and can be an analysis end goal in and of itself. In other settings, clustering is a pre-processing step and the groups found are further investigated as part of some other analysis. Clustering is usually designed to find groups of observations, but can be used to find groups of variables as well.

The applications of clustering are as wide-ranging as the methods are; indeed, clustering methods are often adapted to their applications. Clustering can be used to find communities in social networks (often called community detection), identify documents with common stylistic patterns to help ascertain authorship, find a new consumer group for marketing, study relationships among biological species, find genes that may have similar expression patterns, and much more. Briefly, we expound on some applications.

5.1.2 Example applications

Have you ever wondered what the referral network between physicians looks like? Landon et al. examined referrals among over 4.5 million Medicare beneficiaries and their nearly 70,000 physicians in order to find what referral communities existed. Their interest was prompted by the passage of the Affordable Care Act and the desire to investigate creation of Accountable Care Organizations. Finding clusters of physicians who are revealed to already work in a community together may help in creating Accountable Care Organizations [299]. What about examining communities found using social networks to track linguistic shifts? A study in Raleigh, North Carolina examined a particular linguistic variant called the southern vowel shift and how changes in linguistic patterns were reflected over time in communities found using a community detection algorithm called Infomap [404].

Clustering has also proven useful for examining authorship. Work from the early 1990s examined authorship of the *Book of Mormon* and related texts using cluster analysis to group text samples together based on literary properties. (There is much more to be said for text analysis here, but our focus is on clustering.) Specifically, hierarchical clustering was used to look for groupings of text samples that may have (or are already known to have) the same author or be written in the same style. Additionally, text samples that were outliers with regard to the set can also be identified via clustering, and a few were noted in the study [226]. More advanced clustering methods were used to examine authorship as it relates to the book *Tirant lo Blanc*, for which single author and two-author claims have been made, with some detail about when the second author would have taken over writing the text. A Bayesian approach to clustering was undertaken, with an aim to allow for comparison of a single-cluster or two-cluster solution (to match the authorship claims). The result of the analysis lends support to the two-author claim (or at least, a change in style that could be associated with such a claim) [188].

If you're interested in understanding relationships among biological species, clustering is used there, too. Phylogenetic trees are used to show believed evolutionary relationships between organisms based on similarities. A 2012 study clustered soil bacteria organisms. The study authors were interested in examining how seasonal change and environmental characteristics affected phylogenetic relationships [429]. Beyond that, statisticians have looked into methods for evaluating and comparing phylogenetic and clustering trees using an idea of distance between trees in a microarray setting [87].

What about finding similar genes, or genes that work in concert? Gene expression studies often cluster genes together looking for various relationships. Lotem et al. examined genes that are over-expressed in cancers relative to normal tissues. They found differences between the types of cancers in terms of which genes were over-expressed [321]. If you are interested in examining expression levels over time and seeing which gene clusters are found, you can try a Bayesian model approach such as the one proposed in Fu et al. [173]. Their method is designed to handle fairly noisy time series data and is able to estimate the number of clusters from the data itself.

5.1.3 Clustering observations

Now that we've discussed a handful of applications where clustering has been used, we can get to the business of developing our understanding of clustering methods. We will assume we have n observations on p variables, and focus on clustering the n observations into k clusters. While it is possible to cluster the p variables (sometimes called the features) with appropriate adjustments to methods, our focus will be on using the p variables to cluster the observations. The variables may be quantitative or categorical, though categorical variables present some challenges. For most of our presentation, we assume the variables are quantitative, and address working with categorical variables in a few places. We will assume there is no missing data as that is a separate issue to be dealt with carefully, as in Section 2.4.3.1. The variables are assumed to have been chosen as inputs to the clustering algorithm; most clustering algorithms do not perform variable selection. We'll address this challenge later in the chapter.

There are a wide range of clustering algorithms available to tackle this unsupervised learning challenge. We will examine the properties and applications of a few key methods, after some necessary setup. First, we examine visuals that might suggest clustering is appropriate for your data. After all, it is possible that no clustering solution is really appropriate for your data (i.e., that no clusters really exist).

5.2 Visualization

Visualizing your data is one way to determine if clustering might be an appropriate analysis tool. However, multivariate visualization can be a major challenge. So, let's start with univariate ideas that show how clusters might show in your data. For these examples, we will use the `iris` dataset [19]. This dataset was collected by Edgar Anderson, and has been used in demonstrating statistical ideas starting with the statistician R.A. Fisher in 1936 [161]. The dataset consists of 150 observations on irises, with 50 observations from each of three species, with four measurement variables observed. One of the measurement variables is the length of the iris petal. A density plot of the variable for all 150 irises reveals two major modes, as shown in Figure 5.1. Seeing the two modes might lead us to suspect that there are several groups (at least two) present in the dataset. Here, that is indeed the case, one species has much smaller petal lengths than the other two.

What about looking for patterns across several variables? Well, a scatterplot matrix can be used, if the number of variables is relatively small. This allows for bivariate exploration to see if clusters may be present. A scatterplot matrix allows for the examination of many bivariate relationships all at once; all pairs of included variables have their own scatterplot displayed. In some software, only the lower triangular portion of the matrix is shown. Others show both portions, so that each variable plays the role of X in one plot and plays the role of Y in the other. In our chosen software, R, the upper triangular portion shows the Pearson correlation coefficient for the corresponding relationship and the matrix diagonal is a univariate density plot [397]. This plot can be very useful to examine the relationships among a few variables, but grows unwieldly and hard to read as the number of variables increases.

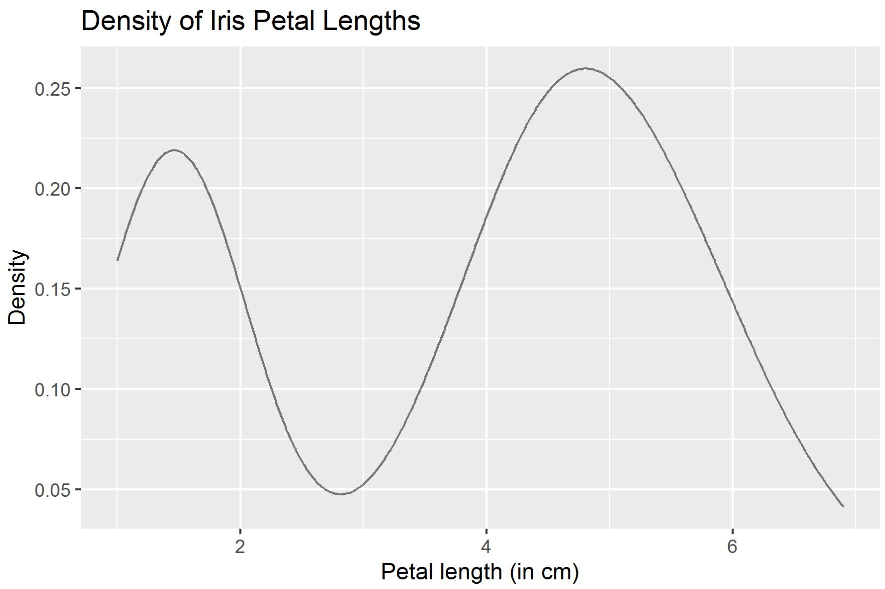


FIGURE 5.1: Density plot (i.e., smoothed histogram) of petal length variable for `iris` data.

Instead of looking at a scatterplot matrix for all p variables at once, a common strategy is to examine multiple subsets of the variables. In tasks such as regression analysis, the response variable would be included in each subset examined. In our example, there are only four variables, so the matrix is fairly easy to read. The scatterplot matrix of the `iris` dataset is shown in Figure 5.2. Again, visually, it appears there may be at least two clusters of observations when looking at the separation of data points in the different scatterplots.

There are options for other visuals to be used, including projecting to lower dimensions (such as with principal components analysis), which is the approach we will take for visualizing some clustering solutions below. For information on additional visualization options, the reader is directed to Everitt et al. [149]. Principal components analysis is covered in Section 7.3.

5.3 Distances

For many clustering methods, a key choice that must be made by the analyst is the choice of metric for dissimilarities or distance between observations. After all, if we are interested in finding natural groups of similar observations,

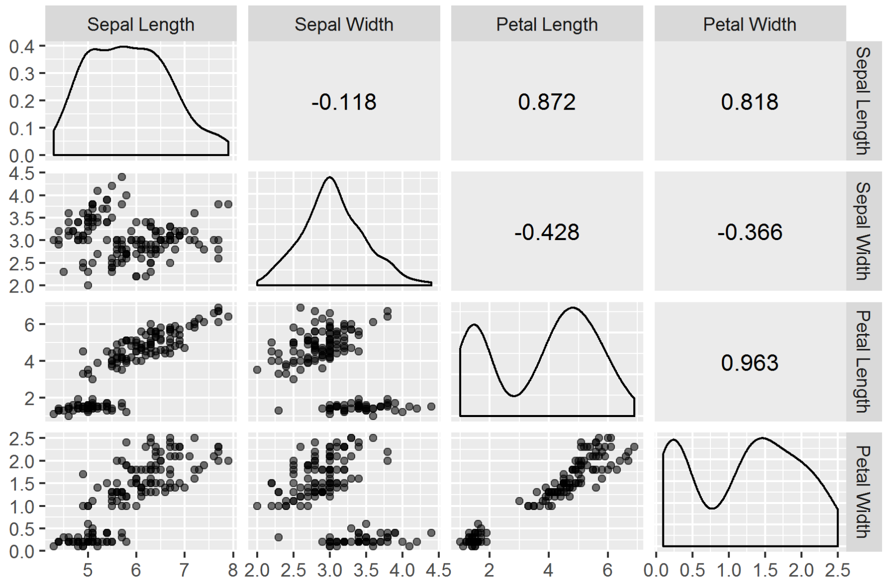


FIGURE 5.2: Scatterplot matrix of the four measurement variables in the `iris` data.

we must have some measure of similarity or dissimilarity between the observations. Some sources call this proximity [243]. There is a minor distinction to be made between dissimilarities and distances. For a measure to be a dissimilarity, we assume that the resulting values are non-negative, the dissimilarity from an observation to itself is 0, and that dissimilarities are symmetric. Dissimilarity values are low when objects are similar and high when they are different. For notation, let x_i represent the i^{th} data point, which is a p-dimensional vector. Then if we take x_i and x_j, any pair of observations, $i \neq j$, and let $d(x_i, x_j)$ be the value of the dissimilarity measure, then we require all of the following.

1. $d(x_i, x_j) \geq 0$,
2. $d(x_i, x_i) = 0$, and
3. $d(x_i, x_j) = d(x_j, x_i)$.

However, most common dissimilarity measures also satisfy the triangle inequality, making them metrics. In this case, we swap from calling them dissimilarities to calling them distances. Satisfying the triangle inequality means that for any three observations x_i, x_j, and x_k, we have that $d(x_i, x_j) \leq d(x_i, x_k) + d(x_k, x_j)$.

If clustering is being performed on the variables rather than observations, common dissimilarity measures involve the Pearson correlation coefficient, r, (or its absolute value) between the variables. These measures do not satisfy the triangle inequality.

Intimately related to the choice of dissimilarity or distance is whether or not the variables are scaled or standardized in some way. The most common method of scaling is to scale so that each variable has mean zero and variance one, by subtracting the mean and dividing by the standard deviation. Scaling in other ways is possible, such as putting all variables on a scale from 0 to 1 by shifting the minimum value to 0 by adding a constant to all values, and then dividing all values by the new maximum value. If variables are not scaled, and are on very different scales, a variable with a large range may dominate the distance measure and hence be the main driver of the clustering solution. Finally, variables can be given different weights as part of this construction, but by default, we assume all variables are weighted equally in the dissimilarity or distance measure. For simplicity, we will refer to these measures as distances, though for a particular application, they may be only dissimilarities.

Several common options for distance measures are described next.

1. Euclidean distance: The most common and probably most easily understood distance measure is the typical Euclidean distance between two observations. The usual formula applies, so that

   $$d(x_i, x_j) = \left[\sum_{k=1}^{p}(x_{ik} - x_{jk})^2\right]^{1/2}.$$

   A variant is sometimes used that is squared Euclidean distance (i.e., just omit the square root).

2. Manhattan (city-block) distance: Another common distance measure, Manhattan distance uses absolute values between variable values for the pair of observations to compute the distance between them. This means the distance formula is

   $$d(x_i, x_j) = \sum_{k=1}^{p} |x_{ik} - x_{jk}|.$$

   When thinking of this distance on the Cartesian plane, the idea is that one is forced to walk between points on the grid lines, and cannot take diagonal steps, which is where the city-block analogy originates.

3. Minkowski l-distance: Euclidean distance and Manhattan distance are special cases of Minkowski distance, where the distance formula is given by

   $$d_l(x_i, x_j) = \left[\sum_{k=1}^{p} |x_{ik} - x_{jk}|^l\right]^{1/l}.$$

In particular, Euclidean distance is $l = 2$, and Manhattan distance is $l = 1$.

4. Mahalanobis distance: Mahalanobis distance may be useful for clustering because it takes the covariance matrix (variability information, as defined in Section 4.7.1) into account in the distances. This distance was originally proposed between a point and a distribution with a population mean and covariance matrix, so that it measured how far from the mean of the distribution the point was following the principal component axes which were derived from the covariance matrix. For our purposes, it can be expressed as a distance between two points assuming they are from the same distribution, with a shared population covariance matrix. The sample covariance matrix, S, with dimension $p \times p$, is typically used as an estimate of the population covariance, so that the distance between the two points is given by

$$d(x_i, x_j) = \left[(x_i - x_j)^T S^{-1} (x_i - x_j)\right]^{1/2}.$$

   Note that if the sample covariance matrix is taken to be the identity matrix, this reduces to Euclidean distance. This distance is often used in k-means clustering [97]. Note that it is a quadratic form, as defined in Section 3.2.3.

5. Gower's distance: The four distances above have been described for numeric variables. They can be applied on numerically coded categorical variables (say, if one has created indicator variables, as we discuss below), but these methods are not designed to deal with categorical data. (Indicator variables were defined in Section 4.6.5.) Gower's distance is a distance measure that is able to work with a mixed collection of variable types. The reader is directed to Gower [197] for more details. The distance has been implemented in several software packages (including via the `daisy` function in R).

For distances that rely on quantitative inputs, how might a categorical variable be incorporated? One can designate one of its levels as a reference level and create indicator variables (which take the value of 1 for that level, and 0 for all other levels) for all its other levels, or use one-hot encoding which creates an indicator variable for each level. Creating this series of indicator variables has an impact on the distance measure (the indicator variables may dominate the distance measure), but does allow the incorporation of categorical variables into clustering analysis when working with most distances.

5.4 Partitioning and the k-means algorithm

Over the years, a variety of different clustering methods have been developed. Some of the earliest methods still enjoy tremendous popularity. These methods include partitioning methods, including k-means, and hierarchical clustering methods. These methods are popular because they are simple to understand and are implemented in most modern software packages for data analysis. However, they do not have a strong theoretical foundation, and there are computational considerations with the size of modern datasets. More recent methods have been developed that have stronger mathematical foundations, including model-based methods, and approaches to deal with large-scale data exist.

Another class of methods that is fairly popular is density-based methods, which can even learn the number of clusters from the data. Bayesian methods have also been developed (though these may still be partitioning or hierarchical) to capitalize on that framework. Finally, specific clustering methods have been developed for use on networks (or on data that has been structured like a network). Briefly, we examine these methods and some of the algorithms used to implement them.

For the methods that follow, we will assume we arrive at solutions with k clusters, whether k may be specified in advance or not, depending on the algorithm, and the clusters are denoted $C_1, \ldots, C_k$. For most of our methods, our setup means that observations end up belonging to exactly one cluster. This is considered "hard" cluster membership [97]. Typically, a cluster allocation vector or membership function is created as output from these methods. In the allocation vector, entries denote which cluster each observation (or variable, as appropriate) belongs to. The membership function simply maps observations into their corresponding clusters. For example, if we let the membership function be denoted C, then $C(i) = j$ means that observation x_i is in cluster C_j. Suppose we know that $C(12) = 3$. Then, we know that observation x_{12} belongs to C_3, and in the cluster allocation vector, the 12^{th} entry would be 3.

5.4.1 The k-means algorithm

Partitioning methods, including the k-means algorithm, attempt to take the observations and partition them into k clusters based on creating groups of observations that are similar to one another based on the distance measure, where the groups themselves are far away from one another as well. Partitioning means the union of the clusters $C_1, \ldots, C_k$ is the entire set of n observations, but the intersection of any two clusters C_i and $C_j, i \neq j$, is the empty set. The subclass of algorithms to which k-means belongs is the class of centroid-based algorithms, as other partitioning methods exist which do not focus on centroids. K-means has a long history, being one of the earliest

TABLE 5.1: Number of partitions of n observations into k clusters for selected small values of n and k.

Sample size (n)	Number of clusters (k)	Number of Partitions
8	2	127
10	3	9330
15	4	42,355,950
21	5	3,791,262,568,401

clustering methods developed, and was independently discovered in several disciplines. (For more details, the reader should see Jain [245].) The term k-means was first used in 1967 by MacQueen [330].

In k-means, the partitioning of observations sought is one that minimizes the total squared error (or some other measure of within-cluster variation, as covered in Zhao [519]) between each point in each cluster and its corresponding cluster centroid (mean). In other words, the within-cluster variation should be at a minimum with the chosen metric, which is commonly squared Euclidean distance. The partition is usually solved for iteratively, rather than setting it up as an optimization problem to solve. The challenge is that, with n observations and the specified number of clusters k, the number of possible partitions is immense. Specifically, the number of possible partitions can be found as the Stirling number of the second kind with the reported n and k values. We report a few example values in Table 5.1 to show the scale of the problem even with very small n and k to demonstrate that it is practically infeasible to search through all the partitions.

So, how will we go about finding our solution? Our outline of the k-means algorithm follows.

1. Set a value for k, the number of clusters, and determine the distance measure being used.

2. Initialize k starting cluster centroids; these may be actual data points or random points.

3. Assign each data point $x_i, i = 1, \ldots n$, to the cluster $C_1, \ldots, C_k$ that it is closest to (measured by its distance to the cluster centroid).

4. Update the cluster centroids for all k clusters based on the updated cluster assignments. The cluster centroids are found as the average of all observations in that cluster. For C_j, the cluster centroid is

$$\frac{\sum_{i \in C_j} x_i}{|C_j|}.$$

5. Repeat steps 3 and 4 until convergence (i.e., the cluster assignments no longer change).

Note that the algorithm is strongly dependent on the user specified value of k. However, k is usually unknown, so how can we use this algorithm? Well, you can try a k value found by another algorithm, or you can find solutions for several values of k, and compare the squared error between solutions, noting of course that increasing k will reduce the squared error some. We can find the within-cluster sum of squares using the Euclidean distance metric as

$$\sum_{j=1}^{p}\sum_{l=1}^{k}\sum_{i\in C_l}(x_{ij} - \bar{x}_j^{(l)})^2,$$

where we note that the sum is over all observations (by variable and cluster) and compares each observation's value on each variable to the mean of the variable for the cluster. Alternative metrics may be chosen to compare the solutions to help choose the value of k.

How will we choose k? We can try evaluating the solution for many values of k and see which we like best. Some sources advocate for using an "elbow" method approach in regards to the squared error, like one might see for choosing the number of principal components to retain in principal components analysis, where the squared error is plotted on the y-axis versus the number of clusters k on the x-axis. An "elbow" in this plot, meaning a sharp drop-off followed by a near plateau (or slight decrease) is the sign that the number of clusters after the drop-off and at the start of the plateau is a good balance between the squared error and k [148]. While the error will naturally decrease as k increases, choosing too large a k will overfit the data (putting each observation in its own cluster, for example). Instead, we look for k after several big drops in error before the curve starts to level out, which indicates we are finding signal rather than noise. We will demonstrate this approach in our example below and see other approaches later in the cluster validation section.

5.4.2 Issues with k-means

Every algorithm has advantages and disadvantages to its use. K-means is often one of the only practical algorithms to run in a setting with high p and low n [97]. However, it has significant disadvantages to be aware of. For example, k-means is known to converge to local minima, even when the goal is to find a global optimum [245]. Indeed, it is possible to get k-means "stuck" at a local optimum due to poor choice of initial cluster centroids. Let's consider this via a simulation. First, we generate bivariate normal data with 3 clear clusters as shown in Figure 5.3.

Next, we initialize the k-means algorithm with unlucky initial random values for the centroids, with two points in the single "bottom" cluster, and one point between the two "top" clusters as our centroids. The algorithm runs but is unable to recover the original three clusters. Instead, the solution is presented in Figure 5.4.

In order to alleviate this problem, often multiple random starts or starts from within the closed convex hull of the data are used for the initial cluster

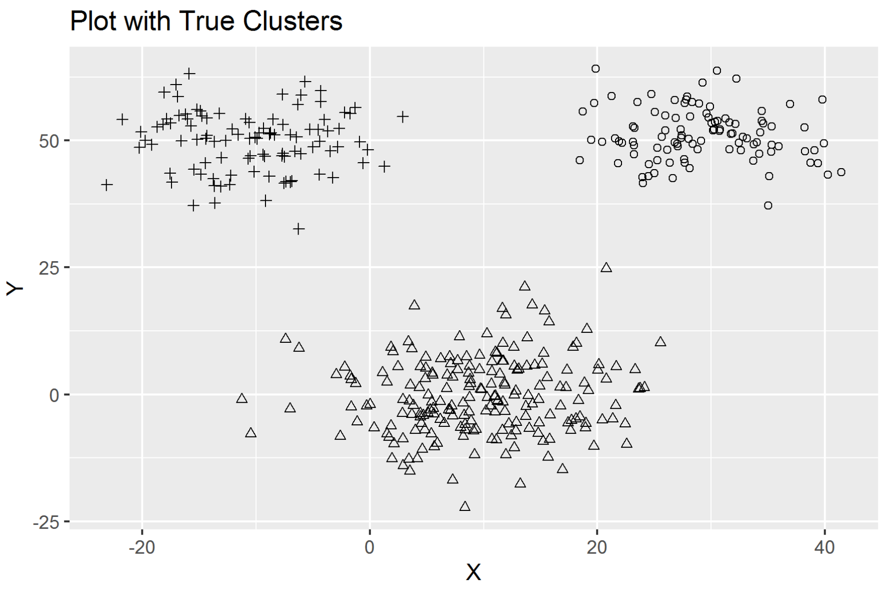


FIGURE 5.3: Simulated bivariate normal data with three clusters, represented as plusses, circles, and triangles.

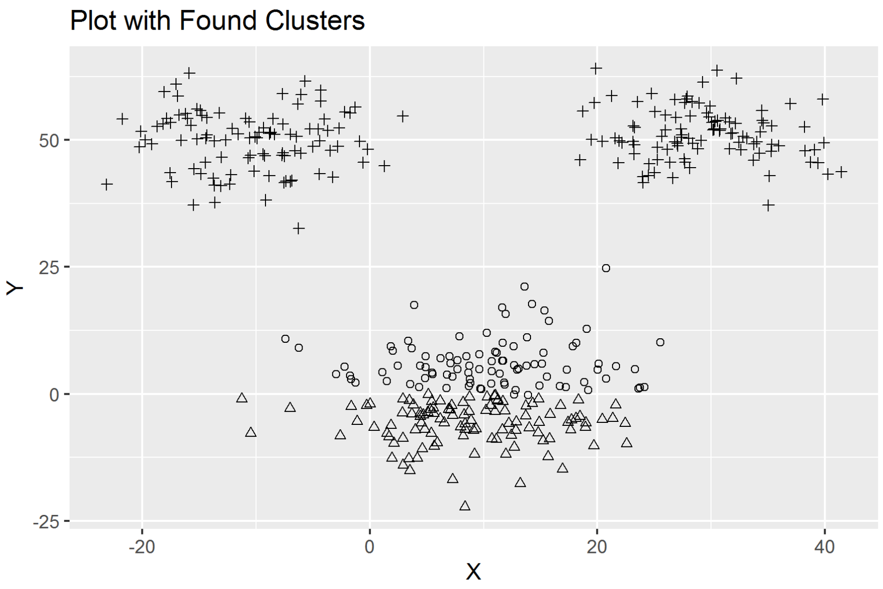


FIGURE 5.4: Found clusters via k-means with poor initial starting centroids. Note that the top two true clusters have been united into one found cluster, and the bottom true cluster has been separated into two found clusters.

TABLE 5.2: Summary statistics on selected variables from a wine dataset.

Variable	Mean	Standard Deviation	Minimum	Maximum
Alcohol	13.00	0.81	11.03	14.83
Malic Acid	2.34	1.12	0.74	5.8
Ash	2.37	0.27	1.36	3.23
Alcalinity of Ash	19.49	3.34	10.6	30
Magnesium	99.74	14.28	70	162
Total Phenols	2.30	0.63	0.98	3.88
Flavanoids	2.03	1.00	0.34	5.08
Nonflavanoid Phenols	0.36	0.12	0.13	0.66
Proanthocyanins	1.59	0.57	0.41	3.58
Color Intensity	5.06	2.32	1.28	13
Hue	0.96	0.23	0.48	1.71
OD Diluted Wines	2.61	0.71	1.27	4
Proline	746.89	314.91	278	1680

centroids in k-means, and the solutions are compared to see if they agree (hopefully at the global solution), or the one with the smallest within-cluster variation can be chosen as the solution [97].

K-means is not scale invariant, meaning that solutions can differ significantly when obtained on standardized and unstandardized versions of the data. (This is true for many clustering algorithms.) It is also known to find clusters that are "spherical," which may not represent the actual clusters present [148]. With problems like these, alternative partitioning algorithms have been developed, which are briefly discussed after our k-means example.

5.4.3 Example with wine data

For demonstrations of our clustering methods, we will use the wine dataset from the University of California–Irvine's (UCI's) machine learning repository [136]. Our analyses are performed using the software R [397], using the RStudio interface [413]. The data consists of 178 observations on wine from Italy from three different cultivars. (A cultivar is a plant variety obtained by selective breeding.) We can use the various measurements on the wine (disregarding the cultivar variable) to attempt to cluster the wines. The measurement variables and some basic descriptive statistics about them are listed in Table 5.2. Values were rounded to two decimal places. Unfortunately, the repository does not have an in-depth description for each variable, nor do we have details on how the variables were obtained or their units.

All 13 of the measurement variables are quantitative, but clearly have different scales, so we will standardize them when performing our analysis so that the ones with higher variance in the original scale do not dominate the distance measure, by subtracting the mean and dividing by the standard deviation (so that after standardizing, each variable will have mean 0 and

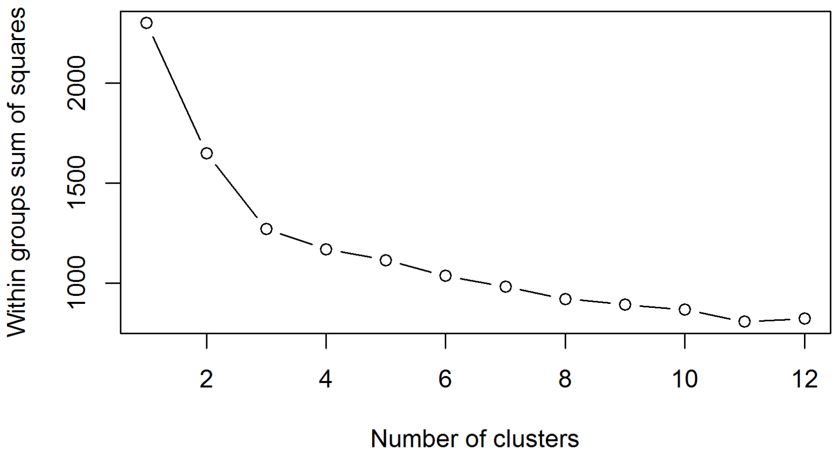


FIGURE 5.5: Within-groups sum of squares for solutions with $k = 1$ to $k = 12$ clusters for the wine example in Section 5.4.3.

standard deviation 1). We will use standard Euclidean distance as the distance measure. If we were interested in including the cultivar variable, we would need to use indicator variables or swap to Gower's distance, but here, it may instead be interesting to see if the recovered clusters match the cultivars.

In order to determine the number of clusters we want to recover, we will use the "elbow" method approach, described previously. Remember that the number of clusters after the drop-off and at the start of the plateau, the elbow, is a good balance between the squared error and k [148]. We found k-means solutions for $k = 1$ to $k = 12$, and have plotted the squared error (within-group or within-cluster sum of squares) versus k in Figure 5.5. There is a fairly clear elbow at $k = 3$ clusters so we will use the solution with three clusters.

This k-means solution was found using the `kmeans` function in R on the scaled dataset (using the `scale` function for the standardization). The solution provides us with an allocation vector, so we can see which cluster each observation belongs to, the size of the different clusters, as well as the cluster centroids. The centroid values for each variable by cluster are shown in Table 5.3. Bearing in mind that these are scaled values, we are able to ascertain some differences between the clusters in terms of their centroids. If desired, we could unscale and put the centroids back in the original variable units as well.

The three clusters are fairly similar in size, with 51, 62, and 65 observations respectively. Typically, we want to visualize the clustering solution, however we are not able to plot in thirteen dimensional space. Instead, a common representation of the solution is to plot the observations in two- (or potentially

TABLE 5.3: Centroids for each cluster in the clustering solution found for the wine dataset.

Variable vs. Centroid	Cluster 1	Cluster 2	Cluster 3
Alcohol	0.16	0.83	-0.92
Malic Acid	0.87	-0.30	-0.39
Ash	0.19	0.36	-0.49
Alcalinity of Ash	0.52	-0.61	0.17
Magnesium	-0.08	0.58	-0.49
Total Phenols	-0.98	0.88	-0.08
Flavanoids	-1.21	0.98	0.02
Nonflavanoid Phenols	0.72	-0.56	-0.03
Proanthocyanins	-0.78	0.58	0.06
Color Intensity	0.94	0.17	-0.90
Hue	-1.16	0.47	0.46
OD Diluted Wines	-1.29	0.78	0.27
Proline	-0.41	1.12	-0.75

TABLE 5.4: Comparison of clustering result to original cultivar variable.

k-means Cluster vs. Cultivar	1	2	3
1	0	3	48
2	59	3	0
3	0	65	0

three-) dimensional principal component space. The principal components are obtained (using the correlation matrix here due to scaling issues) and principal component scores (new coordinates) for each observation are obtained. Then, we plot in that lower-dimensional space with the understanding that it is encapsulating a high percentage of the variation in the original data (55% here for the two-dimensional solution, which may be a little lower than we'd like). In Figure 5.6, we can see that the clustering solution shows a somewhat clear separation of the clusters in the principal component space. (Readers new to principal components analysis should see Section 7.3.)

Finally, since we understood that the wines were collected from three different cultivars, we can examine our clustering solution to see if the clusters capture the cultivar variable. In other words, we are examining whether or not the naturally occurring groups found based on the observations correspond to the different wine cultivars. The short answer here is a resounding yes. As shown in Table 5.4, the clusters line up very well with the cultivars. There was no expectation that this would be the case; indeed, the cultivar variable was not used to find the clustering solution. This example shows that unsupervised learning can result in practically meaningful findings, and illustrates why clustering is used to look for naturally occurring groups of observations. The clusters found may be very useful in identifying some underlying structure.

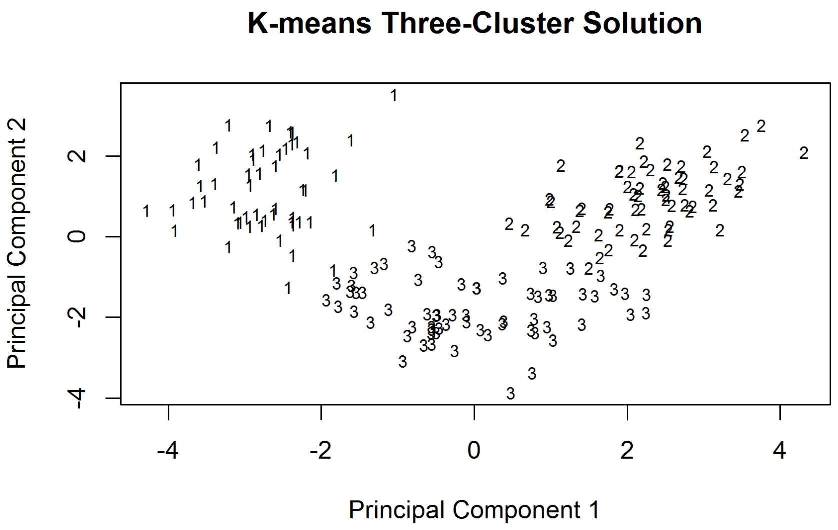


FIGURE 5.6: Three-cluster k-means solution plotted in principal component space. Cluster assignment is plotted for each observation.

TABLE 5.5: Comparison of three- and four-cluster solutions for the wine dataset.

k-means Clusters	1	2	3	4
1	49	0	2	0
2	0	55	0	7
3	0	1	43	21

We can use similar tables to compare different clustering solutions if they are obtained. For example, we can compare a k-means three-cluster solution to a k-means four-cluster solution. Note that since k-means is a partitioning method, there is not necessarily any relationship between the three- and four-cluster solutions. As Table 5.5 shows us, there are some similarities between the clusters recovered, as it appears one cluster from the three-cluster solution was split in two, but the other two clusters from the three cluster solution are largely intact.

Now that we've explored k-means via an example, we can turn our attention to another issue relevant to any clustering solution, the topic of cluster validation.

5.4.4 Validation

How do we know if our clustering solution has any value? In other words, how do we validate or evaluate our clustering solutions? We've seen in our examples that we can compare clustering solutions to see if the clusters are similar with some basic contingency tables, but this isn't really validating the solution. Are there statistics that can help us in this endeavor? The answer is yes, though not all are appropriate in every situation.

A useful statistic (which has an associated graphic) is the silhouette coefficient, first proposed by Rousseeuw in 1987 [412]. This statistic was developed to help visualize clustering solutions from partitioning methods. However, it can be computed for any clustering solution. First, silhouette values are computed for each individual observation. To understand these values, we need some notation. We will assume we are looking at the i^{th} observation, x_i, and that we have a clustering solution with k clusters already computed, and a dissimilarity measure chosen. To compute the silhouette value, we need to identify the cluster the i^{th} observation is in (say C_i), and also what it's next-best cluster fit would be (say $C_{i'}$). In other words, we need to identify which cluster it belongs to best, ignoring its current assigned cluster. In order to identify which cluster is $C_{i'}$, for each cluster c other than C_i, we compute $d(x_i, c)$, which is the average dissimilarity of the i^{th} observation to all elements in cluster c. We take the minimum of those average dissimilarities, and denote it as $b(i)$, i.e.,

$$b(i) = \min_{c \neq C_i} d(x_i, c).$$

The cluster that yielded that minimum is taken to be $C_{i'}$, and is called the neighbor of x_i [412]. We also compute the average dissimilarity $d(x_i, C_i)$ to obtain the average dissimilarity of x_i to all objects in its own cluster, which is denoted $a(i)$. The silhouette value is a comparison of $a(i)$ and $b(i)$, appropriately scaled. Intuitively, for good clustering solutions, we'd like the $a(i)$'s to be smaller than the $b(i)$'s for all observations, indicating that observations fit better in their assigned cluster than in their cluster neighbor. The formula for the silhouette value, $s(i)$, is given by

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

which results in $s(i)$ values between -1 and 1. Values near -1 indicate that the observation would belong better in its cluster neighbor than its current assigned cluster, while values of 1 indicate it fits very well in its own cluster, because its within-cluster average dissimilarity is much less than the average dissimilarity to its cluster neighbor. Values near 0 indicate the observation would belong equally well in both clusters.

Now that we have silhouette values for each observation, we can use them to construct averages per cluster or other statistics of interest. Looking at the overall maximum of the silhouette values may reveal no clustering structure is really present (if the maximal $s(i)$ value is less than, say, 0.25) [243]. A plot displaying the $s(i)$ values sorted in descending order within each cluster is called a silhouette plot. In Figure 5.7, we examine the silhouette plot for the three-cluster solution for the wine data we obtained using k-means. The plot was constructed with the `silhouette` function in the `cluster` library in R [331].

Note that the average of the silhouette values for each cluster has been reported. This value is called the average silhouette width for the cluster, $\bar{s}_j$,

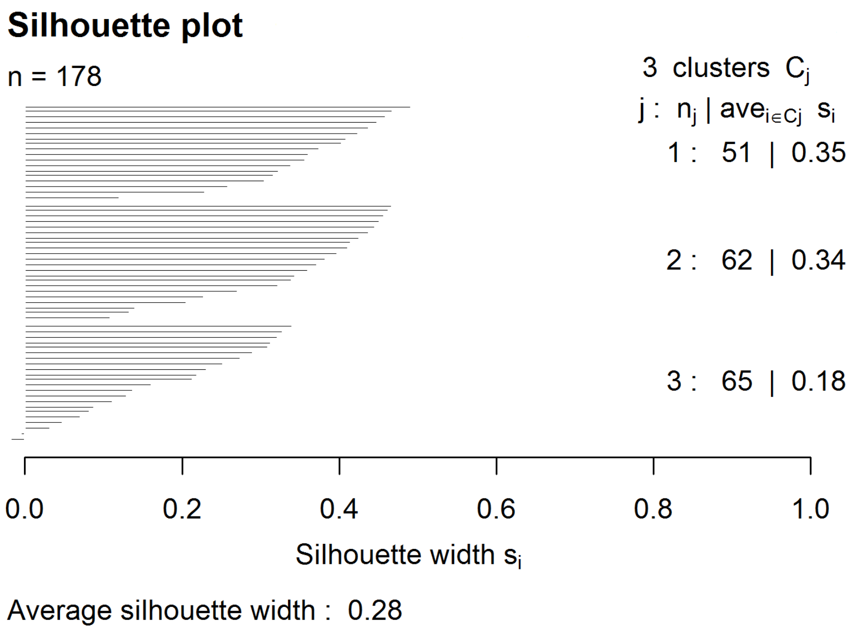


FIGURE 5.7: Silhouette plot for k-means three-cluster solution on wine data.

$j = 1, \ldots, k$. These values can indicate which clusters are "stronger" than others. We also see that the average silhouette width for the entire dataset for this choice of k has been reported. This value is denoted $\overline{s_{(k)}}$ because it applies to the entire solution with k clusters. In our example, $\overline{s_{(3)}} = 0.28$.

Now, suppose we fit clustering solutions with k varying from 2 to $n-1$, and we look at the average silhouette widths (the $\overline{s_{(k)}}$ values) across the solutions. The maximum of these values is called the silhouette coefficient, SC. In other words,

$$SC = \max_k \overline{s_{(k)}},$$

for $k = 2, \ldots, n-1$. Finding the silhouette coefficient is useful for two reasons. First, it provides one with a reasonable choice of k; whichever solution had the maximal average silhouette width is arguably the best solution. Second, it provides an assessment of the overall strength of the clustering solution. Kaufman and Rousseuw proposed the interpretations in Table 5.6 for SC values [263].

We ran a short R script to find the SC value for our k-means solutions, using $k = 2, \ldots, 177$ since there are 178 observations in the dataset. The SC value turns out to be the 0.28 reported for our solution with $k = 3$. So, this was a good choice for k, based on this statistic, even though it was chosen via other means originally. (The $k = 4$ solution had an average silhouette width of 0.26.) However, even though this was the maximal width for the dataset found, the provided interpretations suggest the clustering structure is not very strong.

TABLE 5.6: Kaufman and Rousseeuw's proposed interpretations for silhouette coefficients.

SC	Interpretation
0.71-1.00	A strong structure has been found.
0.51-0.70	A reasonable structure has been found.
0.26-0.50	The structure is weak and could be artificial; try additional methods.
< 0.25	No substantial structure has been found.

A similar statistic called the shadow value has been developed. For details, see Leisch [309]. Many other validation approaches exist (too many to list) and more continue to be developed. Briefly, we take a look at the validation options available in the R package `clValid` [64] by Brock et al., which includes some specific validation methods for biological data. The authors describe the options as falling into three main categories: internal, stability, or biological.

Our focus will be on the internal and stability methods. According to the package authors, "Internal validation measures take only the dataset and the clustering partition as input and use intrinsic information in the data to assess the quality of the clustering. The stability measures are a special version of internal measures. They evaluate the consistency of a clustering result by comparing it with the clusters obtained after each column is removed, one at a time" [64, pp. 2–3]. Here, the columns are the variables in the dataset, so the stability measures are evaluating how the clustering solutions change as variables are left out. The internal measures include connectedness, the silhouette width previously described, and the Dunn Index. Four stability measures are included. For details, please see Brock et al. [64], which has additional references as well. For all of the stability measures, smaller values are desired.

One of the benefits of using `clValid` is that it allows the user to choose from up to nine different clustering algorithms (assuming they are appropriate), select potential values of k and the desired validation approaches, and (after some processing time), it will report on which settings provide the best solution in terms of the validation measures. Several of the available clustering algorithms are discussed in later sections. To illustrate how `clValid` can be applied, we used the wine data, with inputs to the function as follows:

1. clustering algorithms: `agnes`, `kmeans`, `diana`, `pam`, `clara`, with Euclidean distance for all methods and average linkage for hierarchical methods,
2. possible values of k: 2–20,
3. validation measures: internal and stability for a total of 7 validation measures.

A summary of the results (not shown) found that k-means with 3 clusters was the best solution in terms of the silhouette coefficient, `pam` with 20 clusters was the best according to one of the stability measures, and the other 5 validation measures selected agnes (which is a hierarchical agglomerative method) with only 2 clusters as the optimal solution. clValid does allow the use of model-based methods via mclust, which are discussed below, though none were implemented in this example. However, at this time, it does not provide for the implementation of density-based methods that we discuss later. For details on recent work highlighting validation methods for density-based approaches, the reader is directed to Moulavi et al. [354].

For a different take on cluster validation, with a focus on evaluating clustering usefulness, rather than performance using some of the methods we've described or used in this chapter, the reader is directed to Guyon et al. [205] for some interesting discussion.

5.4.5 Other partitioning algorithms

While k-means is the most popular partitioning algorithm, it is not the only available partitioning algorithm. Where k-means focused on centroids (means) and minimizing squared error, one can swap to median-like quantities and use absolute error in a k-medians protocol [97]. One fairly popular alternative is the partitioning around medoids (PAM) algorithm [263]. The PAM algorithm is similar to k-means, but a key difference is that the "centers" of the clusters are medoids, which are representative objects from the clusters (i.e., actual data points), not centroids, which may not be actual data points. The PAM algorithm is less sensitive to outliers than k-means [243]. Further extensions of the PAM algorithm exist, such as CLARA (Clustering LARge Applications) [261], and both of these algorithms can be applied in most common software for data analysis, as we saw them both in our k-means validation example.

There is no one algorithm that is "best" in every situation. Each algorithm has advantages and disadvantages and has been designed such that they work better in some situations rather than others. Indeed, many methods exist that are not based on a partitioning approach, so let's explore other approaches.

5.5 Hierarchical clustering

Like k-means, hierarchical clustering methods are intuitive. In these methods, a hierarchical series of clustering solutions is developed by either slowly merging similar observations into clusters together until all observations are in one cluster (referred to as agglomerative or bottom-up clustering) or by continually splitting clusters down to the level of individual observations after starting from a single cluster (referred to as divisive or top-down clustering). Typically, splits and merges are limited to involve just two clusters (i.e., creating two from one, or merging two into one), though this is not always the case.

In each level of the hierarchy, a different number of clusters is present. One of the hierarchical levels is chosen as the clustering solution, based on various criteria we will discuss below. For example, a solution with 10 hierarchical levels may be obtained and the third level from the top be chosen because it has the best average silhouette width for its clusters among the 10 levels, or the fifth level from the top may be chosen because of the number of clusters it has.

Agglomerative methods are more common than divisive methods, so we will focus on agglomerative methods to develop our understanding. The challenge with these methods is that the distance measures are defined between observations, but we need to be able to merge clusters based on cluster distances to create these solutions. The methods used to update the cluster distances in the overall distance matrix are called the *linkage*s. There are several common linkages available in most clustering software programs. Let $d(C_a, C_b)$ be the distance between cluster C_a and C_b, which each could be made up of different numbers of observations. The linkages tell us how to update these distances as we merge clusters together, regardless of what distance measure we have chosen to use between observations.

5.5.1 Linkages

Here, we describe a few linkage types, commonly available in most software packages that perform hierarchical clustering.

1. Single Linkage: The distance between any two clusters is the minimum distance between any pair of observations in the different clusters.

$$d(C_a, C_b) = \min d(x_i, x_j) \text{ for } x_i \in C_a \text{ and } x_j \in C_b \tag{5.1}$$

2. Complete Linkage: The distance between any two clusters is the maximum distance between any pair of observations in the different clusters.

$$d(C_a, C_b) = \max d(x_i, x_j) \text{ for } x_i \in C_a \text{ and } x_j \in C_b \tag{5.2}$$

3. Average Linkage: It may come as no surprise that there is a compromise option between single and complete linkage, due to their use of extremes. In average linkage, the distance between any two clusters is the average of all the pairwise distances between all pairs of observations in the different clusters.

$$d(C_a, C_b) = \text{mean } d(x_i, x_j) \text{ for } x_i \in C_a \text{ and } x_j \in C_b \tag{5.3}$$

4. Centroid Linkage: The challenge with average linkage is that it requires looking at all pairwise distances between the observations in the two clusters under consideration. Centroid linkage is simply the distance between the means (centroids) of the two clusters.

5. Ward's Method: Based on the work of Joe Ward [491], while not strictly a way of defining distances between two clusters, it is a commonly available method in software. At each level of the hierarchy, a statistic called the error sum of squares is computed for the clusters in the level. The error sum of squares is designed to capture the loss in information from merging points into clusters compared to keeping them all as individual units. Thus, smaller values of the statistic are desired. When moving up to the next level, and determining which two clusters to merge, the error sum of squares statistic is computed for each possible new cluster arrangement at that level. The arrangement that has the smallest increase in the error sum of squares is chosen, and the relevant merge performed. This method has been studied over the years but is implemented differently in various software packages. Please see Murtagh and Legendre [355] for details on selected implementations of this method.

The various linkages have different advantages and disadvantages related to their use and behavior. For example, single linkage often exhibits a behavior called "chaining" where two clusters separated by a few points are strung together. Also in single linkage, it is not uncommon for observations to merge at higher distances to a main cluster, one at a time. This behavior can be useful for identifying some unusual points, but may yield unrealistic clustering solutions (a large number of clusters with one or two observations each), or due to outliers, it may join two clusters which have a majority of points rather far apart. However, single linkage is capable of handling diverse cluster shapes, though the sensitivity to outliers can often outweigh its usefulness.

At the other end of the spectrum, complete linkage tends to break large clusters, creating many small clusters, and may generate spherical clusters, which may not be realistic. Ward's linkage also suffers from a preference for spherical clusters [97]. Centroid linkage may improve on average linkage in some applications, but can also suffer from inversions. As an example of an inversion, suppose we have two disjoint clusters at some level in the hierarchy. An inversion would occur if our process results in their union being lower in the hierarchy than the disjoint clusters themselves. Our plan is that we are going from many clusters to a single one, so this should be impossible. Inversions are relatively rare, but can result based on the choice of linkage and the dataset [247].

For a detailed comparison of k-means and some common linkages for hierarchical clustering, please see Fisher and Ness [160].

5.5.2 Algorithm

The agglomerative hierarchical clustering algorithm can be summarized in the following steps:

1. Select a distance measure, and then obtain the starting distance matrix for all observations.

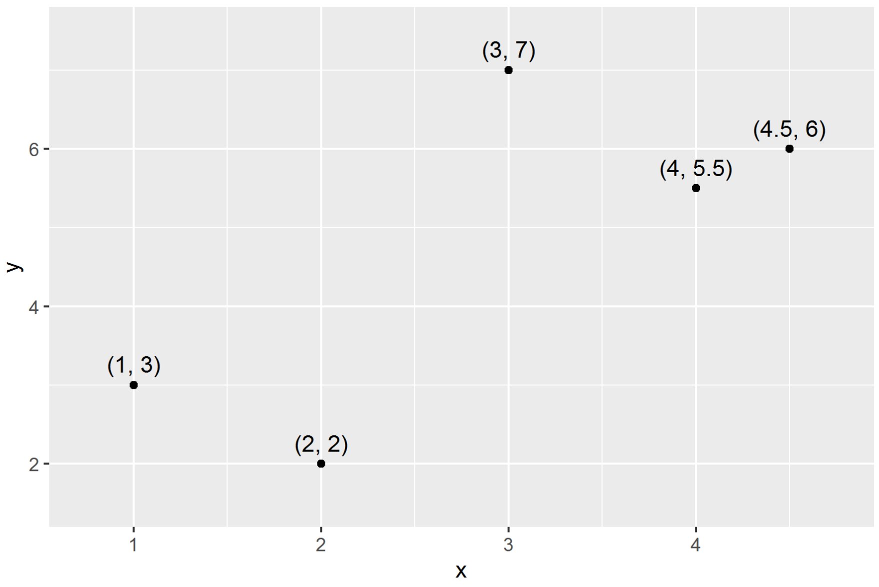


FIGURE 5.8: Five sample data points.

TABLE 5.7: The Euclidean distance matrix for the five data points shown in Figure 5.8.

Obs	1	2	3	4	5
1	0				
2	4.03	0			
3	0.71	4.72	0		
4	1.80	5.10	1.80	0	
5	3.91	1.41	4.61	4.47	0

2. Merge the two observations (or clusters) with the smallest distance (performing multiple merges in the case of ties) to form a new cluster (or clusters, in the event of ties).

3. Update the distance matrix with the new cluster distances based on the linkage selected.

4. Repeat the previous two steps (merge and update) until all observations are merged into a single cluster.

5.5.3 Hierarchical simple example

For this example, we will assume we have five bivariate observations that have been selected for clustering as shown in Figure 5.8. The associated starting Euclidean distance matrix is provided to us as shown in Table 5.7.

TABLE 5.8: Distance matrix from Table 5.7, but with clusters 1 and 3 merged. The distances for this new cluster have not yet been computed.

Cluster	1,3	2	4	5
1,3	0			
2	?	0		
4	?	5.10	0	
5	?	1.41	4.47	0

TABLE 5.9: Distance matrix from Table 5.8, but with missing entries computed using single linkage.

Cluster	1,3	2	4	5
1,3	0			
2	4.03	0		
4	1.80	5.10	0	
5	3.91	1.41	4.47	0

Let's find the agglomerative clustering solution for these five observations based on single linkage. To begin, we find the minimum distance in the distance matrix, which will indicate which two observations we merge to create a new cluster. The merge is always based on smallest distance; the linkage just tells us how to update distances after the merge is complete (except for Ward's method). In the event of ties, multiple merges (creating multiple clusters, or a cluster with three or more observations) may occur.

In our matrix, we can see that observations 1 and 3 have a distance of 0.71, and that is the minimum distance in the matrix. We merge these observations to create a new cluster called $C_{1,3}$. Note that this will leave a number of distances unchanged; for example, the distance between observations 2 and 4 is unaffected by this merge. We can see what the updated distance matrix looks like in Table 5.8. The question marks denote all the distances that need to be updated based on the merge of observations 1 and 3 into a cluster.

To fill in the question marks in Table 5.8, we use single linkage. We note that $d(x_1, x_2) = 4.03$ and $d(x_3, x_2) = 4.72$, so $d(x_2, C_{1,3})$ is the minimum of those, or 4.03. We can fill in the other entries similarly, as shown in Table 5.9.

We continue the merging process based on the smallest distance between observations or clusters, which happens to be between observations 2 and 5, at a distance of 1.41. The distance matrix is updated using single linkage for all distances that involve the new cluster, and the process continues. To summarize what happens in this example, we find that observations 1 and 3 join at a distance of 0.71, observations 2 and 5 join at 1.41, observation 4 joins 1 and 3 at 1.80, and all observations are finally joined together at a distance of 3.91 into a single cluster.

5.5.4 Dendrograms and wine example

For hierarchical clustering, a common graphical representation of the solution is a dendrogram. The dendrogram is designed to show (at least for

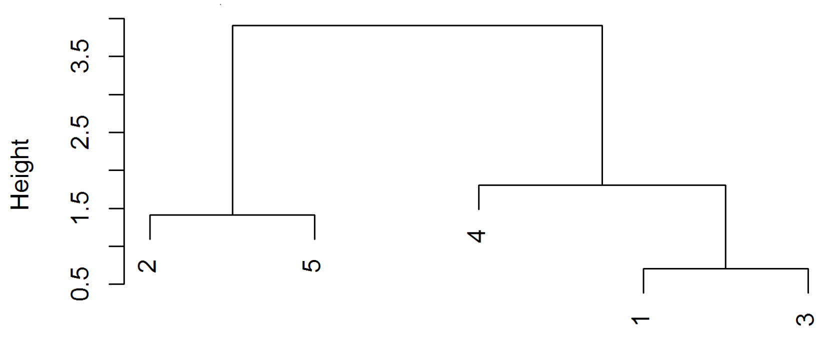


FIGURE 5.9: Dendrogram from single linkage agglomerative hierarchical clustering of 5 sample data points.

relatively small numbers of data points) which clusters merged (or split) and at what dissimilarities (the height on the y-axis) the merges (or splits) occurred. The plot thus shows a collection of possible clustering solutions, and an individual solution is obtained by "cutting" the dendrogram at a particular height. In Figure 5.9, we see the dendrogram that resulted from our single linkage hierarchical clustering example. The distances at which each cluster joined are all visible in the dendrogram.

In terms of obtaining a particular clustering solution, cutting the dendrogram at any height less than 0.71 would result in 5 distinct clusters, and cutting the dendrogram at heights greater than 3.91 yields a single cluster. Using a height of 1.75 to make the cut would result in three clusters (observations 1 and 3, observations 2 and 5, and observation 4), since this is just before observation 4 joins observations 1 and 3. Typically, a judgment call must be made about where to make the cut, which is often based on distance between merges or the number of clusters desired or both. For example, if there is a large gap in distance before a merge perhaps the clusters existing before the merge are really separate clusters. This might be a good solution to examine. Another approach is to define an error criterion and plot the value of the error criterion versus k as suggested before with k-means. As before, an "elbow" in the plot can indicate a reasonable value for k.

For our simple five-point example, the solution under both single and complete linkage turns out to be the same. In order to demonstrate differences in the linkages, we turn to our wine dataset. Recall there are 178 observations with 13 measurement variables. As before, due to the scaling issues between the measurement variables, we standardize the variables and will use Euclidean distance just as we did in our k-means example. The dendrograms for the solutions using both single and complete linkage are shown in Figures 5.10 and 5.11. The observation numbers are shown but are very small with 178 observations. However, the general structure of the dendrograms shows that there are differences between the single and complete linkage types.

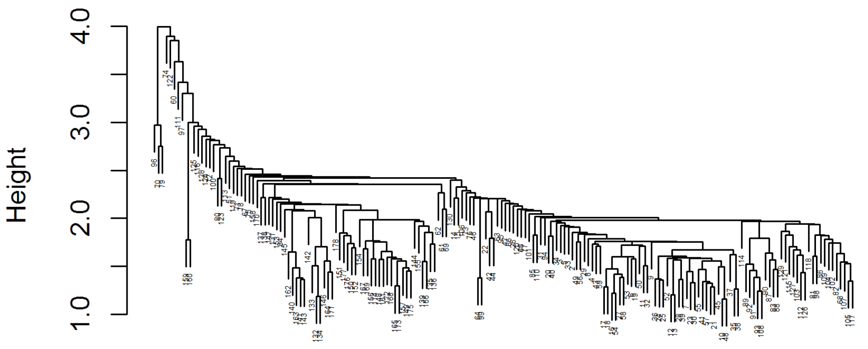


FIGURE 5.10: Dendrogram from single linkage agglomerative hierarchical clustering of wine data.

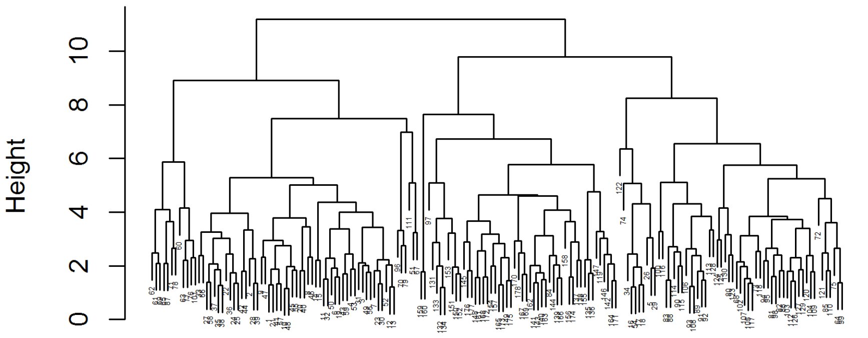


FIGURE 5.11: Dendrogram from complete linkage agglomerative hierarchical clustering of wine data.

5.5.5 Other hierarchical algorithms

The agglomerative algorithm discussed above (and implemented via `hclust` or `agnes`—agglomerative nesting—in R) is not the only algorithm available for hierarchical clustering. Divisive methods can also obtain a hierarchical solution, and an example of such an algorithm is `diana` in R. This algorithm was originally published in *Nature* in 1964 [329]. The method relies on a series of binary splits to separate the observations into clusters, starting from all in a single cluster and slowly splitting the observations apart based on dissimilarities between clusters. Another alternative is the algorithm `mona`, also available in R [262].

5.6 Case study

For our case study examining k-means and hierarchical methods, we will use the College Scorecard dataset, available from the United States Department of Education [367]. We downloaded the most recent institution-level data file, which gave us a dataset of 7112 observations and 1978 variables. A data dictionary that describes the variables and how they are coded is also available [367]. The dataset contains information on colleges and universities in the United States, as part of an initiative to help prospective students understand their college options, especially regarding costs. Different institutions submit different amounts of information to the Department of Education, so there is a high level of missing data.

How can we make sense of all this information about the different institutions in the United States? Prospective students, parents, and guardians might search for schools with tuition in a certain range, or schools located in their home state, or schools with good retention rates. However, this is only considering one variable of interest at a time. We could combine these and filter for several characteristics, but this still relies on our knowledge of what types of schools we want to find. Instead, we can use clustering to look for natural groups of similar institutions, which we can then investigate further. Perhaps finding that one institution is in a particular cluster will lead us to consider other institutions in the same cluster. For our case study, we will look to see what clusters of institutions are recovered, with no particular expectations about what will be found.

For our analysis, we initially isolated 20 quantitative variables we wanted to include in the clustering. However, that gave us fewer than 1200 observations remaining with complete cases (no missing data). We investigated our subset of 20 variables and isolated 9 that we could retain and still have 4546 observations (from the original 7112). We also kept variables that indicated the school name, type of institution (e.g., public versus private), and region in which the school is located, even though those variables will not be used

TABLE 5.10: Nine variables selected from the College Scorecard dataset and the definition of each.

Variable	Description
UGDS	Enrollment of undergraduate certificate/degree-seeking students
TUITFTE	Net tuition revenue per full-time equivalent student
PCTPELL	Percentage of undergraduates who receive a Pell Grant
COMP_ORIG_YR4_RT	Percent completed within 4 years at original institution
COMP_ORIG_YR6_RT	Percent completed within 6 years at original institution
PAR_ED_PCT_1STGEN	Percentage first-generation students
GRAD_DEBT_MDN	The median debt for students who have completed
AGE_ENTRY	Average age of entry
MD_FAMINC	Median family income in real 2015 dollars

in the clustering. The 9 variables we used in our clustering and their descriptions (from the Department of Education [367]) are shown in Table 5.10. The 9 variables are clearly on different scales, so we will standardize them for our clustering.

5.6.1 k-means results

As in our wine data example, we examined k-means solutions and computed the within-group sum of squares in order to choose an appropriate k. Here, we implemented the process for $k = 2$ to $k = 20$ (slightly more clusters). The results are shown in Figure 5.12. There are several potential elbows indicating we might want a k between 3 and 6. Since an elbow was not clear, we computed the average silhouette width for k values from 2 to 100. This revealed that the silhouette coefficient, the maximum average silhouette width, was obtained with the $k = 4$ solution, with a value of 0.3428.

We took a closer look at the four-cluster k-means solution. The clusters had 739, 1559, 1203, and 1045 observations respectively. The cluster centroids (on the standardized variables) are displayed in Table 5.11. Briefly looking

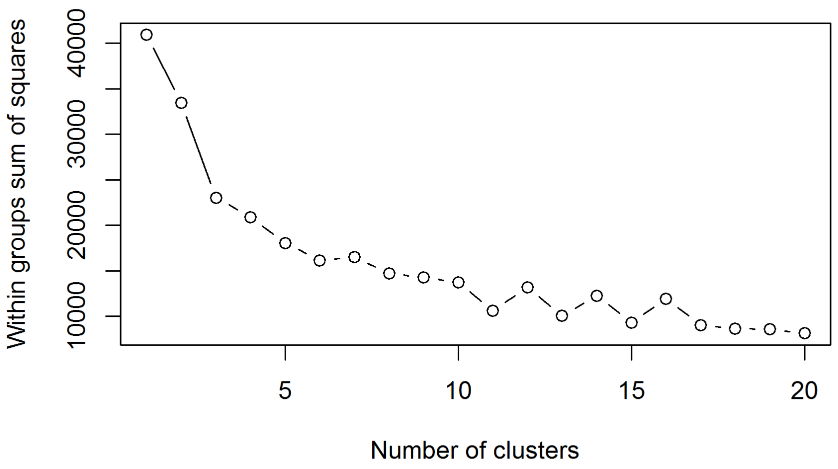


FIGURE 5.12: Within-groups sum of squares for k-means solutions with $k = 1$ to $k = 20$ clusters for the College Scorecard data.

over the cluster centroids, it appears that clusters 1 and 3 had similar net tuition revenue per student, but differed greatly on retention rates (both 4 and 6 years), average age of entry, and median family income. Cluster 4 has the lowest net tuition revenue per student, the lowest retention rates, but moderately low debt at graduation. Finally, cluster 2 is distinguished by having the smallest number of students, highest retention rates, and lowest debt at graduation.

To determine how strong the clustering structure found here is, we examined the average silhouette width for each cluster. The values were 0.2041, 0.4200, 0.3270, and 0.3439, respectively. These all suggest the structure found

TABLE 5.11: Cluster centroids for the four-cluster solution for the College Scorecard dataset.

Variable vs. Centroid	Cluster 1	Cluster 2	Cluster 3	Cluster 4
UGDS	-0.31	-0.46	0.31	0.55
TUITFTE	0.35	-0.03	0.33	-0.58
PCTPELL	0.63	0.69	-0.85	-0.51
COMP_ORIG_YR4_RT	-0.87	0.82	0.44	-1.11
COMP_ORIG_YR6_RT	-0.76	0.73	0.51	-1.15
PAR_ED_PCT_1STGEN	0.41	0.70	-1.24	-0.10
GRAD_DEBT_MDN	1.12	-0.78	0.84	-0.59
AGE_ENTRY	0.94	0.50	-1.01	-0.25
MD_FAMINC	-0.47	-0.63	1.35	-0.29

TABLE 5.12: Comparison of the four-cluster solution for the College Scorecard dataset with the control variable of institution type.

Control versus k-means Clusters	1	2	3	4
1 - Public	44	98	399	963
2 - Private nonprofit	255	131	773	71
3 - Private for-profit	440	1330	31	11

is weak and could be artificial, or, for cluster 1, that the structure is not substantial. Unfortunately, graphing the solution in principal component space is not very revealing. The clusters do appear to separate along the first principal component axis, but there is an extreme outlier very far from the other points (visual not shown). Additionally, it is difficult to gain much meaning from a plot with over 4000 data points on it. Removing the outlier, which is in cluster 1, does not look like it would affect the solution much.

Instead of trying to gain visual insights into our clusters, we decided to see how the clusters found corresponded to the type of institution and school's region. For region, we didn't find much of an association with the four-cluster k-means solution. However, for the type of institution (Control variable: 1 = Public, 2 = Private nonprofit, and 3 = Private for-profit), we found some association, as shown in Table 5.12. Comparing these associations to the cluster centroids is interesting.

5.6.2 Hierarchical results

To find a clustering solution to compare to the k-means four-cluster solution, we examined a variety of hierarchical models with `hclust`. Interestingly, both single linkage and complete linkage identified one institution as an extreme outlier. It joined all the other observations very late in the associated dendrograms. This institution was identified as "Ultimate Medical Academy – Clearwater." This outlier turns out to be the same one visible in the principal component space. Keeping this outlier in the analysis meant that neither single nor complete linkage appeared to be good options for a hierarchical clustering solution. Instead, we performed hierarchical clustering using Ward's method, and this resulted in a dendrogram that seemed easier to work with, as shown in Figure 5.13. Note that the merges at low distances are impossible to read due to the sheer number of data points, but it is fairly easy to identify heights at which to cut the dendrogram to recover anywhere between two and seven clusters. We decided to explore the five-cluster solution.

The overall average silhouette width for the five-cluster solution was 0.2612, indicating a weaker structure than that found via k-means. The clusters had 939, 543, 713, 715, and 1636 observations each with average cluster silhouette widths of 0.1721, 0.2106, 0.2186, 0.3729, and 0.2990 respectively. This suggests only the fourth and fifth clusters recovered any real structure, and it was weak. Briefly, we compare the hierarchical solution to the k-means solution, because it appears there isn't much else to learn from the hierarchical

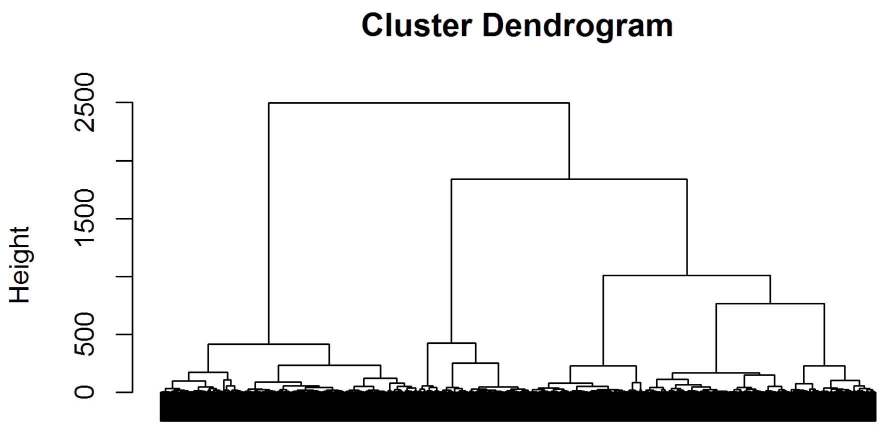


FIGURE 5.13: Dendrogram from Ward's method hierarchical clustering of College Scorecard data.

TABLE 5.13: Comparison of the four-cluster k-means solution with the five-cluster hierarchical solution on the College Scorecard dataset.

Hierarchical versus k-means Clusters	1	2	3	4
1	206	34	425	279
2	464	63	12	4
3	0	1	694	18
4	6	0	0	709
5	63	1461	72	40

solution. The overlap between the clustering solutions is shown in Table 5.13. We see that cluster 2 from k-means overlaps a great deal with cluster 5 in the hierarchical solution. In particular, it loses fewer than 100 observations to other hierarchical clusters and picks up less than 200 observations from other k-means clusters. Similarly, cluster 4 from k-means overlaps significantly with cluster 4 in the hierarchical solution (numbers match by coincidence), though over a quarter of its observations ended up in hierarchical cluster 1. However, the other clusters are different, resulting in the differences in average silhouette widths. With the k-means solution preferred here, we can consider next steps.

5.6.3 Case study conclusions

Natural next steps in this analysis would be to explore the k-means cluster centroids further, and try to see if there is a "name" that can be given to each cluster. We can return to our brief summaries of the k-means clusters to try this ourselves. We also looked at names of institutions in each cluster to try to validate our descriptions. Generally speaking, cluster 4 appears to be "big" public schools, including a large number of community colleges. These schools have large numbers of undergraduates coupled with low net tuition revenue and poor retention rates compared to the other institutions. This makes sense given the status of community college in the United States. Cluster 2 appears to be largely small, private, for-profit schools which have high retention rates and low debt upon graduation—perhaps these are technical schools. Checking names of some schools in this cluster, we do indeed find a large number of technical schools, including many beauty schools, in the cluster.

Cluster 3 is largely a mix of public and private nonprofit schools with fairly traditional students in terms of age (lowest ages at entry), fairly high tuition revenue per student, and relatively well-off families sending students there (judging by the median family income). As examples of schools included in this cluster we find the University of Southern California, Yale University, Emory University, and the University of Notre Dame, along with a number of liberal arts schools such as Amherst College, Smith College, Wellesley College, and Williams College. These appear to be more traditional four-year institutions rather than community colleges. Finally, considering cluster 1, we see schools that look in many respects like those in cluster 3, but with older students, comparatively poor retention rates, and lower family median income. It's harder to find a pattern here to give a name to the cluster (perhaps this is reflected in the cluster having the lowest average silhouette value among these four). There appear to be few community colleges, with a mix of technical and institutions with four-year programs in this cluster.

Finding these divisions is not that surprising. However, the clustering results might be of interest to audiences beyond students and their families. Perhaps a company wants to market a new educational software tool and target it at schools with the most students (i.e., cluster 4). Services for non-traditional students might be advertised for schools in cluster 1. Companies may want to advertise unpaid internships to the more traditional students in cluster 3. Obviously, we are making some broad assumptions about institutions in the clusters when we assign names or suggest these applications, but the clusters give us a starting place to work from.

After exploring the clustering solution we chose, we could consider further refinements to the clustering solution. Perhaps the outlier should be removed and the set of variables involved be adjusted. The choice of variables can greatly affect the solution, and here, recall that we originally selected 20 out of the 1978 available, and reduced that down even further to just 9 variables due to issues with missing data. Perhaps less missing data is present with slightly older versions of the dataset; we could investigate older releases, or find other sources of information to merge.

We could filter down to certain observations to cluster on as well. For example, what if we looked for clusters only among the private four-year institutions? What clusters would we find in the Rocky Mountains? We could find completely different clusters with an altered focus. If you want to know which of these clusters your undergraduate institution or your local college ended up in, we suggest downloading the dataset, rolling up your sleeves to get the data in an appropriate software program, and trying clustering out on your own.

5.7 Model-based methods

While partitioning methods, like k-means, and hierarchical methods can be written as algorithms to follow, they do not have a strong mathematical or statistical foundation. Model-based clustering methods have a stronger foundation, though they can still be considered partitioning methods. In these methods, assumptions are made about the underlying clusters to be discovered that enable statistical models to be specified and then fit based on the data in order to uncover the clusters.

The idea of fitting models means that we can choose among models (i.e., perform model selection) to find a good fit to the data. However, in order to fit such models, one makes a strong assumption that the underlying population is actually a collection of subpopulations (the clusters), and that they have multivariate probability density functions that enable them to be distinguishable from each other. With a fixed (finite) number of subpopulations, the result is a finite mixture density because the population is a mixture of the subpopulations (which can be in different proportions relative to each other). This model means that the challenge lies in estimating the components of the mixture, and their relative proportions based on assigning observations to the cluster they most likely belong to based on the model fit [148]. Mixture models have their origins in the work of Wolfe, for example [512]. We discuss model development and estimation techniques next.

5.7.1 Model development

As before, assume we have k clusters present in the population (and our data). Then, in a finite mixture model, each cluster is associated with a multivariate probability density function, denoted $g_j(x)$, where $j = 1, \ldots, k$, and x is our p-dimensional random vector of values for an observation. In other words, $g_j(x)$ is the probability density of $x \in C_j$. The proportion of observations in each cluster is given by q_j. We combine all of the individual component densities together to obtain the population density function. In other words, the population density function $f(x \mid q)$ is given by

$$f(x \mid q) = \sum_{j=1}^{k} q_j g_j(x).$$

The density functions $g_j(x)$ are usually governed by parameters, specific to each cluster, suppressed in the notation here, though they can be added. For example, the density function might be written as $g_j(x \mid \theta_j)$, and the population density function as

$$f(x \mid q, \theta) = \sum_{j=1}^{k} q_j g_j(x \mid \theta_j).$$

With the goal of estimating the cluster probability density functions (or, more specifically, their parameters) and the mixing proportions, we note that one necessary constraint in the solution is that $\sum_{j=1}^{k} q_j = 1$, and we assume that the mixing proportions are non-negative.

When working with these models, a common assumption made (which simplifies the problem) is that the cluster probability density functions all come from the same family. Most commonly, they are assumed to be multivariate normal, such that the mixture distributions arising here are Gaussian mixtures. In that case, the parameters θ_j, $j = 1, \ldots, k$, refer to different means and covariances for multivariate normal distributions. When attempting to find estimates for these parameters, with the assumption of multivariate normality, we will see that some constraints may be made on the covariances to make the optimization more feasible. It should be noted that the model itself does not expressly require the cluster density functions to be in the same family; however, that assumption significantly simplifies the estimation process.

5.7.2 Model estimation

In order to find estimates for our model parameters, we will need to specify a value of k in order to know how many components are in our mixture. The true values of k, q, and θ are unknown, so let's assume we choose an arbitrary k for now, and we want to estimate q and θ based on that choice. The estimates we will find are typically denoted $\hat{q}$ and $\hat{\theta}$.

How will we solve for $\hat{q}$ and $\hat{\theta}$? A classical statistical approach to estimate parameters is via maximum likelihood estimation. However, even simple examples using a mixture of two univariate normal distributions show serious challenges for maximum likelihood parameter estimation. If the maximum likelihood estimation approach is not feasible, what can we do instead? A modern approach is to use the Expectation-Maximization (EM) algorithm [126]. This algorithm is an iterative process, alternating between an expectation and maximization step. We can motivate this idea in the context of the mixture of two univariate normal distributions, following from Clarke et al. [97].

In this setting, we suppose $p = 1$ and $k = 2$, with both underlying densities being normal. Then we have that $g_j(x) = \phi(x \mid \theta_j) = \phi(x \mid \mu_j, \sigma_j^2)$ for $j = 1, 2$, where ϕ is the usual normal density function. When we combine the g_j's to obtain the mixture density, we obtain

$$f(x) = q_1 \phi(x \mid \mu_1, \sigma_1^2) + (1 - q_1)\phi(x \mid \mu_2, \sigma_2^2).$$

With k of only 2, this is showing that the observation is drawn from one of two normal distributions, with probability q_1 of being from the first distribution, and hence, probability $q_2 = 1 - q_1$ of being from the second. Note that in this case $\theta = (\mu_1, \sigma_1^2, \mu_2, \sigma_2^2)'$, and we need only an estimate of q to get both mixing proportions.

If we consider the observed data as $x_1, \ldots, x_n$, since we don't know which density each observation came from, it is like we are missing a corresponding $y_i = 1$ or 2 for each x_i that would indicate which density the observation was drawn from. In other words, we could consider our data to be pairs $(x_1, y_1), \ldots, (x_n, y_n)$, but the y_i's are missing. If the y_i's were known, it would be easy to obtain an estimate for q_1 using the sample proportion of y_i's equal to 1, and sample means and standard deviations for the the estimate of θ. Unfortunately, we don't have the y_i's. This is precisely the situation that the EM algorithm is designed for. The goal is to reconstruct the y_i's (or a similar construct that conveys what information is missing), using that information to optimize the remaining parameters.

The algorithm needs initial starting values for the estimates of θ and q. With no prior information, in this example, we could set the means to be different sample values, use the sample standard deviation for both sigmas, and set $\hat{q}_1 = 0.5$, assuming the mixtures are in equal proportions. A variety of other choices are also possible [97]. Then, we start alternating the expectation (E) and maximization (M) steps of the algorithm. We start with the E step, whose goal is to estimate the missing portion of the data by finding the conditional expectation for the missing portion given the observed data, current $\hat{q}$, and current $\hat{\theta}$. This "fills in" the missing portion and allows us to proceed to the M step.

In the M step, we use the "filled-in" missing data values to maximize the log-likelihood (which in this case is the natural log of the mixture density above), and update our estimates of q and θ. The two steps then iterate—back to the E step, then the M step, etc.—until some convergence or stopping criterion is met. The criterion could be a set number of iterations or that the change in estimates of q and θ falls below a certain small threshold value (measured by some metric). For details in this example, please see Clarke et al. [97].

The EM algorithm has applications far beyond this example. For some insights into the EM algorithm itself, see Dempster et al. [126], for details about the general case (for non-normal densities) in the mixture model context, see Borman [57], and for more derivations in the normal mixture case for k components, see Clarke et al. [97]. Bayesian analysts might use Markov Chain Monte Carlo-based (MCMC) procedures such as a Gibbs sampler for the same optimization [148]. Indeed, one might pursue Bayesian clustering techniques or more general non-parametric clustering techniques; see Quintana [396] for some insights.

Using the EM algorithm, we can find the estimates of q and θ that work best with our choice of k. But how do we know that choice of k was a good one? We need a metric to evaluate the solutions. A common one (which is incorporated in the example we show below) is the Bayesian Information Criterion (BIC) [168]. We will find estimates of q and θ for several values of k, i.e., fit multiple mixture models, and then compare their values of the evaluation measure, BIC. The BIC is based on optimizing the log-likelihood of the data

(x) given the parameters. The BIC for a solution with k clusters is computed as:

$$BIC(k) = -2\log f(x \mid \hat{q}, \hat{\theta}) + k' \log(n),$$

where k' is the number of parameters being estimated. The choice of k drives the value of k', depending on the underlying distributional choices and their respective parameters in θ.

5.7.3 `mclust` and model selection

A popular package for fitting normal mixture models in R is `mclust` [167], which was created based on work by Fraley and Raftery [168]. It uses the EM algorithm approach for estimating the model parameters, and assumes the clusters are all ellipsoidal (including spherical). While the mean vectors for the clusters are unconstrained, `mclust` has a family of models that impose constraints on the covariance matrices (defined in Section 4.7.1). The user could choose one of these models, or use criteria to select one after fitting them all. To understand the models, we first have to understand how the covariance matrices are represented.

The covariance matrix Σ_j for the j^{th} cluster is represented by its familiar eigendecomposition, as follows. (Here we use the notation from [167], to be consistent with their model presentation that follows, but the reader may also refer to Section 3.3.1 for a review of the eigendecomposition, or eigenvalue decomposition.)

$$\Sigma_j = \lambda_j D_j A_j D_j^T$$

In the decomposition, the various components affect the covariance matrix differently. A_j is diagonal, with elements that are proportional to the eigenvalues, and this matrix determines the shape of the density contours for the cluster. On the other hand, D_j is the usual orthogonal matrix of eigenvectors, and it determines the orientation of the principal components for the cluster. Finally, λ_j is a scalar, and relates to the volume of the cluster. In `mclust`, these are referred to as the shape, orientation, and volume aspects of the clusters, respectively. As originally proposed, the `mclust` models allow for each of these aspects to be constrained to be the same for all clusters, forced to be the identity matrix, or allowed to vary (increasing the number of parameters that must be estimated) [167]. In Table 5.14, we show the characteristics and name of each model in `mclust`; a version of this table appears in Fraley and Raftery 1999 [167] and another version in Everitt and Hothorn [148].

We will demonstrate the use of `mclust` in our next example.

5.7.4 Example with wine data

To demonstrate model-based clustering, we fit a normal mixture model to the wine data previously described. The 13 measurement variables are all used (again, not including cultivar) for all 178 observations. The `mclust` package in R is used for the model fit [167], and a BIC approach taken to determine

TABLE 5.14: The models supported by R's `mclust` package and the characteristics of each.

Name	Model	Volume	Shape	Orientation
EII	λI	equal	identity	identity
VII	$\lambda_j I$	vary	identity	identity
EEI	λA	equal	equal	identity
VEI	$\lambda_j A$	vary	equal	identity
EVI	λA_j	equal	vary	identity
VVI	$\lambda_j A_j$	vary	vary	identity
EEE	λDAD^T	equal	equal	equal
EVE	λDA_jD^T	equal	vary	equal
VEE	$\lambda_j DAD^T$	vary	equal	equal
VVE	$\lambda_j DA_jD^T$	vary	vary	equal
EEV	$\lambda D_jAD_j^T$	equal	equal	vary
VEV	$\lambda_j D_jAD_j^T$	vary	equal	vary
EVV	$\lambda D_jA_jD_j^T$	equal	vary	vary
VVV	$\lambda_j D_jA_jD_j^T$	vary	vary	vary

the optimal model. The plot of the solution shown in Figure 5.14 allows us to compare the different models and their performance on the dataset. Note that Figure 5.14 is not the default plot from R, which would show all considered models and their BIC values. Here, only the best models are shown for easier comparison at selected values of k, the number of components in the mixture.

Our analysis finds that many of the models are performing similarly well in terms of optimizing the BIC criterion. A summary command can reveal the "best," which in this case is the VVE model with three clusters, which means that the volume and shape parameters may vary across clusters but equal matrices for all clusters are assumed for the matrix of orthogonal eigenvectors, D. The solution found 3 clusters, with 59, 69, and 50 observations, for mixing probabilities of 0.33, 0.39, and 0.28, respectively. The model fit also returns the cluster means and variances, and offers many plotting options to view the solution, including one that makes a scatterplot matrix colored by cluster. However, since there are 13 measurement variables here, such a plot is difficult to read, so instead we plot the clusters found in the principal component space as we did previously for k-means. The result in Figure 5.15 shows that the clusters found are very similar to what we observed previously with `kmeans` and `hclust`.

As with k-means, we observe that the model-based clusters recover the cultivars remarkably well, even though the cultivar variable was not used to find the clusters.

5.7.5 Model-based versus k-means

In our wine data example, we noted that the clustering solution provided by `mclust` was similar to that provided by `kmeans`. So, why do we need

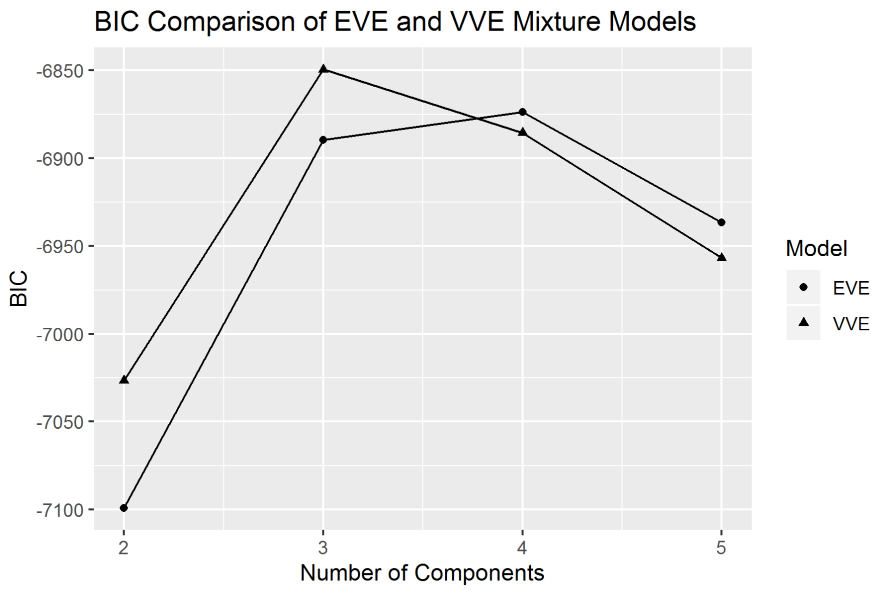


FIGURE 5.14: Snapshot of model comparison for normal mixture models on wine data showing best models.

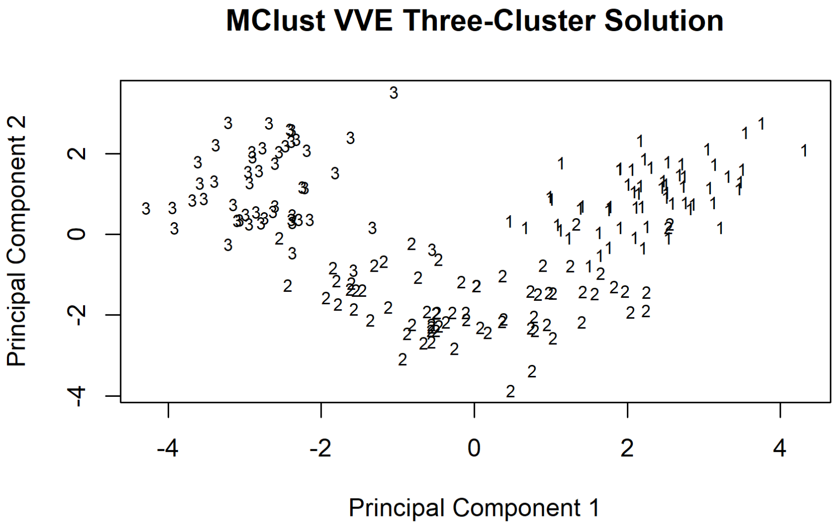


FIGURE 5.15: Three-cluster solution via `mclust` model VVE for wine data shown in principal component space.

model-based methods? One of the major motivators of model-based methods is simply that there is a statistical model fit to the data. This allows for the application of more formal model selection methods, such as the BIC, rather than heuristics, such as looking for an elbow in a within-group sum of squares plot or cutting a dendrogram at a distance that looks appropriate.

Fraley and Raftery [168] compare k-means and model-based methods via `mclust` using the Wisconsin Breast Cancer dataset [333], which is available via the UCI machine learning repository. We reexamine this comparison, noting that the software has undergone various updates since the original example was published, such that we get similar, but not exactly the same, results. The dataset contains information on 569 observations of cell nuclei from biopsy samples being analyzed to determine if they are malignant or benign (the diagnosis). Of the variables in the dataset, the ones selected for analysis are extreme area, extreme smoothness, and mean texture. (See Mangasarian et al. [333] for details on how these variables were selected.) Extreme area is the largest area of any cell nuclei in the sample, extreme smoothness is the largest value for the local variation in cell nuclei radii for the cells in the sample, and mean texture is the mean of the texture values (related to gray-scale values) of the cells. We implement both k-means and `mclust` on these three variables and study the relationship of the clustering solutions, as well as their relationship to the known diagnosis variable.

For k-means, we again try to select k using an elbow approach. The variables are on different scales, so the solutions were obtained for the standardized variables. We can fit $k = 2$ or $k = 3$ to see how those solutions match the diagnosis variable, but the plot (Figure 5.16) does not show a clear elbow here.

The k-means two-cluster solution matches the diagnosis variable fairly well, with only 53 observations not matching the majority class of their cluster. A three-cluster k-means solution finds one cluster of malignant samples (96 malignant, 0 benign), one that is predominantly benign (286 benign, 28 malignant), and one cluster that is a mixture of both (71 benign, 88 malignant). Can model-based clustering do any better?

Using `mclust`, the original reference states that an unconstrained model—denoted VVV in our notation—with three clusters was the best, though they also consider the VVV two-cluster solution [168]. For our analysis, we ran `mclust` to let it provide us with the summary comparing the methods and determine which was best using the BIC. The results indicate that the VVI model with either three or four clusters or the VVE model with three clusters has the lowest BIC (values within 9 of each other). The VVV three-cluster model is the 6th best model, with a BIC difference of about 26 from the best model, VVI with three clusters, so it is still a top competitor. We examine the VVI model with two and three clusters.

From the two-cluster VVI model, we see 50 observations do not match the majority class of their cluster. This is similar though slightly better than the k-means solution if we are interested in recovering the diagnosis variable. However, the three-cluster VVI solution shows us the major benefit of `mclust`.

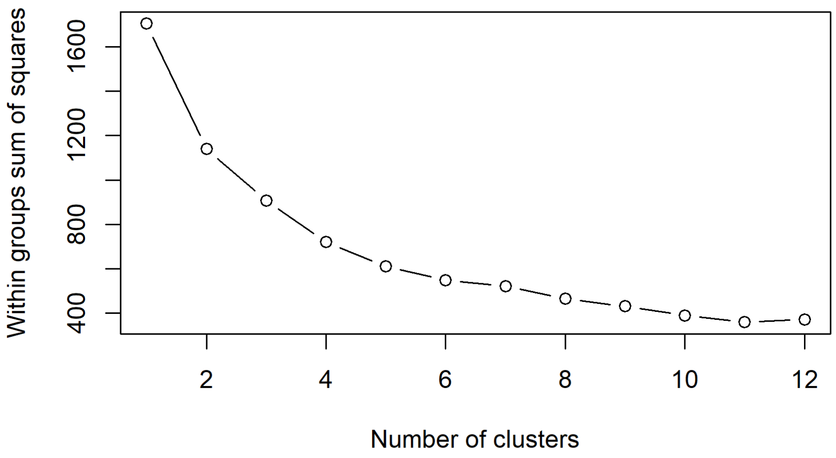


FIGURE 5.16: Within-groups sum of squares for k-means solutions with $k = 1$ to $k = 12$ clusters for Wisconsin Breast Cancer dataset.

This solution finds one cluster of malignant samples (143 malignant, 1 benign), one that is predominantly benign (291 benign, 5 malignant), and one cluster that is a mixture of both (65 benign, 64 malignant). The model-based method provides a much better separation of the malignant and benign clusters, and identifies a smaller intermediate cluster. New samples could be classified into these clusters, and samples in the intermediate cluster could be sent for further analysis [168].

Thus, model-based methods have proven useful in some contexts. Next, we'll briefly examine density-based methods, which are related methods that are quite popular in some fields, but have less of a model framework.

5.8 Density-based methods

Density-based methods approach clustering as a problem of looking for regions in (potentially high-dimensional) space where data points are dense (the clusters) separated by more sparsely populated space. One of the most popular methods of this type is density-based spatial clustering of applications with noise, or DBSCAN [147], and variants of it that have arisen since it was first proposed. To develop our intuition about these methods, we will use the framework provided in the original DBSCAN paper by Ester et al. [147].

The motivations for DBSCAN point to some of the strengths of the algorithm. The method was designed to work with large databases, being efficient computationally in that setting, requiring minimal domain knowledge, and has the ability to detect clusters of arbitrary shape, including non-spherical clusters. DBSCAN looks for clusters where the density of points internal to the cluster is relatively consistent and high, and the density of points between clusters is lower. In thinking about method performance, DBSCAN would not be expected to work well on the wine data. The cluster separation in that example that we have seen with other methods was observed in the principal component space, not in the original variable space. However, it has computational advantages when working with large databases that still make it a valuable technique.

Trying to identify clusters as regions of consistent, relatively high density could be approached by requiring all points within a cluster to have a minimal number of points within a certain distance of them (in their neighborhood, specified by some radius). However, points on the borders of clusters could fail this requirement, so requiring it for all points in the cluster does not make sense. DBSCAN works around this by setting criteria based on density requirements that define core points (internal to the cluster, which need to have a certain number of points within a neighborhood of specified radius around them) and border points. Clusters are defined by a set of points which are density-reachable and density-connected; see Ester et al. [147] for details. With these criteria, DBSCAN can also identify points that do not belong to any clusters, which other methods cannot do. These points are considered outliers or noise.

In Ester et al. [147], the DBSCAN algorithm is compared to CLARANS (a hierarchical method built for a similar setting on large databases, extending the CLARA algorithm [362]) on several simulated and real datasets. For the real dataset, the run times of the algorithms show DBSCAN has clear superiority over CLARANS in terms of computational efficiency. The comparison of solutions on simulated data also shows DBSCAN outperforming CLARANS.

When running DBSCAN, the algorithm needs to know the size of the neighborhood around the core points to use and how many points are required to be within that neighborhood. These values could differ from cluster to cluster, but Ester et al. [147] suggest using the values that correspond to the "thinnest" cluster. This way, the thinnest cluster could be detected and clusters that are more dense would easily satisfy the requirements. The authors provide a heuristic to determine the values of these parameters for the dataset at hand [147].

Variants of DBSCAN improving on the original algorithm were readily developed. These include algorithms designed to detect clusters of varied density [381], and versions designed to work with spatial and temporal data [419]. For additional background, see the review in Kriegel et al. [279], and the sur-

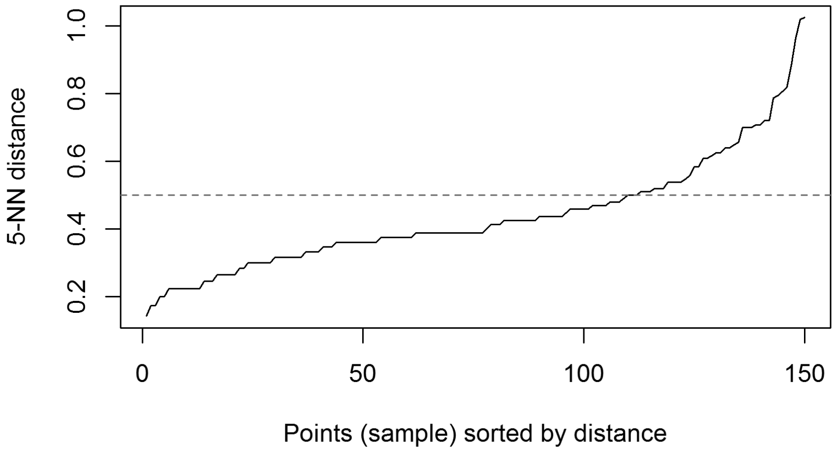


FIGURE 5.17: Nearest neighbor plot for iris data, used to select input parameter for DBSCAN.

vey of five density-based algorithms (DBSCAN and four variants) in Parimala et al. [381].

DBSCAN and its variants can suffer challenges in high-dimensional space, where data is usually sparse (the typical sparsity challenge in high-dimensional settings), due to the density focus [149]. Alternative algorithms to combat this problem, such as the CLIQUE algorithm, first find areas of high density and then focus on finding the clusters in lower-dimensional space [8].

5.8.1 Example with iris data

To demonstrate a density-based clustering method, we run DBSCAN using the `dbscan` R package and function [206]. We swap to the `iris` dataset that we saw in the visualization section, so that we can view the solution in low-dimensional space to see where DBSCAN is finding noise points. This example on the `iris` data is the example from the help file for the package. As a reminder, the dataset contains four measurement variables on three different species of iris. All four measurement variables are provided as inputs to DBSCAN, but species is not used. The first step in the process is to identify the input parameters that DBSCAN needs to run. This is accomplished with a k-nearest neighbor distance plot, and looking for a "knee" (similar concept to looking for an elbow). The plot is shown in Figure 5.17.

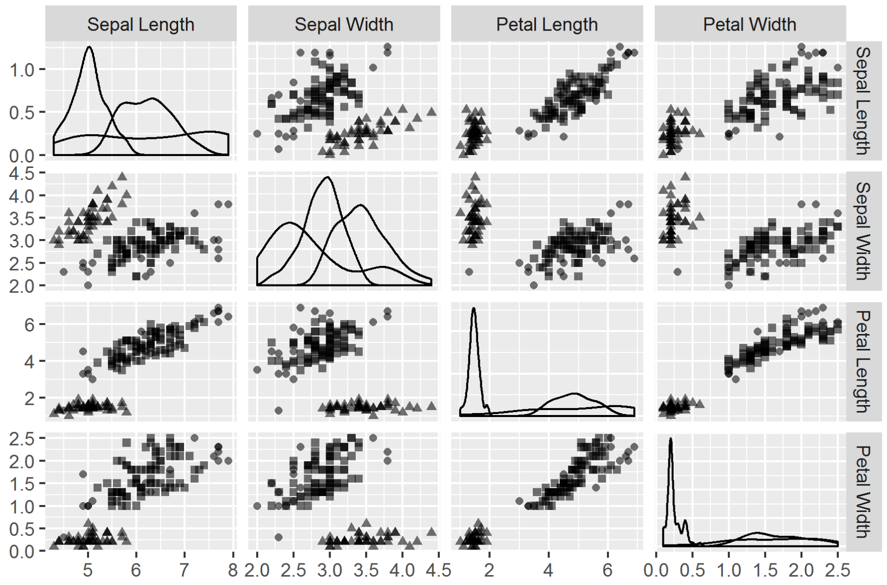


FIGURE 5.18: DBSCAN solution on iris dataset. The noise points found are denoted by circles, while the clusters are denoted by squares and triangles.

Based on that figure, we see a knee at a height of around 0.5, so this is the value used as the radius of the neighborhood in DBSCAN, and the minimum number of points that goes with it is set to be the dimension of the data plus one, or 5, in this case. DBSCAN is then run with those settings. The solution returned finds two clusters, with 49 and 84 points, respectively, and identifies 17 points as noise points. With only four measurement variables used, we can plot the scatterplot matrix using all variables and display the data points with a plotting character indicating their cluster or noise label, as shown in Figure 5.18.

Since we have the species variable for the `iris` dataset, we can make some comparisons between the DBSCAN solution and species. It appears that the DBSCAN solution identified one species as one cluster (indicated by triangles in the plots), and the other two species (squares) as a second cluster, leaving out 17 observations as noise (circles). This is not unrealistic; the virginica and versicolor species are more similar to each other than to the setosa species. If the dataset were higher-dimensional, we would have plotted the solution in principal component space in order to view it.

For more practice with density-based methods, particularly to see how they work with outliers and recovering clusters of arbitrary shape, see Exercises 10 and 11.

5.9 Dealing with network data

An important application of clustering is to find clusters or communities in network data (social networks or otherwise). In these situations, the clustering algorithm may not have access to any information other than the vertex and edge connection information of the network. Here, the clustering is treated as a graph partitioning problem. That is, the dataset is a collection of vertices, V, and edges, E, which describe a graph, G. Typically, these graphs are viewed as simple (meaning no self-edges and no multi-edges), though they may be weighted or unweighted, and directed or undirected. The goal is to partition the graph into k components, where each component's vertices define a cluster. There are many different algorithms that can be used to find clusters on networks, so we briefly outline a few that you may encounter.

1. Hierarchical: As previously described, hierarchical methods can be either agglomerative or divisive, and operate in a similar way as we saw in Section 5.5. They can even use similar linkage concepts, or alternative cost metrics, such as the modularity statistic described below. As we saw before, hierarchical methods give a hierarchy of solutions, one of which must be selected as the final solution, and it turns out that dendrograms can again be used to help visualize solutions.

2. Minimal Spanning Trees: Clustering methods based on minimal spanning trees may be considered a subcase of hierarchical methods, but these methods can also be considered in a graph-theoretic context. Different variants of these algorithms exist, with one common application being in web image analysis [202]. For connections to hierarchical methods, the reader is directed to Gower and Ross [198].

3. Edge Removal-Based: A number of algorithms have been described that focus on finding clusters via iterative edge removal in the graph. Some of these have been based on edge betweenness (a statistic describing the number of shortest paths in the graph that go along that edge) as a metric, and may be called divisive hierarchical algorithms [389].

4. Random Walk: A relatively intuitive algorithm to find communities in networks was proposed using random walks on the graph to define a measure of similarity between vertices [389]. This measure can then be used in agglomerative hierarchical methods as the metric to propose merges to find communities. The general idea is that when starting from a vertex, a relatively short random walk from that vertex (based on its local connections) will tend to end in the same community as the starting vertex.

5. Spectral Partitioning: These methods rely on results from graph theory that tie the number of nonzero eigenvalues of the graph Laplacian (a

matrix derived from the adjacency matrix of the graph) to the number of connected components in the graph. The connected components are treated as the clusters of the graph, and thus this method can be used for clustering. Related methods focus on other properties of the second eigenvalue and eigenvector, using the sign of the entries in the second eigenvector to partition a graph, bisecting it. Similar bisections can be performed on the now separated graphs, until a final solution is obtained. For more details and additional references, see Kolaczyk [273].

These algorithms all have advantages and disadvantages. For example, in practice, edge removal-based algorithms are very computationally intensive, and it has been found that other algorithms, such as the hierarchical ones mentioned above, obtain similar solutions at less computational cost [273]. As you can see from just this sampling of algorithms, some are more heuristic and others rely on mathematical results from graph theory. Many more algorithms exist, all with the goal to find optimal clusters present in the network. Before we explore an example, we introduce a useful statistic called modularity, since this may be used for cluster comparison or validation.

Modularity was first proposed by Newman and Girvan who use this measure to discuss the quality of a particular partitioning solution of the network [361]. Modularity compares the current partition of a network to another network with the same partitioning structure but with randomly placed edges in terms of the fraction of edges that are within a community (i.e., connect vertices within the same cluster). Strong structure is indicated when the modularity is large, because that means the current partition has a much higher fraction of within-community edges than the similar network with randomly-placed edges [361]. This allows modularity to be used as a cost metric to find hierarchical solutions to network problems. In that case, ideally the optimization with respect to modularity would be performed via an exhaustive search of all possible partitions, but that is not feasible in practice. Instead, several greedy search options have been developed that can be used, one of which we demonstrate below. Additionally, modularity can be used as a metric to compare solutions or validate them, though we must bear in mind some solutions are designed to optimize it.

5.9.1 Network clustering example

For our example, we consider network data on dolphins. The data is an undirected social network of frequent associations among 62 dolphins in a community living off Doubtful Sound, New Zealand [327]. Our aim is to identify clusters of dolphins, likely social groups here based on the data collected. In the network, the individual dolphins are the vertices and edges are placed between those that frequently associate. This is the only information used to create the clustering solution on the social network. Our clustering methods will be performed in R using the `igraph` package [112]. We will run three different

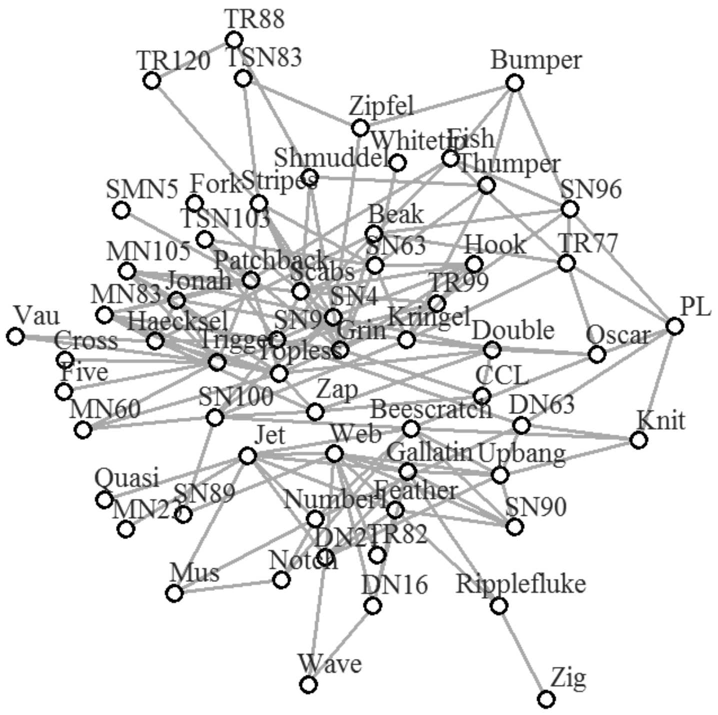


FIGURE 5.19: Dolphin social network with edges indicating frequent associations.

algorithms on the network. Our agglomerative hierarchical algorithm designed to optimize modularity will be implemented via the `fastgreedy.community` function, our random walk-based algorithm will be implemented with the `walktrap.community` function, and finally we will perform spectral partitioning via the `leading.eigenvector.community` function.

Before we look at the clustering solutions, let's look at the original graph, as plotted via the `igraph` package, in Figure 5.19. The network has 62 vertices (one per dolphin) and 159 edges. Summary statistics on the network reveal that each dolphin has at least one connection, and the maximal number of connections is 12. From the plot of the network, we can see that there might be at least two communities present, with a few dolphins that go between them. Now we can run our algorithms and see what communities they find.

Each clustering algorithm here returns a community object, which can be plotted, or have summary measures run on it. For example, the hierarchical algorithm optimizing modularity found four communities, of sizes 22,

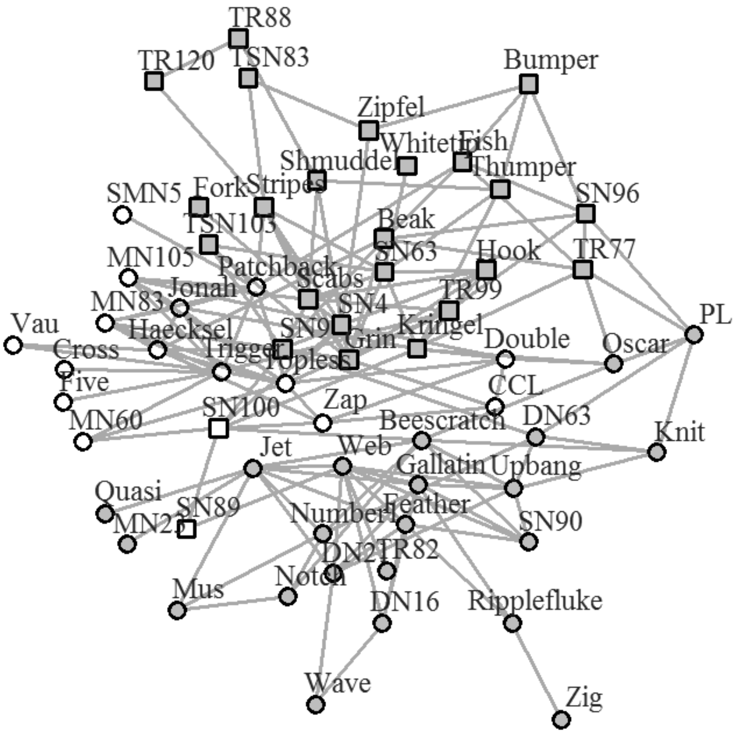


FIGURE 5.20: Agglomerative hierarchical solution optimizing modularity via `fastgreedy.community` function on dolphin data. Communities are represented by vertices which are either white circles, white squares, gray circles, or gray squares.

23, 15, and 2, respectively. We also have the cluster allocation vector previously described. That solution, with vertex shapes and colors indicating the community membership, is shown in Figure 5.20.

Each of the solutions can be plotted this way (though we are showing only one). In addition, hierarchical solutions can still be plotted with a dendrogram, using some additional packages and commands (`dendPlot` in the `ape` package [380]) as shown in Figure 5.21.

Briefly, we compare our three clustering solutions in Table 5.15. We compute modularity for each solution, though we note that only the hierarchical method is specifically trying to optimize this measure. All the solutions have very similar modularity, nearly 0.5, which seems to reflect that some structure is present. A quick comparison of cluster labels (not shown) shows that

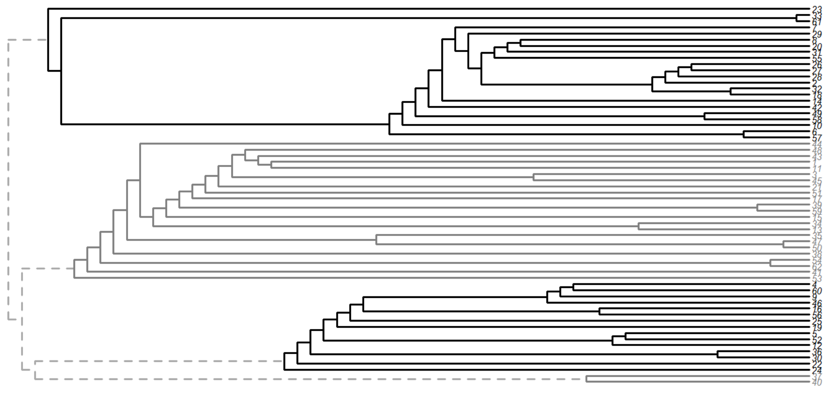


FIGURE 5.21: Dendrogram showing agglomerative hierarchical solution optimizing modularity via `fastgreedy.community` function on dolphin data.

TABLE 5.15: Comparison of three clustering solutions for the dolphin network data.

Method	Number of Clusters	Cluster Sizes	Modularity
Hierarchical (Agg)	4	22, 23, 15, 2	0.4955
Random Walk	4	21, 23, 16, 2	0.4888
Spectral Partitioning	5	8, 9, 17, 14, 14	0.4912

the hierarchical solution optimizing modularity and the random walk-based solution are nearly identical (though their smallest clusters are different). The spectral clustering solution differs from both with an additional cluster that splits one of the larger clusters in the other solutions in two.

Community detection is an important application of clustering, and now that we've seen an example, we can turn our attention to some final challenges regarding clustering.

5.10 Challenges

The challenges described below apply across methods covered in previous sections.

5.10.1 Feature selection

Which variables in your dataset should be used in the clustering analysis? For our exposition, we assumed that the variables for use in clustering had already been selected. However, choosing the variables for use in the analysis is not always so straightforward. This problem is often termed variable or feature selection. Although this subject could use an entire chapter's worth of exposition itself, we wanted to briefly highlight some work and some references for interested readers.

In our discussions of dissimilarity measures, we noted that different weights could be used for each variable in the construction of the dissimilarity, but that we assumed equal weights. Allowing for different weights is one way to incorporate variable selection, and this approach has been studied by statisticians for years. In 1995, Gnanadesikan et al. compared multiple approaches for weighting on several simulated and real datasets. Their findings found that weightings based on the standard deviation or range of variables were problematic, while more complicated weightings, if chosen carefully, fared better [191].

More recent work in 2004 tackled the problem in different ways. Dy and Brodley examined feature selection in conjunction with needing to estimate k, the number of clusters. They describe different ways of approaching the feature selection (filter versus wrapper approaches), and present work in relation to a Gaussian model-based EM algorithm for clustering using a wrapper approach. They point out that the number of clusters found can differ based on the number of variables selected. They examine various performance criteria and report on their findings based on simulated and real data. In conclusion, they find that incorporating the selection of k into the feature selection process was beneficial, particularly because not all subsets of features have the same number of clusters [138].

In related work, Friedman and Meulman consider feature selection using weights, where the goal is to find clusters which are based on potentially different variable subsets. Their approach is called clustering objects on subsets of attributes (COSA) and is also developed in a wrapper framework. Results on both simulated and real datasets are presented. A bonus with this work is a series of discussion responses from other statisticians discussing the problem and this proposal [172].

Finally, for a recent review of feature selection work in clustering, see Alelyani et al. [12]. This review touches on the problem in many different clustering applications and for different types of datasets.

5.10.2 Hierarchical clusters

Thus far in our exploration of clustering, we've seen methods that assign each observation to a single cluster. However, in practice, this may not be realistic for many applications, especially in social network settings. Some challenges arise when trying to study hierarchical clusters, or communities, or as described in the next subsection, overlapping clusters. Clearly, hierarchical structures occur naturally in the world—just consider the organization of a school, college, or university. Our hierarchical methods can provide ways to uncover these structures, by considering multiple solutions at different values of k. Higher levels in the hierarchy are examined by choosing a small k, while lower levels are examined by choosing a large k. For example, in a social network setting, with clustering childen who participate in club soccer, hierarchical methods might reveal teams at a lower level, and then reveal leagues at a higher level, based on social connections between children. When considering automobiles, a lower-level cluster might be vehicles all of the same model, but a higher-level cluster might reveal make (as manufacturers make multiple models) or vehicles of a similar type, such as a cluster of SUVs or vans. These sorts of relationships are more easily explored with hierarchical methods, where there is a clear understanding about how clusters are related to one another (e.g., clusters C_3 and C_5 merged to make cluster C_8), rather than in partitioning methods, where there is not necessarily a relationship between clusters found at different values of k.

5.10.3 Overlapping clusters, or fuzzy clustering

When studying hierarchical clusters, an observation may have cluster memberships defined by the different hierarchical levels studied. A related but distinct issue is the issue of overlapping or fuzzy clustering, where an observation can have membership in more than one cluster at a time (not dependent on its hierarchical level). Cluster memberships here are often represented by a vector of probabilities of assignment to each of the clusters. For example, in a three-cluster solution, observation 4 may have a cluster membership vector (0.40, 0.20, 0.40), meaning that the observation belongs equally well to clusters 1 and 3, but less so to cluster 2. For a real-life contextual example, imagine clusters created based on spending habits of Amazon.com customers. A customer might end up in clusters for purchases related to horror movies as well as for children's books (e.g., a parent who is a horror movie fan). Similarly, in text analysis, one could consider texts that have multiple authors, and based on their writing styles, the text might end up in several clusters or, if clustering based on meaning, words with several meanings could also cause texts to be in several clusters.

Specific methods have been developed to help study hierarchical clusters and overlapping clusters. As examples, the reader is directed to Lanchichinetti et al. [298] and Darmon et al. [116].

5.11 Exercises

For our exercises, we describe some example analyses that can be completed in R with the packages indicated. Feel free to use your software of choice instead. For most of our problems, it is assumed that you will be standardizing the data, unless otherwise specified.

TABLE 5.16: A sample distance matrix for use in Exercise 5.

Obs	1	2	3	4	5	6
1	0					
2	4.7	0				
3	6.6	11.1	0			
4	8.1	11.6	3.4	0		
5	4.9	5.7	6.2	7.7	0	
6	2.9	2.5	7	6.8	10.4	0

1. Find an example of clustering in an application area of interest to you. What clustering methods are used? How is the number of clusters in the solution determined?

2. The `iris` dataset (available in R using the command `data(iris)`) was used to show example visualizations that can indicate clustering is appropriate for a dataset. Implement k-means on the dataset without the Species variable using the unstandardized measurement variables using three clusters. Then, run it using the standardized variables with three clusters. What impact does standardizing have on the solutions? Does one solution recover Species better than the other?

3. PAM and CLARA are alternative partitioning methods to k-means, and both functions are implemented in R in the `cluster` package [331]. How do these methods differ from k-means? Let's find out. The wine data in our examples is freely available from UCI's machine learning repository [136]. Obtain the dataset, and then perform PAM and CLARA on it (without the cultivar variable), using an appropriate approach to determine a number of clusters for the solution. Compare the solutions to the k-means solution shown in Section 5.4.3, or to a k-means solution you obtain.

4. We saw the `clValid` package used to compare several clustering solutions on the wine dataset. For practice with this approach, apply `clValid` using your choice of inputs on the `iris` dataset without the Species variable. Which methods and numbers of clusters are chosen as the optimal solutions by the different validation measures?

5. Our simple five-point example for hierarchical clustering (Figure 5.8 and Table 5.7) showed how to implement the single linkage approach starting from the distance matrix. A distance matrix for six points is shown in Table 5.16. Using this distance matrix, perform single, complete, and average linkage hierarchical clustering. For each of these three approaches, sketch the dendrogram of the solution. How different are the solutions?

6. In our application of hierarchical clustering to the wine dataset, we used the `hclust` function. `agnes` is an alternative agglomerative hierarchical function in the `cluster` package, and `diana` is an associated divisive hierarchical function. Apply `agnes` and `diana` to the `iris` dataset (without Species), and plot the associated dendrograms. How do the solutions compare?

7. We saw dendrograms as our primary visualization of hierarchical clustering solutions. Banner plots are an alternative, available via the `cluster` package in R. For the hierarchical solutions in Exercise 6, obtain the corresponding banner plots. How do the banner plots compare to the dendrograms? Do you find you prefer one visualization over the other?

8. Implement model-based clustering using `mclust` for the `iris` dataset (without Species). What model is selected as the "best"? What criterion is used to assess which solution is "best"?

9. For the `iris` dataset (without Species), obtain silhouette values for model-based clustering solutions with $k = 3$, $k = 7$, and $k = 10$ clusters. Based on the silhouette values, which clustering solution do you prefer?

10. To investigate density-based methods, let's generate data to cluster. The `clusterGeneration` package in R allows for the generation of clusters according to user specifications [395]. Generate data that you think DBSCAN or another density-based method will work well on, then implement the density-based method and see if it recovers the clusters well. Note that the R function here can generate outlier points (outside of any cluster), which DBSCAN can detect and label as such, but methods such as k-means cannot do.

11. One of the strengths of density-based methods is the ability to find clusters of arbitrary shape, rather than fairly common spherical clusters. Generate data with at least one non-spherical cluster in low dimensions. (You will need to think about how to do this.) Suggestions include crescent moon shapes or a ring cluster around a spherical one. Implement k-means and a density-based method and see which recovers the clusters better.

12. The dolphin dataset used in our network community detection example is freely downloadable [327]. In our results, we showed only the result of the `fastgreedy.community` function. Implement the random walk-based algorithm with the `walktrap.community` function and spectral partitioning via the `leading.eigenvector.community` function and verify the results presented in the text about the comparison of the solutions.

13. Our case study on the College Scorecard dataset demonstrated k-means and hierarchical clustering solutions on a set of nine variables [367]. Using

those same nine variables, implement model-based clustering, DBSCAN, and any other methods you are interested in, and compare the solutions.

14. The College Scorecard case study dataset had 1978 variables available and is a good illustration of having to do feature selection while dealing with missing data. Explore this dataset yourself, identify roughly 10 variables you'd like to try clustering on, then examine how much data is missing. Cluster the complete cases you have for your 10 variables (not complete cases overall) using your preferred clustering method(s). What do you find? How many observations were you able to cluster? Did you find a strong clustering solution?

Chapter 6

Operations Research

Alice Paul

Olin College

Susan Martonosi

Harvey Mudd College

Operations research (OR) is the use of mathematical and computational tools to guide decision-making under dynamic and uncertain conditions. At the heart of such decision-making is data, and operations researchers

- make sense of data (descriptive analytics)
- to model complex systems (predictive analytics)
- and recommend actions (prescriptive analytics).

In this sense, operations research encompasses data science. Additionally, operations research offers powerful tools, such as optimization and simulation, that are embedded in the engines of common statistical and machine learning methods to help those methods make better decisions about, e.g., how to cluster data points or tune algorithm parameters. This chapter provides a brief overview of the most central operations research methods that connect to data science, tying in examples of current research that illustrate OR both as an overarching process for leveraging data to make better decisions, and as a set of computational tools that permit rapid analysis of ever-growing datasets.

Section 6.1 provides an overview of the field and its connection to data science, articulating the fundamental trade-off in mathematical modeling between model efficiency and model complexity. Section 6.2 presents the deterministic optimization tools of mathematical programming that form the foundation of traditional operations research. Given the omnipresence of *stochasticity* (randomness) in most complex systems, simulation is a powerful tool for modeling uncertainty; its use as a modeling tool and as a subroutine in statistical and machine learning is described in Section 6.3. Stochastic optimization, the subject of Section 6.4, incorporates uncertainty into the optimization framework. We illustrate these techniques in the context of applications of prescriptive analytics in Section 6.5. Section 6.6 provides a brief overview of the commercial and open-source software available for operations research methods. Having laid the foundation of operations research methodology, Section 6.7 concludes by describing exciting new directions in the field of OR as it connects to data science.

6.1 History and background

The field of operations research emerged in Britain around 1938, a few years before the start of the Second World War. During WWII, OR methods are credited with reducing losses of United States naval ships from German submarine action and increasing bombing accuracy (i.e., reducing dispersion) by 1000%. Additionally, operations researchers envisioned the first neural network in 1943. At the end of WWII, the military maintained its OR groups, and research continued in the use of OR methods to improve military and government operations. Over the past century, the military has remained a strong user and producer of OR methodology, while OR's reach has extended to nearly all industries, including manufacturing, transportation, and e-commerce [178].

6.1.1 How does OR connect to data science?

Operations research has always concerned itself with decision-making under uncertainty, and the well-trained operations researcher has a firm foundation in the methods of probability and statistics. As the collection and availability of data has grown ubiquitous in the last several decades, the need for operations research thinking to make sense of that data and prescribe action has likewise grown.

This chapter describes four ways in which operations research connects to data science:

1. Many traditional tools of operations research, including optimization methods, serve as subroutines in common statistical and machine learning approaches.
2. Simulation facilitates inference and decision-making when sufficient data are absent.
3. Prescriptive analytics is the use of the OR modeling process to derive recommendations from data.
4. A new area of research in OR casts common statistical and machine learning problems as optimization problems to generate optimal predictions. (See Section 6.7.)

We describe several of the tools commonly associated with operations research in this chapter. However, it is important to realize that OR is more than a black box of tools that can be used to draw insights from data. Additionally, it is a way of thought that seeks first to distill a system, problem, or dataset into its most fundamental components, and then to set about using mathematical tools to deepen one's understanding and make recommendations.

6.1.2 The OR process

An OR approach to problem-solving (and likewise, to data science) typically consists of these steps [102, 219, pp. 109-112]:

1. Defining the problem
2. Formulating a model
3. Solving the model
4. Validating the model
5. Analyzing the model
6. Implementing the recommendations of the model

While the mathematical techniques described in this chapter and in other chapters of this book address steps 2–5, one can argue that steps 1 and 6 are the most challenging. To properly define a problem, one needs a deep understanding of the problem context: who are the stakeholders, what are their objectives, and what factors constrain their actions? And successful implementation of the recommendations of a model requires the trust of the stakeholders, which can only be achieved through clear communication throughout the process. Two quotations from Robert E. D. "Gene" Woolsey, a renowned industrial consultant, highlight this [515][1]:

- "I was approached by the city of Denver to help them site a new firehouse costing some millions of dollars. I had a particularly bright-eyed and bushytailed student who volunteered for the project. . . . I told him gently that the first requirement was to go and be a fireman for a few months and then we would talk. . . . It did not take long as a working fireman for him to discover that virtually all of the assumptions he had brought to the problem were wrong."
- "[I]t is easier to do the math correctly (which requires no politics) than to get the method implemented (which always requires politics)."

[1]Several of Gene Woolsey's papers and lectures have been compiled in *Real World Operations Research: The Woolsey Papers* [516]. The quotes cited above also reiterate the importance of domain knowledge, first mentioned in Section 1.5.

Thus, to have any hope for the modeling process to inform organizational decisions, the reverse must also be true: the organizational context must inform the modeling process.

6.1.3 Balance between efficiency and complexity

Computation began to emerge around the same time as OR. Without high-speed computers to analyze data and solve optimization problems, early operations researchers embraced the craft of trading off model complexity for model efficiency. There are two famous sayings that exemplify this trade-off:

> "All models are wrong, but some are useful." (George E. P. Box, statistician, [58])

This first quote reassures us in our inevitable imperfection: the world is complex and replete with variability. Even the most sophisticated model cannot perfectly capture the dynamics within a complex system, so we are absolved from the requirement of perfection. Our goal, therefore, is to develop a model that is useful in its ability to elucidate fundamental relationships between variables, make predictions that are robust to the variation exhibited in past data, and most important to the operations researcher, recommend relevant and sage decisions.

> "A model should be as simple as possible, and no simpler." (popular paraphrasing of a statement by Albert Einstein, physicist, [76, 146])

This second quote provides guidance on how to create such a model. We start with the simplest model conceivable. If that model fails to capture important characteristics of the system it is trying to emulate, then we iteratively add complexity to the model. This iterative process requires validation against known behavior (i.e., data) at each step. We stop as soon as the model is adequate for its intended purposes.

At the interface of OR with data science, we see the complexity/efficiency tradeoff materialize in many ways:

- When developing a mathematical programming model (which is the topic of Section 6.2), we try whenever possible to represent the decision problem as optimizing a *linear* function over a feasible region defined by a system of *linear* constraints. Surprisingly, this simple linear framework is adequate for broad classes of systems! Only when a linear framework is unachievable do we start to incorporate nonlinear objective functions or constraints, and even then, we prefer to sacrifice model fidelity to maintain computational tractability. This trade-off is discussed in greater detail in Section 6.2.1.

- When choosing input distributions for parameters of a simulation, we rely heavily on simple distributions whose properties generally capture natural phenomena while remaining analytically tractable. This is discussed in Section 6.3.1.

- Multistage sequential stochastic optimization problems suffer from the so-called *curse of dimensionality*, which means that the solution space explored by optimal algorithms grows much too fast with the size of the problem. Such problems can rarely be solved to guaranteed optimality; instead approximations and even simulation are used to yield very good solutions in a reasonable amount of time. These methods are described in Section 6.4.2 and the curse of dimensionality is revisited in Section 7.2.

- When fitting a statistical model or tuning parameters in a machine learning procedure, we choose predictor variables or parameter values that achieve not only a low error rate on the data used to fit the model (training data) but also a low error rate on unseen testing data. While a very complex model that is highly tuned to the training data can appear to have a good "fit," a simpler model often does a better job of capturing the prominent trends that permit good prediction on unseen data. This problem of avoiding *overfitting* was discussed in Chapter 4 and will appear again in Chapter 8. We also describe it in the context of operations research methods in Sections 6.2.4 and 6.3.3.2.

The challenge in achieving the right balance in this trade-off is to sufficiently understand the problem being modeled in order to discern the critical factors and relationships from those that add complexity without insight. Sensitivity analysis is a means to assess whether simplifying assumptions dramatically alters the output of the analysis. If our model or approach is robust to these assumptions, then we have no need to make the model more complex. Section 6.5 presents several real-world examples of operations research methods in action to solve large-scale, complex decision problems. But first, we must introduce the foundation of OR methodology: optimization.

6.2 Optimization

Optimization is an important tool used in the overall operations research process to fit a model or make decisions based on a model. Consider some typical data science problems: a credit card company wants to predict fraudulent spending, a hotel wants to set room prices based on past demand, or a medical researcher wants to identify underlying populations with similar risk profiles. As seen in previous chapters, data scientists approach all of these problems by transforming them into corresponding optimization problems. In supervised learning problems, such as linear regression introduced in Chapter 4, fitting a given model to past data is an optimization problem in which we set the model parameters to obtain the best fit. Outside of prediction, unsupervised learning problems such as clustering the data into k subgroups or clusters can also be framed as an optimization problem trying to maximize the intra-cluster similarity, as detailed in Chapter 5. Further, after analyzing the data or fitting a model, there remain operations research problems to be solved based on these insights. For example, a hotel may want to set prices for the coming spring to maximize revenue. We refer to this category of problem as prescriptive analytics and discuss it in more detail in Section 6.5.

In any optimization problem, we are given a set of feasible solutions, each associated with some objective function value. The goal is to find a feasible solution that optimizes that objective function. To write this in mathematical form, we use a *mathematical program*:

$$\begin{aligned} \text{maximize } & f(\mathbf{x}) \\ \text{subject to } & g_i(\mathbf{x}) \leq 0 \quad \forall i = 1, 2, \ldots, m \\ & \mathbf{x} \in \mathcal{X} \end{aligned}$$

Here, the variables are given by the vector $\mathbf{x}$, and the set $\mathcal{X}$ represents the type of values $\mathbf{x}$ can take on. Typically, we think of $\mathcal{X}$ as the set of real vectors $\mathbb{R}^n$, but we could also restrict $\mathbf{x}$ to be non-negative $\mathbb{R}^n_{\geq 0}$ or integer-valued $\mathbb{Z}^n$ or a combination. A *feasible solution* is defined as a vector $\mathbf{x} \in \mathcal{X}$ that satisfies the m constraints $g_i(\mathbf{x}) \leq 0$ for $i = 1, 2, \ldots, m$, where $g_i : \mathcal{X} \to \mathbb{R}$. Lastly, $f : \mathcal{X} \to \mathbb{R}$ gives the objective function value of $\mathbf{x}$. Solving this mathematical program would return a feasible solution $\mathbf{x}$ that maximizes $f(\mathbf{x})$. Such a solution is called an *optimal solution*.

This framework is general in that we can also capture problems minimizing $f(\mathbf{x})$ (equivalent to maximizing $-f(\mathbf{x})$), inequality constraints of the form $g_i(\mathbf{x}) \geq 0$ (equivalent to the constraint $-g_i(\mathbf{x}) \leq 0$), and equality constraints $g_i(\mathbf{x}) = 0$ (equivalent to the constraints $g_i(\mathbf{x}) \leq 0$ and $g_i(\mathbf{x}) \geq 0$). If there are no constraints on $\mathbf{x}$, then the problem is called an *unconstrained optimization problem* and is often easier to solve. Otherwise, the problem is a *constrained optimization problem.*

Note that we have not restricted the functions f or g_i to take on any particular form. In fact, we have not even specified that we can efficiently calculate their values. As it stands, any optimization problem could be placed into this form. However, we also have not guaranteed that we can solve the problem above. As we will see, a simpler format of these functions generally allows us to find an optimal solution more easily. Sections 6.2.2–6.2.4 introduce the most common tiers of optimization problems of increasing complexity and emphasize the corresponding tractability trade-off.

6.2.1 Complexity-tractability trade-off

The generality of the objective function f, constraint functions g_i, and set $\mathcal{X}$ allows us to capture very complex optimization problems. For example, this form captures both the simple optimization underlying linear regression as well as the much more complicated optimization problem of setting the parameters for a neural network. However, there is a trade-off between the complexity of the optimization problem and the corresponding tractability of solving it.

As the complexity of an optimization problem increases, the time needed to find an optimal solution also increases. How this time grows with the number of data features and points is especially important as the size of the data we have access to increases. For the credit card company wanting to predict fraud, this time also matters from a practical standpoint since there is limited time in which the company must flag the transaction. If we cannot find an optimal solution in fast enough time, then we may have to settle for finding a solution we think is "good enough." Therefore, it is important to understand when we can restate a problem in a simpler form or make some simplifying assumptions without sacrificing too much accuracy in problem representation in order to gain efficiency.

Further, simpler optimization problems may yield more useful results to the designer. By returning a more structured and interpretable result, the user may gain more insight into future decisions. To solve a prediction problem for hotel bookings, we can choose between many models, each of which has its own underlying optimization problem. If in one model, the effect between price and the probability of booking is clear from the optimized model parameters, then the manager can try to adjust prices to increase the occupancy rate. However, another more complicated model may fit the data slightly better but not give this relationship so clearly. This inference from the data and model is discussed more in Section 6.5.

In the following sections, we will introduce several common optimization paradigms of increasing complexity and corresponding optimization algorithms. The algorithms introduced take as input the optimization problem and return a feasible solution $\mathbf{x}$. We say an algorithm is *efficient* or polynomial-time if the runtime of the algorithm (i.e., the number of steps) scales polynomially in the input size of the problem. Here, the input size corresponds to the space needed to store $\mathbf{x}$ and the space needed to represent f, g_i, and $\mathcal{X}$. Having an efficient algorithm that returns an optimal solution is an important benchmark for the complexity of a problem. However, we won't always be guaranteed to find such an algorithm. In those cases, we will introduce efficient algorithms that return a feasible solution that we hope is close to optimal. These optimization algorithms are called *heuristic algorithms.*

Almost all data science problems can be captured in the following paradigms. While problem-specific algorithms can also be developed, understanding when an optimization problem falls into each category is an important first step to solving the problem. Further, we direct the reader to open-source software for solving problems in each of these paradigms in Section 6.6.

6.2.2 Linear optimization

We first consider the simplest functional form for f and g_i, linear functions. Linear optimization problems are comprised of a linear objective function and linear constraints. In other words, we can write the objective function as $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} = c_1x_1 + c_2x_2 + \ldots + c_nx_n$ and each constraint as $g_i(\mathbf{x}) = \mathbf{a_i}^T\mathbf{x} = a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n - b_i \leq 0$ for $i = 1, 2, \ldots, m$. (For a refresher on linear algebra, see Chapter 3.) We also assume the variables $\mathbf{x}$ are real-valued, i.e., $\mathcal{X} = \mathbb{R}^n_{\geq 0}$. We often write this mathematical program in *standard form* using vector and matrix notation,

$$\begin{aligned} \text{maximize } & \mathbf{c}^T\mathbf{x} \\ \text{subject to } & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{6.1}$$

Here, $\mathbf{c}$ and $\mathbf{b}$ are vectors and A is an $m \times n$ matrix of constraint coefficients. We let $\mathbf{a_i}$ denote the ith row of A and $\mathbf{A_j}$ denote the jth column of A. This restricted form of optimization problem is called a *linear program*. The linear program (6.2) gives an example.

$$\begin{aligned} \text{maximize } & 12x_1 + 10x_2 \\ \text{subject to } & x_1 \leq 3 \\ & x_2 \leq 4 \\ & x_1 + x_2 \leq 6 \\ & x_1 \geq 0, x_2 \geq 0. \end{aligned} \tag{6.2}$$

Example Applications of Linear Programming

1. Transportation Problem: optimally assign customer orders to factories to minimize shipping costs, subject to factory production limits [282].
2. Image Segmentation: segment an image into foreground and background [159].
3. Image Reconstruction: restore a noisy image to original form [443].

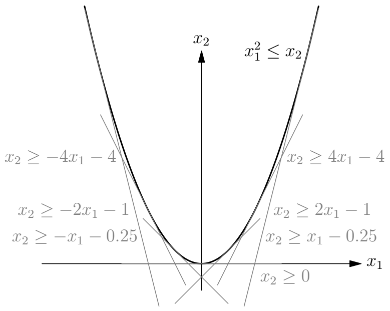


FIGURE 6.1: Approximating a nonlinear constraint with multiple linear constraints.

At first, it may seem very limiting to assume a linear objective function and constraints. However, many common problems can be formulated as linear programs. Further, more complicated constraints or objective functions can sometimes be well approximated using linear functions. As shown in Figure 6.1, the constraint $x_1^2 \leq x_2$ can be approximated to arbitrary precision using multiple linear constraints.

Linear optimization benefits greatly from its imposed structure. First, the *feasible region*, the set of all feasible solutions, is given by the intersection of the half-spaces defined by each linear inequality ($\mathbf{a_i}^T\mathbf{x} \leq b_i$ and $x_j \geq 0$). This type of subspace is called a *convex polytope*. The polytope for the linear program (6.2) is given in Figure 6.2 along with several *isoprofit lines* $\mathbf{c}^T\mathbf{x} = \alpha$ plotting all points with the objective function value α. It is easy to see that the vertex $(3, 3)$ is an optimal solution to the linear program. One nice feature of linear programming is that the set of optimal solutions will always be given by a face of the polytope, where a 0-dimensional optimal face corresponds to a single optimal vertex. This implies that we can always find an optimal vertex (although it becomes harder to visually find the solution as the dimension increases).

Optimal Set for Linear Programming

Consider any linear program. Then, any local optimal solution is a global optimal solution. Further, when the feasible region contains a vertex, then if there exists an optimal solution (i.e., the feasible region is nonempty and the objective function is bounded), then there exists an optimal solution that is a vertex of the feasible region.

This insight is critical to the algorithms used to find an optimal solution. The most commonly known algorithm to solve linear programs, Dantzig's *simplex algorithm* [115], is not a polynomial-time algorithm. Nevertheless, the simplex algorithm tends to work very well in practice, and the idea behind the algorithm is beautifully simple and intuitive.

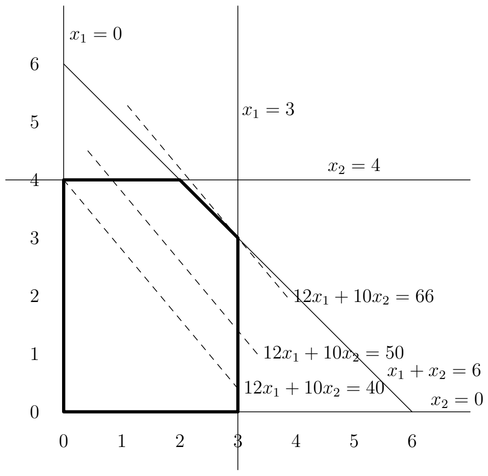


FIGURE 6.2: Finding the optimal solution of the linear program (6.2). The feasible region is outlined in bold and isoprofit lines for $\alpha = 40, 50, 66$ are drawn dashed, showing that (3,3) is optimal.

Simplex Algorithm

The simplex algorithm starts at a vertex of the feasible region. Finding such a point can be done efficiently by actually solving a related linear program for which we know a feasible vertex [47]. In each iteration, the algorithm searches to find an adjacent vertex with a higher objective function value. If it finds one, it moves to that new vertex. Otherwise, the algorithm outputs the current vertex as an optimal solution. The possible inefficiency in this algorithm comes from the fact that in the worst case the algorithm can visit exponentially many vertices. This worst-case behavior requires a very particular constraint structure, and in practice, the number of vertices visited tends to scale well with the input size.

6.2.2.1 Duality and optimality conditions

As one might expect, given that they are the simplest form of constrained optimization, linear programs can be efficiently solved. However, developing a polynomial-time algorithm requires using *duality* to find additional structure of an optimal solution and to understand how the optimal solution would change if we modified the constraints or objective function. In particular, consider any linear program in the form (6.1) (called the *primal linear program*) and let y_i be a Lagrangian multiplier associated with the inequality constraint $\mathbf{a_i}^T x - b_i \le 0$. Then, the Lagrangian relaxation is given by

$$\max_{\mathbf{y}\ge 0} \inf_{\mathbf{x}\ge 0} \left[\mathbf{c}^T\mathbf{x} + \sum_{i=1}^{m} (b_i - \mathbf{a_i}^T\mathbf{x}) y_i \right] .$$

Assuming that (6.1) has a nonempty feasible region and bounded objective function, then this is equivalent to the following optimization.

$$\min_{\mathbf{x}\geq 0}\sup_{\mathbf{y}\geq 0}\left[\mathbf{b}^T\mathbf{y}+\sum_{j=1}^{n}(c_j-\mathbf{A_j}^T\mathbf{y})x_j\right].$$

This leads to the following dual linear program.

$$\begin{aligned}\text{minimize } & \mathbf{b}^T\mathbf{y} \\ \text{subject to } & A^T\mathbf{y}\geq \mathbf{c} \\ & \mathbf{y}\geq \mathbf{0}.\end{aligned} \tag{6.3}$$

Since the dual variables $\mathbf{y}$ can be seen as Lagrangian multipliers associated with each constraint of (6.1), y_i represents how much the optimal solution value of (6.1) would increase if we relaxed the constraint $\mathbf{a_i}^T\mathbf{x}\leq b_i$ to be $\mathbf{a_i}^T\mathbf{x}\leq b_i+1$.

It is easy to show that for all feasible $\mathbf{x}$ to (6.1) and feasible $\mathbf{y}$ to (6.3), $\mathbf{c}^T\mathbf{x}\leq\mathbf{b}^T\mathbf{y}$. This property is called *weak duality* and it implies that the objective function in (6.3) can be interpreted as finding the tightest upper bound for the program (6.1). In fact, linear programs satisfy the stronger condition of *strong duality*: if both (6.1) and (6.3) have a nonempty feasible region, then there exist feasible $\mathbf{x}^*$ and $\mathbf{y}^*$ such that $\mathbf{c}^T\mathbf{x}^*=\mathbf{b}^T\mathbf{y}^*$. This implies that if we find feasible $\mathbf{x}$ and $\mathbf{y}$ with matching objective function values then we know both are optimal for their respective linear programs.

Another way to test for optimality is through a set of conditions called the Karush-Kuhn-Tucker (KKT) conditions. These conditions are based on the theory of Lagrangian multipliers and also rely on the dual solution $\mathbf{y}$. In particular, for linear programs, the KKT conditions can be stated as follows.

KKT Conditions for Linear Programming

Consider any linear program in the form (6.1). Then $\mathbf{x}\in\mathbb{R}^n$ is an optimal solution if and only if there exists $\mathbf{y}\in\mathbb{R}^m$ such that

1. $\mathbf{x}$ is feasible for (6.1),
2. $\mathbf{y}$ is feasible for (6.3),
3. $y_i(\mathbf{a_i}^T\mathbf{x}-b_i)=0$ for all $i=1,2,\ldots,m$, and
4. $x_j(\mathbf{A_j}^T\mathbf{y}-c_j)=0$ for all $j=1,2,\ldots,n$.

The last two constraints are referred to as *complementary slackness*. In fact, the simplex algorithm can also be viewed as maintaining a feasible solution $\mathbf{x}$ and an infeasible dual solution $\mathbf{y}$ that satisfies complementary slackness with $\mathbf{x}$. In each step, the algorithm moves towards feasibility of $\mathbf{y}$.

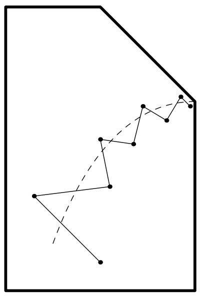


FIGURE 6.3: The iterative path of the interior-point algorithm (solid) following the central path (dashed).

Interior-point algorithms

We can use duality and the KKT conditions to design an algorithm to find an optimal solution. Khachiyan [266] introduced the first polynomial-time algorithm to solve linear programs called the ellipsoid algorithm. This was shortly followed by Karmarkar's [260] efficient interior-point algorithm, which led to the popular class of algorithms called *interior-point methods.* This class of methods updates a feasible solution on the interior of the feasible region rather than a vertex. For simplicity, we will describe the *primal-dual interior-point* algorithm, but there exist other interior point methods that are also efficient but used less in practice [517]. In the primal-dual interior-point algorithm, we maintain a primal feasible solution $\mathbf{x}$ and a dual feasible solution $\mathbf{y}$ and work towards satisfying complementary slackness. Let $\mathbf{s_1} \geq \mathbf{0}$ and $\mathbf{s_2} \geq \mathbf{0}$ be new variables that represent the slack in the primal and dual constraints, respectively. Then, we can rewrite the primal feasibility condition as $A\mathbf{x} + \mathbf{s_1} = \mathbf{b}$, the dual feasibility condition as $A^T\mathbf{y} - \mathbf{s_2} = \mathbf{c}$, and the complementary slackness conditions as ${\mathbf{s_1}}^T\mathbf{y} = 0$ and ${\mathbf{s_2}}^T\mathbf{x} = 0$. Let $\mathcal{F} := \{(\mathbf{x}, \mathbf{y}, \mathbf{s_1}, \mathbf{s_2}) \mid A\mathbf{x} - \mathbf{b} + \mathbf{s_1} = \mathbf{0}, A^T\mathbf{y} - \mathbf{c} - \mathbf{s_2} = \mathbf{0}, (\mathbf{x}, \mathbf{y}, \mathbf{s_1}, \mathbf{s_2}) \geq 0\}$ be the set of primal and dual feasible solutions. We define the *central path* as the set of solutions that satisfy primal and dual feasibility and approximately satisfy complementary slackness:

$$\text{CP} := \{(\mathbf{x}, \mathbf{y}, \mathbf{s_1}, \mathbf{s_2}) \in \mathcal{F} \mid {\mathbf{s_1}}^T\mathbf{y} = \tau, {\mathbf{s_2}}^T\mathbf{x} = \tau \text{ for some } \tau \geq 0\}.$$

The set is called the central path since the set of $\mathbf{x}$ in CP traces a path on the interior of the feasible polytope. See Figure 6.3. Further, for all $\tau > 0$ there is a unique feasible solution $(\mathbf{x}, \mathbf{y}, \mathbf{s_1}, \mathbf{s_2}) \in \mathcal{F}$ such that ${\mathbf{s_1}}^T\mathbf{y} = \tau$ and ${\mathbf{s_2}}^T\mathbf{x} = \tau$, and as $\tau \to 0$, this solution approaches an optimal one.

The primal-dual interior-point algorithm tries to approximate this central path and decrease τ on each step. The algorithm starts at a feasible solution $(\mathbf{x}, \mathbf{y}, \mathbf{s_1}, \mathbf{s_2})$ on the interior of the polytope that is close to the central path for some $\tau > 0$. Then, on each iteration, the algorithm moves towards a feasible point $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{s_1}}, \hat{\mathbf{s_2}}) \in \mathcal{F}$ such that $\hat{\mathbf{s_1}}^T\hat{\mathbf{y}} = \tau$ and $\hat{\mathbf{s_2}}^T\hat{\mathbf{x}} = \tau$ for some set value of τ. The algorithm uses Newton's method to take a single step towards this point.[2] By gradually decreasing τ and the step size, the algorithm maintains feasibility and roughly follows the central path to approach an optimal solution. Thus, this method can be seen as an example of gradient descent on the primal and dual problems simultaneously. Interestingly, the central ideas behind the primal-dual interior-point algorithm will extend to more general convex optimization, as discussed further in Section 6.2.3.

6.2.2.2 Extension to integer programming

While linear programming is a powerful optimization tool, sometimes we want to restrict some or all of the variables to be integer-valued. This allows us to code integer variables such as the number of items to produce in a given month as well as binary decision variables such as whether or not to offer a sale. Further, binary integer variables allow us to capture logical constraints such as A OR B and IF A THEN B, where A and B are themselves linear constraints. This added complexity allows integer programming to capture the optimization behind problems such as determining a new bus route, creating fair voter districts, and matching kidney donors and recipients. An application related to bike-sharing networks can be found in Section 6.5, and the reader is invited to formulate integer programs for problems related to vehicle routing and electric power systems in Exercises 1 and 2, respectively.

A linear program in which all variables are restricted to be integer-valued is called an *integer program* and one with both continuous- and integer-valued variables is called a *mixed integer program*. Adding in the integer constraint complicates the optimization procedure since the feasible region is no longer given by a convex polytope. However, note that if we removed the integer constraints we get a linear program that is a relaxation of the problem we can solve. If the optimal solution to this relaxation is integer-valued, then it will also be the optimal solution to the original integer program. (One setting in which the optimal solution is guaranteed to be integer-valued is when the vector b is integer-valued and the matrix A is *totally unimodular*.) This is exactly the idea behind branch-and-bound and cutting plane methods, two of the main algorithms used to solve integer programs. On each iteration, both algorithms solve a linear programming relaxation to the original problem. Either the relaxation yields an integer-valued optimal solution or the algorithm updates the relaxation by adding valid linear constraints. For more information about integer programming and linear optimization in general, see [47, 425].

6.2.3 Convex optimization

Linear programming is itself a subset of a larger class of polynomial-time solvable optimization problems called *convex programming*. Convex program-

[2]For more details about Newton's method and other numerical methods, see [363].

ming, or convex optimization, is where the bread and butter of machine learning algorithms lies, including linear regression (Section 4.3.1), lasso regression (Sections 4.7.2 and 8.5.4), and support-vector machines (Section 8.8). Returning to our more general optimization problem notation, we can write a convex program in the following format.

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) \\ &\text{subject to } g_i(\mathbf{x}) \leq 0 \quad \forall i = 1, 2, \ldots, m \end{aligned} \tag{6.4}$$

Such an optimization problem is a convex program if (1) the objective function $f(\mathbf{x})$ is a convex function and (2) the feasible region is a convex set. Note that we have switched to the standard minimization representation for convex programs rather than maximization. If each g_i is a convex function, then this is referred to as *standard form.* The program (6.5) gives an example of a convex program in standard form.

$$\begin{aligned} \text{minimize } & 0.5x_1^2 \\ \text{subject to } & x_1 + x_2 \leq 4 \\ & -\log(x_1) + x_2 \leq 0 \\ & x_1 \geq 0, x_2 \geq 0. \end{aligned} \tag{6.5}$$

Example Applications of Convex Optimization

1. Linear Regression: fit a linear prediction model to past data (Chapter 4).
2. Support-Vector Machines: find a partition of labeled data (Chapter 7).
3. Source-Location Problem: locate a source from noisy range measurements in a network of sensors [379].

A set $\mathcal{X} \subseteq \mathbb{R}^n$ is a *convex set* if for all $\mathbf{x_1}, \mathbf{x_2} \in \mathcal{X}$ and for all $0 < \alpha < 1$, $\alpha\mathbf{x_1} + (1-\alpha)\mathbf{x_2} \in \mathcal{X}$. In other words, the line segment between $\mathbf{x_1}$ and $\mathbf{x_2}$ is also in the set. Figure 6.4 shows that the feasible region of (6.5) is a convex set but adding the constraint $-x_1^2 + 6x_1 + x_2 \leq 9.5$ loses the convexity. A simple example for which we can easily show convexity is a half-space $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{a_i}^T\mathbf{x} \leq b_i\}$. To prove convexity of more complex sets we can either use the definition given above or use two nice properties of convex sets: (1) the intersection of convex sets is itself convex, and (2) if $\mathcal{X} \subseteq \mathbb{R}^n$ is a convex set then the affine transformation $\{A\mathbf{x}+\mathbf{b} \in \mathbb{R}^n \mid \mathbf{x} \in \mathcal{X}\}$ is a convex set. Already these allow us to prove that the feasible region of a linear program is a convex set, hence why we call the feasible region a convex polytope.

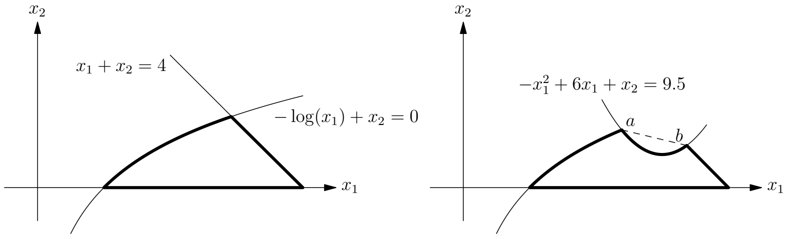


FIGURE 6.4: On the left, the convex feasible region of (6.5) is outlined in bold. Adding the inequality $-x_1^2 + 6x_1 + x_2 \leq 9.5$ yields the feasible region on the right, which is not convex since the line connecting a and b is not contained in the feasible region.

Given a convex set $\mathcal{X}$, a function $f : \mathcal{X} \to \mathbb{R}$ is a *convex function* on $\mathcal{X}$ if for all $\mathbf{x_1}, \mathbf{x_2} \in \mathcal{X}$ and $0 < \alpha < 1$,

$$f(\alpha\mathbf{x_1} + (1-\alpha)\mathbf{x_2}) \leq \alpha f(\mathbf{x_1}) + (1-\alpha)f(\mathbf{x_2}).$$

Figure 6.5 illustrates that the objective function of (6.5) is indeed convex. Some simple convex functions are linear functions $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$ and a quadratic function $f(x) = x^2$. The former implies that linear programs are a subset of convex programs. To prove convexity for more complex functions, we can again rely on convexity preserving operations:

1. if $f_1, f_2, \ldots, f_k$ are convex functions and $w_1, w_2, \ldots, w_k \geq 0$ are non-negative weights, then $f(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \ldots + w_k f_k(\mathbf{x})$ is convex,
2. if $\{f_a\}_{a \in \mathcal{A}}$ is a family of convex functions, then $f(\mathbf{x}) = \sup_{a \in \mathcal{A}} f_a(\mathbf{x})$ is a convex function, and
3. if $f(\mathbf{x})$ is a convex function, then $f(A\mathbf{x} + \mathbf{b})$ is a convex function.

In addition, if f is twice continuously differentiable, then f is convex if and only if its Hessian matrix[3] is positive semidefinite for all $\mathbf{x} \in \mathcal{X}$.

Like linear programs, convex programs exhibit strong structure, which allows us to solve them efficiently. First, the convexity of the feasible region and objective function imply that a locally optimal solution is a globally optimal one. Suppose we have a feasible solution $\mathbf{x}$ and optimal solution $\mathbf{x}^*$, then by the convexity of the feasible region, the line between $\mathbf{x}$ and $\mathbf{x}^*$ is contained in the feasible region. Further, by the convexity of the objective function, moving along this line is an improving direction.

Optimal Set for Convex Programming

In any convex program, a local optimal solution is globally optimal.

[3]The Hessian is the matrix of all second-order partial derivatives, $H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$, assuming all such derivatives are defined.

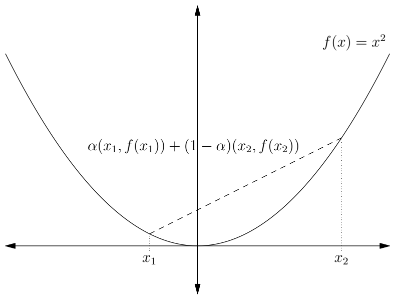


FIGURE 6.5: Example of a convex function.

First-order optimization methods

This is the intuition behind the simple, first-order-based methods for solving convex programs, first introduced by Shor [433]. If we ignore the constraints in (6.4) and assume that f is differentiable, then the problem becomes that of minimizing a differentiable, convex function, which we can easily solve using *gradient descent*. In gradient descent, in each step t we are at some point $\mathbf{x_t}$, and we find $\mathbf{x_{t+1}}$ by moving in the direction of the negative of the gradient $\mathbf{x_{t+1}} = \mathbf{x_t} - \eta_t \nabla f(\mathbf{x_t})$. By setting the step size η_t appropriately, the solution converges to an optimal solution. This convergence can be improved using Newton's method to approximate f at each point $\mathbf{x_t}$. However, there are two things that become difficult when applying gradient descent or Newton's method to a convex program: (1) we need to maintain feasibility, and (2) the function f is not differentiable in many common data science settings.

We tackle the latter problem first. If f is not differentiable, then instead of using the gradient to obtain a linear approximation for f, we will instead use a *subgradient* to find a direction in which to move. A subgradient of f at $\mathbf{x_t}$ is a vector $\mathbf{v_t}$ such that

$$f(\mathbf{x}) - f(\mathbf{x_t}) \geq \mathbf{v_t} \cdot (\mathbf{x} - \mathbf{x_t})$$

for all $\mathbf{x}$. In other words, $\mathbf{v_t}$ gives us a lower bound for f that is equal at $\mathbf{x_t}$. For example, $v \in [-1, 1]$ is a subgradient for $f(x) = |x|$ at $x = 0$. See Figure 6.6. In *subgradient descent*, the algorithm uses this subgradient to decide which direction to move in. In particular, at each time step t with current solution $\mathbf{x_t}$, the algorithm sets $\mathbf{x_{t+1}} = \mathbf{x_t} - \eta_t \mathbf{v_t}$, where η_t is the step size. If f is differentiable, then $\mathbf{v_t}$ is the gradient and this is equivalent to steepest descent, which is slower to converge than Newton's method but still finds an optimal solution. In the general case when f is not differentiable, since $\mathbf{v_t}$ is a subgradient, the solution value is not guaranteed to improve at each step. However, as long as the sequence η_t satisfies that $\eta_t \to 0$ and $\sum_t \eta_t = \infty$, then the best solution seen also converges to an optimal solution. There exist convergence proofs for other choices of η_t as well.

The subgradient method detailed so far does not deal with feasibility constraints. In order to add these in, the *projected subgradient method* maintains

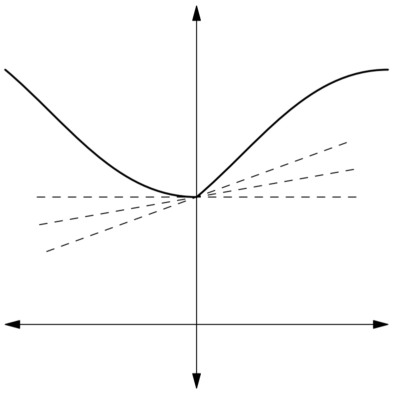


FIGURE 6.6: Example of several subgradients (dashed) for a function (solid) at $x = 0$.

a feasible solution by projecting the solution $\mathbf{x_{t+1}}$ back to the feasible region. This projection can be efficiently done for many different types of convex sets, and as long as the sequence η_t satisfies that $\eta_t \to 0$ and $\sum_t \eta_t = \infty$, then the best solution seen also converges to an optimal solution. The simplicity of subgradient methods makes them appealing to implement in practice despite being less efficient than the interior-point methods discussed in Section 6.2.3.1. In addition, these methods can be run on large problem instances since they are not very memory-intensive.

One way to improve the runtime on larger data is using *stochastic gradient descent*. Many objective functions in machine learning can be written as a sum of functions $f(\mathbf{x}) = \frac{1}{n}\sum_i f_i(\mathbf{x})$, where each f_i corresponds to some loss function on the i^{th} observation measuring how well the model matches that observation. Rather than evaluating the gradient or subgradient for every f_i, stochastic gradient descent samples a random subset. Assuming some properties of the f_i's, this sample gives a good estimate for the overall fit of the model for large datasets. We will see uses of stochastic gradient descent in Chapters 8 and 9.

6.2.3.1 Duality and optimality conditions

As with linear programs, we can use duality to give more structure about the set of optimal solutions and develop more efficient algorithms. Associating a Lagrangian multiplier y_i with each inequality constraint yields the following dual program.

$$\begin{aligned} \text{maximize } \mathcal{L}(\mathbf{y}) &:= \inf_{\mathbf{x}\in\mathbb{R}^n}\left[f(\mathbf{x}) + \sum_{i=1}^{m} y_i g_i(\mathbf{x})\right] \qquad (6.6)\\ \text{subject to } &\mathbf{y} \geq \mathbf{0} \end{aligned}$$

The objective function in (6.6) can be interpreted as finding the tightest lower bound for (6.4). It is easy to show that convex programs satisfy *weak duality*. That is, for all feasible $\mathbf{x}$ to (6.4) and feasible $\mathbf{y}$ to (6.6), $f(\mathbf{x}) \geq \mathcal{L}(y)$. However, unlike linear programs, *strong duality* only holds for convex programs that satisfy *regularity conditions*. While there exist several possible ways to show regularity, one simple test is to show that there exists feasible $\mathbf{x}$ such that $g_i(\mathbf{x}) < 0$ for all $i = 1, 2, \ldots, m$. Such a point $\mathbf{x}$ is called a strictly feasible point and this condition is called the *Slater condition*. If the convex program satisfies regularity and both (6.4) and (6.6) have a nonempty feasible region, then there exist feasible $\mathbf{x}^*$ and $\mathbf{y}^*$ such that $f(\mathbf{x}^*) = \mathcal{L}(\mathbf{y}^*)$. This implies that if we find feasible $\mathbf{x}$ and $\mathbf{y}$ with matching objective function values then we know both are optimal for their respective optimization problems. Luckily, most reasonable convex programs satisfy these regularity conditions.

The dual program also yields the corresponding Karush-Kuhn-Tucker (KKT) conditions of optimality. These conditions are necessary and sufficient for optimality in most convex settings. For example, the conditions are necessary and sufficient if the convex program (6.4) is in standard form and satisfies the regularity conditions described above.

KKT Conditions for Convex Programming

Consider any convex program (6.4) in standard form that satisfies regularity conditions and such that f and every g_i is continuously differentiable at some point $\mathbf{x} \in \mathbb{R}^n$. Then, $\mathbf{x}$ is an optimal solution if and only if there exists $\mathbf{y} \in \mathbb{R}^m$ such that

1. $\nabla \mathcal{L}(\mathbf{x}) = 0$ (stationarity),
2. $\mathbf{x}$ is feasible for (6.4) (primal feasibility),
3. $\mathbf{y} \geq 0$ (dual feasibility), and
4. $y_i g_i(\mathbf{x}) = 0$ for all $i = 1, 2, \ldots, m$ (complementary slackness).

For optimality conditions for nondifferentiable functions, see [221, 359].

Interior-point algorithms

Again, we can use these optimality conditions to understand the intuition behind some of the most common algorithms used to solve convex programs. The primal-dual interior-point method for linear programming in Section 6.2.2.1 can be extended to convex programming and is one of several interior-point methods for convex optimization [360]. As in linear programming, the goal of this interior-point method is to maintain a primal feasible solution $\mathbf{x}$ and a dual feasible solution $\mathbf{y}$ and work towards satisfying complementary slackness in the KKT conditions. The algorithm starts at a feasible solution $(\mathbf{x}, \mathbf{y})$, and on each iteration, the algorithm moves towards a new feasible point $(\mathbf{x}', \mathbf{y}')$ such that $y_i g_i(\mathbf{x}) \approx \tau$, where τ decreases to zero as the algorithm progresses. The algorithm takes a step towards this point via Newton's method. In general, interior-point methods are very efficient for convex programs provided that the constraints and objective function fall into some common paradigms [59]. For more details on the theory of convex optimization, see [32, 59].

6.2.4 Non-convex optimization

Last, we move to our most general setting, non-convex optimization, in which we allow the objective function and constraints to take on any form. For example, consider the following optimization problem.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0,1\} \quad \forall i = 1, 2, \ldots, n. \end{aligned} \tag{6.7}$$

This is the classic *knapsack problem*. In this problem, we are given n items each with weight w_i and value v_i, and the goal is to choose the maximal value set of items with total weight $\leq W$. Despite the simplicity of this problem, there is no polynomial-time algorithm to solve it. However, if we relax the integer constraint and are allowed to fractionally choose items, then it is simple to find an optimal solution. Non-convexity in machine learning problems arises from a similar problem. Many non-convex machine learning problems are problems in high-dimensional settings in which it is necessary to add some constraint on the complexity or sparseness of a solution in order to avoid overfitting. Further, neural networks are designed to capture complex relationships between variables and are inherently non-convex. Even the relatively simple objective function $\min_x \mathbf{x}^T Q \mathbf{x}$ is non-convex and difficult to solve.

Example Applications of Non-Convex Optimization

1. k-Means Clustering: partition data into k similar clusters (Chapter 5).
2. Neural Networks: fit a highly nonlinear prediction model (Chapter 9).
3. Designing Bus Routes: use traffic data to optimally route students to school [39].

Without convexity, the algorithms described so far fail to work. Under regularity conditions, an optimal solution still satisfies the KKT conditions.

However, these conditions are no longer sufficient or as informative. Without a convex feasible region, the improving direction proposed may not be feasible, and without a convex objective function, there may exist local minima or saddle points that satisfy the KKT conditions but are not globally optimal. In fact, a polynomial-time algorithm that finds an optimal solution for any non-convex problem is unlikely to exist.[4] Therefore, in order to solve a non-convex problem, we have a few options: imposing additional structure that allows us to optimally solve the problem, settling for a non-optimal solution, or finding the optimal solution inefficiently. This last option is only realistic for very small problems, and is not used for many data science questions.

Let's consider the option of imposing additional structure. First, we can try to relax a non-convex optimization problem to a convex one. If the optimal solution to the relaxed problem is feasible for the original problem, then it is the optimal solution overall. Otherwise, the optimal solution value to the relaxed problem still gives us a bound on the optimal solution value for the original problem. We can then try to map to a feasible solution and measure the loss in objective function, or we can try to strengthen our relaxation. If we cannot obtain a direct relaxation, then we can also consider a convex approximation.

As an example, consider lasso regression, introduced in Section 4.7.2. Lasso regression is motivated by a common preference among statisticians for *parsimonious* regression models, in which relatively few predictor variables have nonzero coefficients. Ideally, a constrained optimization framework could achieve such parsimony by minimizing the sum of squared errors subject to the non-convex constraint, $||\beta||_0 \leq k$. (k is a tunable *hyperparameter* which reflects how severely we wish to constrain the parsimony of the model. The tuning of hyperparameters is discussed in greater detail in Section 6.3.3.2.) Due to the complexity of solving this non-convex optimization problem, lasso regression can be seen as the convex approximation of optimal feature selection where the non-convex constraint $||\beta||_0 \leq k$ is replaced with the convex constraint $||\beta||_1 \leq k'$. Using Lagrange multipliers then yields the form for lasso regression given in Chapter 4.

Alternatively, we can try to exploit the structure of our specific non-convex problem. For certain non-convex problems, there exist efficient problem-specific algorithms that find an optimal solution or the problem can be cleverly reformulated to an easier format. As an example, several image segmentation problems such as separating foreground and background can be reformulated as linear programs [159, 432]. Additionally, some algorithms designed for convex optimization problems can be extended to restricted non-convex settings. Projected gradient descent has been shown to converge to an optimal solution if the problem constraints fall into one of a few patterns and the objective function is *almost* convex [246].

[4]The existence of such an algorithm would imply that P = NP [176].

In the other direction, we can settle for a non-optimal solution by using heuristic algorithms that are not guaranteed to return an optimal solution. One such simple algorithm is a grid search: search over many possible feasible solutions by gridding the range for each variable and return the best feasible solution found. Or, better yet, we could iterate over many possible feasible solutions and use these as different initial starting points for gradient descent. While in each iteration we are not guaranteed to find the global optimal solution, we hope that by starting from many different points in the feasible region that we will stumble upon it anyway. This general algorithm idea to start at a feasible solution and search for improvements is called *local search*. If the variables in our optimization problem are restricted to be integer-valued, there exist other appropriate local search algorithms such as genetic algorithms or tabu search that we can apply. Exercise 1 includes designing and evaluating a local search algorithm for a vehicle routing problem. For a more detailed exposition of non-convex optimization and its applications in data science, we direct the reader to [246, 440].

6.3 Simulation

Up until now, the methods described in this chapter have all been deterministic; we've assumed that the problem at hand can be formulated as an optimization problem with known objective function and constraints. However, uncertainty lies at the core of data science. How do we describe complex relationships between random variables? How do we predict a random outcome on the basis of known realizations of random variables, and how do we measure our uncertainty in that prediction? How do we prescribe decisions that are robust to uncertainty?

An important operations research tool for making decisions under uncertainty is *simulation*. In its simplest sense, simulation is the generation of realizations of a random variable or process.

Example Applications of Simulation

1. Air Traffic Flow Management: Improve the utilization of the national airspace by testing policies in an agent-based simulation [249].

2. Ranked Data Analysis: Cluster survey respondents based on partial rankings of preferences using a Gibbs sampler [244].

3. Stochastic Optimization: Compute approximate value functions by simulating the effects of given policies [393].

We first describe the key probability principles on which simulation relies and fundamental techniques for generating random variables. We then provide specific examples of the ways in which simulation underlies some machine learning methods and can be used more broadly as a tool for prescriptive analytics.

6.3.1 Probability principles of simulation

Chapter 4 introduced the notion of a *random variable*. Formally, a random variable, X, is a mapping from the sample space of a random experiment to a real number. For example, consider the experiment of flipping a fair coin five times. The sample space of the experiment is the set of all possible combinations of outcomes for the five coin flips. If we denote flipping heads as H and flipping tails as T, one outcome in the sample space is (H, H, H, T, T), which represents flipping heads on the first three flips and tails on the last two. If we let X denote the number of heads flipped, then X is a mapping from the sample space of outcomes of five coin flips to the numbers $0, 1, \ldots, 5$. For the particular realization (H, H, H, T, T), $X = 3$.

For most purposes, a random variable X is classified as being either *discrete*, meaning it takes on values in a countable set, or *continuous*, meaning it takes on values over a real interval. A random variable is characterized by its *probability distribution*, which specifies the relative likelihoods of the possible values of X. The distribution of a discrete random variable X is given by its *probability mass function* (PMF), $p(x)$, which specifies for any value x the probability of X equaling that value: $p(x) = P(X = x)$. For a continuous random variable, the probability of X exactly equaling any value is 0, so we instead refer to its *probability density function* (PDF), $f(x)$, which is defined so that

$$P(a \leq X \leq b) = \int_a^b f(x)dx.$$

Equivalently, a random variable X is uniquely specified by its cumulative distribution function (CDF), $F(x) = P(X \leq x)$. For a discrete random variable having probability mass function $p(x)$, the CDF is given by

$$F(x) = P(X \leq x) = \sum_{y \leq x} p(y).$$

For a continuous random variable having probability density function $f(x)$, the CDF is given by

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(y)dy.$$

We can use the CDF of a random variable to determine *quantiles*, or *percentiles*, of the distribution. Let the distribution of random variable X be defined by CDF F. A value x is the q^{th} quantile of F if $F(x) = \frac{q}{100}$. That is,

x is the value such that random variable X takes on a value no greater than x with probability $q\%$. A common quantile of reference is the *median*, which corresponds to the 50^{th} quantile, or percentile, of the distribution.

Because the CDF defines a probability, it is bounded below by 0 and above by 1. It is also monotonically nondecreasing, which is a useful property for simulation, as we describe in the next section.

Two other important characteristics of a random variable are its *expected value* (also referred to as *mean*) and its *variance.* By the Strong Law of Large Numbers, the expected value is the limiting value of the *sample average* of n observations of the random variable, $\frac{x_1+x_2+\ldots x_n}{n}$, as n goes to infinity. For discrete random variables, it is defined as

$$E[X] = \sum_x xp(x),$$

that is, the weighted sum of all possible values the random variable can take on times the likelihood of that value. The analogous definition for continuous random variables is

$$E[X] = \int_{-\infty}^{+\infty} xf(x)dx.$$

As these formulae suggest, the expected value of a random variable can be thought of as the center of mass of its probability mass or density function.

The variance reflects the spread of the random variable's distribution about its expected value. It is defined as

$$\sigma^2 = E[(X - E[X])^2].$$

Large variance indicates great uncertainty about the random variable's value, while small variance indicates that the random variable tends to take on values very close to its expected value. Variance reflects squared deviations about the expected value, but it can also be useful to measure variability in the original units of the random variable. For this, we use the square root of the variance, known as the *standard deviation.*

6.3.2 Generating random variables

At its essence, simulation concerns itself with generating random variables, also known as *Monte Carlo simulation.* We now describe common techniques for generating random variables from a known distribution, from an empirical distribution (also known as bootstrapping), and from an unknown distribution (using Markov Chain Monte Carlo (MCMC) methods).

6.3.2.1 Simulation from a known distribution

When the distribution of X is known or assumed, simulation seeks to generate a sequence of realizations from that distribution. That is, given a cumulative distribution function F, we wish to generate a sequence of values $X_1, X_2, \ldots$ such that $P(X_i \leq x) = F(x)$. There are many techniques for simulating random variables from a known distribution, with increasingly sophisticated techniques required for simulating jointly distributed random variables.

Choosing a distribution

The first step to any simulation is choosing which probability distribution to simulate. The modeler must make assumptions about the nature of the random variables being generated. For example, if the simulation is modeling Internet traffic as an arrival process, the Poisson distribution is a common choice for modeling the number of arrivals to occur in a given time interval. As another example, a triangular distribution is a useful and simple distribution for modeling a continuous random variable having an assumed minimum and maximum possible value and a likely modal value (forming the peak of the triangle). As is common in modeling, the modeler must make a trade-off between model fidelity and simplicity: a modeler might prefer to use a simpler distribution that is more tractable and interpretable than a distribution that captures more system complexity.

Most mathematical computing environments have built-in functions for generating random variables from common univariate and multivariate distributions. The availability of built-in random variable generators streamlines the computation associated with simulation. Eight common discrete and continuous distributions are given in Tables 6.1 and 6.2. For each distribution, its probability mass (density) function is displayed, as well as a summary of common uses and the R command used to generate values from this distribution. (The book website provides comparable commands in other languages such as Python, Matlab, and Julia).

When a sample of data is available, the modeler can hypothesize a distribution on the basis of a histogram or other graphical summary, and from there assess the *goodness-of-fit* between the data and the hypothesized distribution. The sample mean and sample variance can also be computed to provide information about the true mean and variance of the distribution. Some common goodness-of-fit tests are the Kolmogorov-Smirnov test, the Cramér-von Mises criterion, and the Anderson-Darling test. For discrete distributions, the χ^2 goodness-of-fit test is also common. If the hypothesized distribution fits the sample data well, then the modeler can feel comfortable using that distribution in the simulation.

Inverse method

Absent a built-in function, the modeler might need to write their own method for generating random variables. The simplest technique that is useful for generating a random variable from a univariate distribution is called the *inverse method*. Let F^{-1} be the inverse of the CDF. Thus, F^{-1} provides, for a desired quantile, the value of the random variable associated with that quantile in distribution F. In other words, if $P(X \leq x) = q$, then $F^{-1}(q) = x$. Because the quantiles of any distribution are, by definition, uniformly distributed over

TABLE 6.1: Common discrete probability distributions: uniform, binomial, geometric, and Poisson. See Table 6.2 for common continuous distributions.

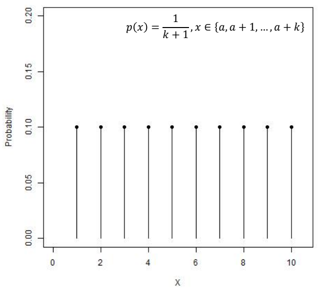
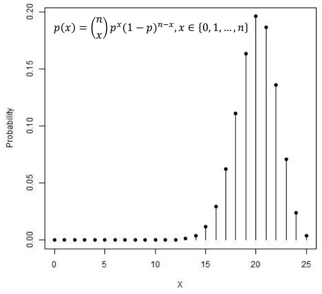
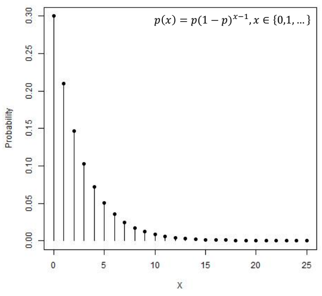
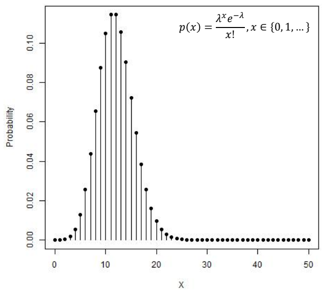
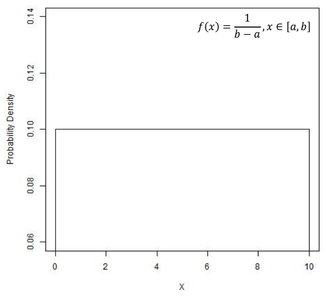
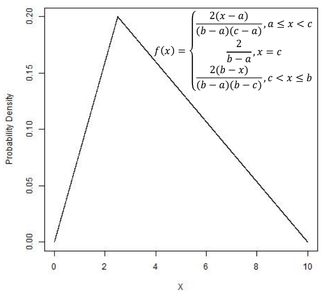
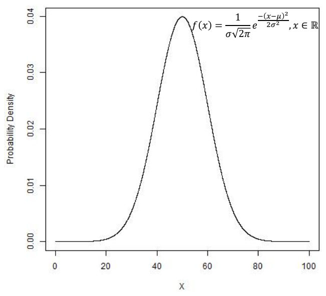
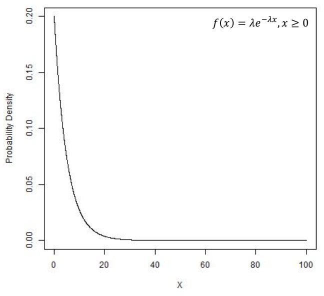
Discrete uniform  **Common uses**: Equally likely discrete integer outcomes from a to $a+k$, no prior knowledge of distribution **In R:** sample	**Binomial**  **Common uses:** Number of successes in n independent trials, where each trial is successful with probability p **In R:** rbinom
Geometric  **Common uses:** Counting the number of independent trials until the first success occurs, where each trial is successful with probability p **In R:** rgeom	**Poisson**  **Common uses:** Counting the occurrence of rare events having mean λ, queuing theory **In R:** rpois

TABLE 6.2: Common continuous probability distributions: uniform, triangle, normal, and exponential. See Table 6.1 for common discrete distributions.

Uniform  **Common uses:** Equally distributed likelihood of outcomes over interval $[a, b]$, no prior knowledge of distribution except for range **In R:** runif	**Triangle**  **Common uses:** No prior knowledge of distribution except for range $[a, b]$ and mode c **In R:** rtriangle
Normal  **Common uses:** Random errors, sums of random variables, scientific measurements, having mean μ and variance σ^2 **In R:** rnorm	**Exponential**  **Common uses:** Spacing between rare events occurring with rate λ **In R:** rexp

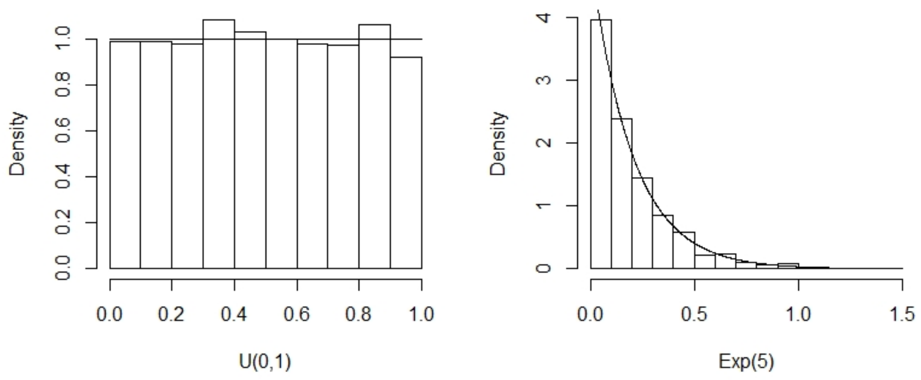


FIGURE 6.7: On the left, histogram of 1000 randomly generated $U(0, 1)$ random variables with density of the $U(0, 1)$ distribution overlaid. On the right, histogram of $X = -\frac{\ln(1-U)}{\lambda}$, where $\lambda = 5$, computed using the 1000 $U(0, 1)$ random variables from the left panel, with the $Exp(5)$ distribution overlaid. (Plots produced using base plotting package in R.)

$[0, 100]$, any random variable can be generated by first generating a number, U, uniformly over $[0, 1]$ (corresponding to the $100U$ quantile) and then returning $X = F^{-1}(U)$, assuming a closed-form expression for F^{-1} can be defined.

As an example, consider the exponential distribution, a continuous distribution having PDF $f(x) = \lambda e^{-\lambda x}, x \geq 0$ for a given parameter $\lambda > 0$, known as the *rate* of the distribution. (The exponential distribution is commonly used for modeling the time between occurrences of rare events, and λ represents the rate of occurrence per unit time.) Integrating $f(x)$ we find the CDF of the exponential distribution to be $F(x) = 1 - e^{-\lambda x}, x \geq 0$. To generate exponential random variables, we first invert F by solving $F(X) = U$ in terms of X. We find $X = F^{-1}(U) = -\frac{\ln(1-U)}{\lambda}$. Thus, given a $U(0, 1)$ random variable, we can generate X from an exponential distribution in this way. The left panel of Figure 6.7 shows a histogram of 1000 randomly generated values uniformly distributed from zero to one ($U(0, 1)$). The right panel shows a histogram of 1000 exponentially distributed random variables with rate parameter $\lambda = 5$ that were generated from the $U(0, 1)$ using $X = -\frac{\ln(1-U)}{\lambda}$.

Because a closed-form expression for F^{-1} is often not available, methods such as *acceptance-rejection* or distribution-specific methods can also be used. The reader is referred to [304, 410] for a detailed treatment of this subject.

6.3.2.2 Simulation from an empirical distribution: bootstrapping

Sometimes a hypothesized distribution is not apparent from the problem context or from the data sample. For example, a histogram of observed data might not resemble any of the distributions given in Tables 6.1 and 6.2. In such a case, the modeler might prefer to use the data's *empirical distribution.* As explained in Section 4.4.3, the empirical distribution is the distribution exhibited by the sample of data itself, where the CDF, $P(X \leq x)$, is approximated by the percentiles observed in the sample. The modeler can sample directly from the (discrete) empirical distribution using a process called *bootstrapping.* In this process, realizations of the random variable are selected uniformly at random, with replacement, from the data in the sample. In this way, the distribution of the generated values mirrors the distribution observed in the raw data.

Bootstrapping is especially useful when the true distribution is unknown and

1. there is little to no intuition as to what a reasonable hypothesized distribution would be,
2. the distribution is complicated to derive because it arises from a nontrivial composition of underlying random variables, or
3. small sample sizes preclude assessing the goodness-of-fit of a hypothesized distribution.

A danger in bootstrapping is that bootstrapped samples can be heavily influenced by the presence or absence of extreme values in the original sample. If the original sample is so small that rare, extreme values did not occur in the sample, the bootstrapped sample, regardless of size, will not exhibit extreme values. On the other hand, if an extreme value occurs in a small original sample, then it is possible this extreme value will be oversampled, relative to the true distribution, in the bootstrapped sample. This can pose a problem if the generated values are used in calculations that are sensitive to outliers, or could result in the simulated sample not accurately representing the real system.

We will revisit bootstrapping in Section 6.3.3, which focuses on the role of simulation techniques in statistical and machine learning.

6.3.2.3 Markov Chain Monte Carlo (MCMC) methods

There are other times when we wish to generate random variables that reflect the behavior of a known system, but for which no probability distribution for the random variables is known and no representative sample of data can be found to approximate the distribution. In situations such as these, Markov chain Monte Carlo methods (MCMC) can be a powerful tool.

First, we give some background on Markov chains. A *discrete-time Markov chain* is a sequence of states experienced by a system in which the *transition* *probability* p_{ij}, of moving to state j given that the system is currently in state i depends only on the state i and not on any prior states visited. For instance, in a random walk along the integer number line, the state of the system moves either $+1$ or -1, with a probability that depends only on the current state. When the Markov chain is *ergodic*,[5] the system achieves steady-state, and a *stationary distribution* describing the long-term, limiting probability of being in a particular state can be either computed analytically or estimated via simulation.

MCMC permits us to simulate from a probability distribution that we are unable to derive or describe by defining a Markov chain sharing the same state-space as the desired probability distribution and whose steady-state distribution *is* the desired distribution. Thus by simulating the Markov chain for a long period of time and returning the last state visited, we can generate random variables from (approximately) the desired distribution.

These ideas extend to generating from continuous probability distributions, but for ease of exposition, we will illustrate two common algorithms for MCMC simulation using discrete distributions as examples.

Metropolis-Hastings algorithm

The challenge in MCMC is to construct a Markov chain whose limiting distribution will be the same as the desired distribution. The Metropolis-Hastings algorithm does just this. Let π be the desired probability distribution. We wish to generate a sequence of states $X_0, X_1, \ldots$ that form a Markov chain having stationary distribution π. Let M be the *proposal chain*, an irreducible Markov chain sharing the same state space as the desired distribution. Let M_{ij} denote the transition probability from state i to state j in Markov chain M. When the proposal chain is in state i, a proposal state j is generated with probability M_{ij}. The proposal state is *accepted* according to an acceptance threshold $a_{ij} = \frac{\pi_i M_{ji}}{\pi_j M_{ij}}$. If $a_{ij} \geq 1$, then state j is accepted automatically. If $a_{ij} < 1$, then state j is accepted with probability a_{ij}. If it is accepted, then $X_{n+1} = j$; otherwise $X_{n+1} = i$. The sequence $X_0, X_1, \ldots$ generated in this fashion can be shown to be an irreducible Markov chain with stationary distribution π. Because the X_i are not independent, it is not appropriate to use the sequence directly as a random state generator. Instead, we run the chain for a sufficiently long time (known as the *burn-in period*) so that the current state of the system is largely independent of the initial conditions, and select one value from this chain. We then start over from scratch to generate the second value, and so on. For details of the proof, several practical applications, and accompanying R code, the reader is referred to [130].

Gibbs sampler

Another important MCMC technique is the *Gibbs sampler* for generating samples from joint multivariate distributions in which the distribution of each

[5]To be ergodic, a Markov chain must be *irreducible* (every state can reach any other state), *aperiodic* (from a given state i, returns to that state are not restricted to a fixed multiple of time steps), and *recurrent* (every state will eventually be revisited).

variable conditioned on all the others is known. The Gibbs sampler iteratively samples a value for one of the variables conditioned on the current values of the remaining variables, cycling in turn through all of the variables. After the burn-in period, the vectors of samples will reflect the true distribution [213]. Casella and George provide a useful tutorial [85].

6.3.3 Simulation techniques for statistical and machine learning model assessment

Now that we understand *how* to generate random variables from a probability distribution, we can return to the applications promised at the start of Section 6.3. We highlight two important considerations in any predictive modeling endeavor for data science.

- **Model Selection**: From many possible models and modeling frameworks, how do we choose which is the best, and which predictor variables should be included?
- **Model Assessment**: How suitable is a given model at describing the observed relationships between the variables and at predicting future observations?

Simulation plays an important role in model selection methods such as random forests, boosting, and bagging, which will be covered in Chapter 8. Simulation also plays an important role in assessing a model's true predictive power. We will describe two simulation techniques that commonly arise in model assessment, bootstrapping confidence intervals for estimated parameters in a linear regression model and cross-validation. Both of these topics were introduced in Chapter 4 and will be discussed later in Chapter 8. We include these topics here to highlight the role of simulation.

6.3.3.1 Bootstrapping confidence intervals

A common question in any data analysis is whether apparent patterns in the data are the result of relationships between observed or latent variables (*signal*) or instead are meaningless abnormalities arising due to chance (*noise*). Recall from Section 4.5 that in traditional frequentist statistics, such as ordinary least-squares regression, a *null hypothesis* representing a default prior belief is compared against an *alternative hypothesis*. A *test statistic* is computed from sample data and is evaluated as being either consistent or inconsistent with the null hypothesis. This determination is made by requiring a low *Type I error rate*, which is the probability of falsely rejecting the null hypothesis, that is, falsely asserting that the pattern in the data is real when it is actually due to noise. Underlying each of these steps are simple distributional assumptions associated with the null hypothesis (often related in some way to the normal distribution) that permit evaluation of the test statistic's statistical significance.

This methodology can break down when simple distributional assumptions are not appropriate or cannot be evaluated. In this case, the bootstrap can come to the rescue! The data are resampled (i.e., simulated from the original dataset) using bootstrapping, and the desired test statistic is calculated separately for each bootstrapped sample, yielding an empirical distribution to which the test statistic calculated on the entire dataset can be compared and a p-value computed.

6.3.3.2 Cross-validation

Section 4.3.2 introduced the technique of *cross-validation*, in which the model is iteratively fit on a portion of the data and tested on a left-out portion of the data. Cross-validation is important for estimating the prediction error of the model to assess the degree of overfitting, and for the tuning of hyperparameters. We start by motivating both of these ideas.

Even if the true relationship between a response variable and the predictor variables were perfectly known, the response variable would still exhibit variability due to environmental factors or measurement error. Some of this variability is caused by the inherent functional relationship between the response and the predictor variables (signal), and it is this relationship in which we are interested. However, some of the variability is due to inherent noise that is not representative of patterns likely to occur in future data points. Just as any n (x, y) pairs (with distinct x values) can be perfectly matched to an $(n-1)$-degree polynomial, an overly complex model can seemingly "fit" the variability observed in a finite sample of data. However, this complex model is unlikely to yield accurate predictions of future data points. Recall from Chapter 4 that we call this phenomenon *overfitting*, and it is characterized by models that achieve a deceptively low *training error rate* on the data used to fit, or train, the model, but high *testing* or *prediction error rates* on new data. Once again, we are faced with the important trade-off between model complexity (i.e., the ability to accurately represent high-dimensional relationships in data) and model parsimony (the ability to extract simple relationships between variables that are robust to the variability in data).

Moreover, in addition to modeling the relationship between a response and predictor variables, many statistical and machine learning models incorporate *hyperparameters*. A hyperparameter is a parameter that permits refinement of the performance of an algorithm used for fitting a model. For example, in the lasso method to regularize a regression model (Sections 4.7.2 and 6.2.4), the tolerance constraint parameter k' that bounds $||\beta||_1$ is a hyperparameter. It does not appear in the model describing the relationship between the response and the predictor variables, but it is used by the algorithm to determine the model parameters. We generally do not know *a priori* the value of k' to use. Instead, we must iteratively vary k' to identify the value that results in a model having lowest prediction error.

Cross-validation is a valuable technique for estimating prediction error (thereby avoiding overfitting) and for tuning hyperparameters. One commonly used method of cross-validation is *k-fold cross-validation.* In k-fold cross-validation, the dataset is divided randomly into k equally-sized subsets of observations, called *folds.* Using only $k-1$ of the folds, with one fold set aside, candidate models are identified and fitted to the data. The prediction error is then estimated on the left-out fold. The process is repeated k times, with each fold set aside exactly once and the model identified and trained on each possible combination of $k-1$ folds. The k estimates of prediction error on the left-out folds are then averaged and used as an estimate of the testing error of the model. When used in conjunction with hyperparameter tuning (e.g., determining the ideal order of polynomial in a linear model, or the lasso regularization parameter k'), the value of the hyperparameter is chosen to minimize the average testing error over the k folds. Once this near-optimal parameter value is determined, the final model can be fit using the entire dataset. A typical value of k is five or ten folds, which trades off *bias* for *variability.* Bias can be thought of as the model's dependence on the data on which it was trained. Variability refers to the variance in the prediction error. Each model is fit to a fraction $\frac{k-1}{k}$ of the data, and the prediction error is estimated on a fraction $\frac{1}{k}$ of the data. Thus, using a larger number of folds decreases the bias of the model but increases the variability of the prediction error from fold to fold. Using a smaller number of folds yields lower variability in the prediction error, but potentially higher bias estimates. Yet again, we see the trade-off between complexity and parsimony.

The process of selecting the folds for cross-validation is itself a simulation. The rows of data are randomly assigned to one of the k folds. Some tools, such as R's `caret` package, will try to create *balanced* folds that each represent the distribution of the predictor variables in the full dataset, but there is still randomization involved. Another data scientist running cross-validation on the same data might end up with different results (and a different end model) due to the random process of assigning observations to folds. For this reason, it is very important to set a consistent random seed before any process involving random number generation. Doing so ensures the reproducibility of your work by others and by your future self!

6.3.4 Simulation techniques for prescriptive analytics

To this point, we have discussed simulation's role under the hood of common statistical and machine learning methods. However, simulation is a powerful approach in its own right for prescriptive analytics. We describe two common simulation approaches that, on their own, are effective tools for guiding decision-making in stochastic (i.e., random) systems, discrete-event simulation and agent-based modeling. We then describe how each can be used effectively for prescriptive decision-making.

6.3.4.1 Discrete-event simulation

When modeling complex systems, such as the emergency department of a large urban hospital, the rapidly changing temporal dynamics must be incorporated into strategic (long-term decisions concerning infrastructure and management structure), tactical (medium-term decisions concerning scheduling and logistics), and operational (short-term decisions concerning in-the-moment allocation of resources) decision-making. In *discrete-event simulation*, a system evolves over time as discrete events arise stochastically to trigger decision points [304, 410].

A simple example of discrete-event simulation is a queue at a bank. The discrete events punctuating the system are the arrivals of customers into the bank, the commencements of service with the bank teller, and the completions of service. Knowing the times at which these events occur is sufficient for understanding the state of the system (e.g., how many people are in line, how long they have waited, and how many people are being served) at any point in the entire time horizon. Thus, a discrete-event simulation can model some complex systems very compactly.

To create a discrete-event simulation, one must specify the probability distributions governing the time between each event. In the case of the bank, we need to identify a probability distribution describing the time between arrivals of new customers and a probability distribution describing the duration of service. These two inputs are sufficient to model the bank's queuing system, and the methods of Section 6.3.2 can be used to do this. If past data are available, they can be used to inform the selection of distributions. An example discrete-event simulation for a bike-sharing system is detailed in Section 6.5.

6.3.4.2 Agent-based modeling

The purpose of an agent-based model is to understand how the interaction of independent entities (agents) making individual decisions will result in collective dynamics in a system. In such a simulation, each individual is assigned a simple set of rules governing their behavior in the system. The simulation then executes the rules for each individual and tracks the state of the overall system [304].

As an example, agent-based simulation can be used to model highway traffic dynamics. Each vehicle follows a simple set of rules governing its speed (selected as a random variable from a provided distribution), risk threshold, and traffic regulations. The dynamics of the entire highway can then be modeled as the interaction of these individual vehicles on the roadway.

To create an agent-based simulation, one must specify the classes of individuals present (e.g., fast versus slow drivers, risky versus cautious drivers, commercial trucks versus private vehicles). One must determine a probability distribution for membership in the class and other distributions for the parameters governing attributes in that class. Then each class must be assigned a set of rules. These should all be informed by data, when available. Here again, the complexity-tractability trade-off emerges: the beauty of agent-based modeling is that often very simple rules of behavior can elicit complex system dynamics. The job of the modeler is to define the rules that are sufficient to reliably describe the system. See Section 6.5 for an application of agent-based modeling to HIV prevention.

6.3.4.3 Using these tools for prescriptive analytics

Discrete-event and agent-based simulations can *describe* an existing system and *predict* the effects of changes made to the processes within the system. For instance, a simulation of our example bank could permit us to investigate whether positioning a greeter at the entrance to the bank can help route people more quickly to the correct teller to reduce waiting time. An agent-based simulation of a highway could allow us to investigate whether new signage can reduce accidents at a highway interchange.

As *prescriptive* tools, these simulations allow us to weigh trade-offs between different choices to choose the "best" decision. One technique for doing this is known as *what-if analysis.* As its name suggests, the operations researcher identifies several possible decisions that could be made and for each possible decision, uses the simulation to quantify the impacts to the system if that decision were adopted. Amongst the decisions tested, the analyst can choose the best.

Another option is to embed the outputs of the simulation as inputs to one of the optimization frameworks described earlier in the chapter. Mathematical programming problems require estimation of parameters that can quantify the costs, resource use, profits, etc. associated with the decision variables in the model. Simulation is a tool that can assist with such parameter estimation.

Yet another set of tools integrates simulation into an optimization framework and is aptly named *stochastic optimization.* This set includes the technique popularly referred to as *reinforcement learning.* We discuss this class of techniques in the next section.

6.4 Stochastic optimization

Section 6.2 described several classes of optimization problems in which all of the inputs and the system state are assumed to be deterministic and known with certainty. In reality, however, we often must make decisions sequentially over time within a system whose evolution is stochastic, which means random, or for which the current state cannot be perfectly observed and must be inferred, probabilistically, from data. This leads to another vast class of problems forming the realm of *stochastic optimization*, which includes commonly-known techniques, such as stochastic gradient search, approximate dynamic programming, optimal control, reinforcement learning, Markov decision processes (MDP), and partially-observable Markov decision processes (POMDPs). Many of these techniques incorporate aspects of simulation into the optimization framework, thus building upon the techniques presented in the two previous sections. Exercise 2 leads the reader through an example application of stochastic optimization to electric power systems.

Example Applications of Stochastic Optimization

1. Energy Generation Portfolio Optimization: Determine schedule for energy generation from multiple sources, including renewables, that accounts for variability in demand and weather [391, 392].
2. AlphaGo and AlphaGo Zero: Use reinforcement learning and approximate dynamic programming to develop an artificial intelligence that can beat grandmasters at the strategy game Go [436].
3. Airline Fare-Locking: Determine a revenue-maximizing policy for airline ticket purchasers to lock in airline fares for a given fee and time duration [450].

Over time, research communities within the fields of operations research, computer science, electrical engineering, and others have studied such problems, often using different vocabulary and notation to describe them. Powell [390, 393] presents a unifying framework for these techniques under the umbrella of *dynamic programming*, which we summarize here. The reader is particularly encouraged to refer to [393] for a thorough survey of stochastic optimization techniques, their history across several different disciplines, seminal references, and how these approaches adhere to this unifying framework. We adopt the notation of [393] throughout this section.

6.4.1 Dynamic programming formulation

Stochastic optimization seeks to identify an optimal *policy*, which can be thought of as a set of contingency plans defining the optimal action to take at each point in time as a function of what is currently known about the system.

We let S_t be a description of the state of the system at time t. We assume that S_t is a *sufficient* description of the state, meaning that only S_t and not $S_1, \ldots, S_{t-1}$ is needed to determine the course of action at time t. Powell notes that different communities interpret the term "state" differently. OR communities tend to think of it as a tangible state of an engineering system, and other communities think of it as the state of an algorithm or a reflection of belief (i.e., probability distribution) about a function. Powell includes in the state S_t three pieces of information: the physical state R_t, other deterministically known information I_t, and the distributional information constituting the belief state B_t.

Given a policy π, $X_t^\pi(S_t)$ is a function that returns the decision, x_t, to make at time t as a function of the current state S_t. That is, $x_t = X_t^\pi(S_t)$. After taking the action, uncertainties in the system and the availability of new information (W_{t+1}) observed at time $t+1$ govern the evolution of the system to a new state according to a (possibly random) *transition function* S^M, such that $S_{t+1} = S^M(S_t, x_t, W_{t+1})$. Uncertainty and state transitions are interpreted by some communities as dynamics that can be modeled using probability distributions (e.g., as in MDPs, stochastic optimization, or robust optimization) or by other communities as being observed from data (e.g., as in reinforcement learning).

We likewise incur a reward (or a cost, depending on the problem context) in period t, $C_t(S_t, x_t, W_{t+1})$. We therefore seek to identify a policy π that maximizes the cumulative *expected* reward, with the expectation taken over all sources of uncertainty or incomplete information:

$$\max_\pi E\left(\sum_{t=0}^{T} C_t(S_t, X_t^\pi(S_t), W_{t+1}) \;\middle|\; S_0\right). \tag{6.8}$$

We can rewrite equation (6.8) in a recursive form, known as Bellman's equation, such that the optimal policy is given by

$$\begin{aligned} X_t^\pi(S_t) = \; &\arg\max_{x_t} E\Bigg(C_t(S_t, x_t, W_{t+1}) \\ &\qquad + \max_\pi E\left(\sum_{t'=t+1}^{T} C_{t'}(S_{t'}, X_{t'}^\pi(S_{t'}), W_{t'+1}) \;\middle|\; S_{t+1}\right) \;\Bigg|\; S_t, x_t\Bigg). \end{aligned} \tag{6.9}$$

The expected values capture uncertainty in the information W_t and in the evolution of the state. Thus, the optimal policy yields a decision $X_t^\pi(S_t)$ at time t that maximizes not only the current period reward but additionally the expected "reward-to-go" from time $t+1$ onward. Equation (6.8) (equivalently, equation (6.9)) is known as a *dynamic program*, which is a cornerstone of operations research. As described in [393], this framework captures all of the stochastic optimization techniques and problem types mentioned above, including stochastic gradient search and reinforcement learning, which are commonly associated with machine learning and artificial intelligence.

6.4.2 Solution techniques

Equations (6.8) or (6.9) can rarely be solved to optimality in a direct way due to the *curse of dimensionality*: due to the sequential structure of the problem, the solution space grows exponentially with the time horizon, the set of actions, and the nature of the probability distribution governing stochasticity in the problem. Techniques to solve equations (6.8) or (6.9) typically use Monte Carlo sampling or direct observation from data, so-called *adaptive algorithms*.

Powell [393] categorizes these solution techniques into four classes, the choice of which one to use depending on the problem type and dimension. (The reader is referred to [393] for details of their implementation.)

- *Policy search*: A class of candidate policies, having associated parameters, is proposed, and the goal is to choose the best among them. This approach is ideal if the policy has a "clear structure" [393].
  - *Policy function approximations*: Choose a general form for the candidate optimal policy (e.g., linear in the state space; or a neural network) and then tune the parameters to find the optimal policy within that class.
  - *Cost function approximations*: Use a simpler cost function for which finding the optimal policy is computationally tractable, and choose a policy that optimizes this function.
- *Lookahead approximations*: Solve a reduced problem to optimality by limiting the extent to which a given action's future impacts are assessed.
  - *Value function approximations*: Here we simplify computation by replacing the "reward-to-go" term of equation (6.9), which represents the cumulative reward earned from time $t+1$ through the end of the time horizon T, with an expected approximate reward depending only on the new state S_{t+1}.
  - *Direct lookahead*: We simulate a single realization (or a sample) of the remaining time steps and use that directly to approximate the "reward-to-go" term of equation (6.9).

The first three of these (policy function, cost function, and value function approximations) are all statistical learning problems in which the approximations are refined as the problem is solved. Powell points out that an important area of research critical to improving the performance of such approaches is the proper modeling of randomness through the appropriate choice of probability distribution (as described in Section 6.3.2.1) and proper assumptions about the system's evolution over time, or by using distributionally-robust modeling.

In the next section, we illustrate how the various tools of operations research have been leveraged to solve real-world prescriptive analytics problems.

6.5 Putting the methods to use: prescriptive analytics

The previous sections describe common tools used in operations research. We now highlight how those tools can be used within the overall operations research modeling process through a series of applications in which the techniques are implemented. As described in Section 6.1.2, this process starts with a real, non-mathematical question, and the goal of the analysis is to mathematically derive a *useful* answer that ends in execution of some action or recommendation. We can break down the analysis into a few key steps:

1. descriptive analysis, which summarizes the data and information about the problem,
2. predictive analytics, which builds a model of the system, and
3. prescriptive analytics, which answers decision-making questions about the system.

While machine learning tends to stop at prescriptive analytics, operations research ties these pieces together. Seeing the analysis as a continuous process is important so that your model both accurately reflects the data you've observed and is able to answer the questions you want to ask. In particular, it is important to understand the techniques available to perform the analysis. As always, there is often a trade-off between how accurate the model is and what questions you can efficiently answer. Below, we present three application areas: bike-sharing systems, online retail recommendations, and HIV prevention. These applications will emphasize the benefits of model simplicity and the difficulties going from model to recommendation to implementation.

6.5.1 Bike-sharing systems

Bike-share systems that allow users to rent bikes are found in most major cities. In this section, we consider docked systems that have stations where users can pick up and drop off bikes at a set of fixed docks. These systems offer a wealth of user data and new transportation challenges since users directly affect the distribution of bikes. The system manager wants to use this data to design a good experience for users; this could translate to maximizing the number of users per day or minimizing dissatisfied customers who cannot find a bike.

Since the locations of bikes change randomly throughout the day, simulation can be used to model the system and ask decision questions. In particular, Jian et al. [252] model the system using discrete-event simulation, as seen in Section 6.3.4.1, by assuming a Poisson arrival process for each possible route consisting of a start and end station. The time-dependent arrival rate is then set to reflect past history. While there exist many other possible ways to predict bike trips, this simulation model is used not just to predict user behavior but also to influence the design of the system. Using this simulation model, Freund et al. [169] show that the question of optimally determining how many fixed docks to place at each station satisfies some convexity properties and can be solved efficiently using a version of gradient descent. Note that this algorithm has to also consider the practicality of the proposed solution by putting limits on how many docks can be at each station. On the other hand, Jian et al. [252] design gradient-based heuristic methods, as described in Section 6.2.4, to determine the optimal allocation of bikes to stations given the number of docks at each station and the expected flow of user traffic throughout the day. (E.g., is it always optimal to fill the docks at Penn station each morning?)

Given the stochastic and often non-symmetric nature of user behavior, bikes need to be redistributed each night in order to meet this proposed allocation. This general practice of redistributing bikes to better meet user demand is called rebalancing and yields many optimization problems that can be formulated as integer programs [170] as defined in Section 6.2.2.2. In particular, one question is how to route a set number of trucks to pick up and drop off bikes throughout the night to meet (or come close to) the desired allocation [170, 403]. Finding these routes needs to be done efficiently in order to start drivers on their routes, and the routes must satisfy a strict time limit. Freund et al. [170] present an integer program for this problem and evaluate the simplifications made in order to find good and feasible solutions in a fixed amount of computation time. In an alternative approach, Nair and Miller-Hooks [356] formulate a stochastic program, the topic of Section 6.4, to redistribute a general fleet of shared vehicles subject to relocation costs, though the routes used to complete this rebalancing are ignored.

6.5.2 A customer choice model for online retail

In online retail settings, retailers often sell many products and want to choose which products to display and how, in order to maximize revenue. Models for customer demand can inform these decisions as well as guide decisions about inventory, pricing, and promotions. This is especially important in the online setting in which users can easily find other sites to search. Choosing an appropriate choice model is often a difficult task as there is a trade-off between model accuracy and tractability; the choice models that capture the most complicated forms of customer behavior are often those whose underlying parameters are difficult to estimate and whose corresponding decision problems are difficult to solve.

One popular model for customer demand is the Multinomial Logit (MNL) choice model. In this model, each product is associated with a value v_i, which is a linear function of the product features, and the probability the item is displayed is based on this value as well as the values of other products displayed. One downside to the MNL model is that it does not capture some logical substitution patterns. For example, this model does not capture the situation when half of the customers want to purchase a polo shirt and the other half want to purchase a t-shirt, regardless of color or other features, since the probabilities will depend on how many t-shirts and polo shirts are shown. However, the corresponding predictive and prescriptive optimization problems tend to be tractable and the model performs well in practice [157]. While Luce [325] introduced the MNL model, interest grew when the seminal work of Talluri and van Ryzin [456] showed that estimating this model's parameters from past customer purchases corresponds to a convex optimization problem, as formulated in Section 6.2.3, that can be efficiently solved. The estimated parameters also give an explicit interpretation into what product features are most influencing customer demand.

Further, Talluri and van Ryzin [456] proved that the assortment optimization problem under this choice model, in which the retailer wants to choose the optimal set of products to display to a customer, can be solved efficiently. In fact, the optimal solution to this problem corresponds to displaying all products with profit above a certain threshold. This can be extended to include practical constraints such as a limit on the number of products shown (e.g., number of products on a web page) [120, 416] and to the robust optimization setting in which the values v_i are assumed to be only partially known [417]. Further, in the network revenue management setting, a retailer wants to determine which products to display over time as inventory fluctuates (e.g., flight tickets at different fare classes). Zhang and Adelman [518] solve this problem using approximate dynamic programming (Section 6.4), with the added complexity of multiple customer segments.

6.5.3 HIV treatment and prevention

While human immunodeficiency virus (HIV) affects roughly 40 million people around the globe [374], the mortality rate from HIV is lower due to treatment and prevention measures. Specifically, antiretroviral therapy (ART) or highly active antiretroviral therapy (HAART), which combines multiple ART drugs, can effectively suppress the virus. However, the effectiveness of this treatment for individuals and its prevention of the spread of HIV depends on identifying individuals who need the therapy and proper dosing and adherence once on it.

In order to identify an individual with HIV, a viral load (VL) test is performed which measures the prevalence of the virus in the bloodstream. However, this testing is expensive, so some places rely on other measures that are correlated with VL. Liu et al. [314] use a tripartition approach to determine whether to perform a VL test by predicting a risk score based on low-cost health measures. If this risk score is above (below) certain thresholds then the person is determined to go on (or not go on) ART. Otherwise, VL testing is used to confirm the diagnosis. To determine the thresholds, the authors consider two approaches, (1) assuming a common distribution on the data, which allows the authors to calculate the optimal thresholds directly, or (2) estimating the distribution and using a grid search for potential thresholds. Interestingly, the authors find that the two methods perform comparably in terms of diagnostic accuracy. Similarly, Liu et al. [313] consider pooled VL testing that combines samples from multiple patients. If the test comes back with a high enough prevalence of HIV, then the individuals within the pool are tested. The benefit of pooling is the potential reduction in the number of tests run and therefore the overall cost. Liu et al. [313] use a similar risk score based on low-cost health measures to determine which patients to test first in order to reduce the overall amount of tests needed. Note that in both cases a model based on only a few health measures is used to inform the overall test process and reduce costs.

Once a patient is on ART, then the dosage and therapy schedule need to be set to effectively treat the disease and minimize potential side effects. Kwon [294] formulates differential equations to model the prevalence of HIV in the system and determine an optimal continuous treatment schedule. On the other hand, Castiglione et al. [86] use a more complex agent-based model of the immune system, as described in Section 6.3.4.2. While the optimal treatment schedule cannot be found efficiently under this model, a genetic heuristic algorithm is used to find a candidate solution and analyze its performance on past data. Further, once on ART, the continuation of care is important for continued viral suppression. Olney et al. [371] introduce an individual-based model of the care cascade to evaluate care recommendations such as pre-ART outreach to patients or population-wide annual HIV screenings, called universal test-and-treat (UTT), on the prevalence of HIV in comparison to the relative cost. Furthermore, Hontelez et al. [228] define a hierarchy of models for HIV prevalence of increasing complexity to evaluate the impact of UTT. They find that all models predict the elimination of HIV if UTT is implemented, but the more complex models predict a later time point of elimination, increasing overall treatment costs.

Each of these applications requires a suite of computational tools to develop, solve, and evaluate the model in the context of real data. The next section describes several widely available solvers and software.

6.6 Tools

Software is constantly evolving, but this section offers the reader some pointers to common software used for optimization, statistics and machine learning, and simulation.

6.6.1 Optimization solvers

To solve the mathematical programs described in Section 6.2, one first needs to represent the mathematical program in some form of computing syntax, and then one needs access to an *optimization solver*.

The simplest way to represent a mathematical program for the purposes of computing is to use an *algebraic modeling language.* The syntax of such languages mirrors the algebraic representation a mathematician would naturally give to a mathematical program (e.g., specifying constraints as summations over certain sets, rather than in a matrix form.) Examples of commonly-used commercial algebraic modeling languages are AMPL and GAMS. Additionally, `Pyomo` is an open-source optimization modeling language for Python [211, 212], and `JuMP` is an open-source optimization environment for Julia [22, 137, 324].

The optimization solver is a piece of software that uses the best known algorithms to solve common classes of mathematical programs. While it is possible to program one's own solver, commercial and open-source solvers have typically been optimized for performance, by incorporating large-scale optimization techniques suitable for solving problems commonly seen in industry. Additionally, most of the popular solvers accept model formulations expressed in the common algebraic modeling languages mentioned above. Thus, our recommendation to the reader is to use an existing solver rather than to create one's own.

The choice of solver to use depends on several factors:

- the type of mathematical program being solved
- whether the problem is being solved in isolation or integrated with a broader code base written in a particular language
- the size of problem being solved
- whether the intended use of the output is commercial, academic, or educational
- the available budget to purchase software
- the comfort and sophistication of the modeler

There are many commercial solvers for various mathematical programming problems. Some well-known solvers include BARON, CPLEX, FICO Xpress, Gurobi, LINDO, and MOSEK. It is not always necessary to purchase an individual license for these commercial solvers; many commercial tools offer free licenses for academic research and classroom use.[6] Moreover, a free resource

[6]For example, at the time of this writing, AMPL offers a free full license for course instructors, which includes several solver licenses, as well as a downloadable textbook [18, 166], making it a good option for classroom purposes.

for academic, non-commercial research is the *NEOS Server*, hosted by the Wisconsin Institute for Discovery at the University of Wisconsin-Madison. The NEOS Server hosts a large number of commercial and open-source solvers for all of the problem classes described earlier and several more. The user uploads files containing the model formulation and input data (typically expressed in an algebraic modeling language like AMPL or GAMS), and the problem is solved remotely on NEOS' high-performance distributed computing servers. Calls to NEOS can also be incorporated into, e.g., `bash` scripts on Linux to facilitate repeated calls [114, 131, 201, 511].

Thus, for the reader first dipping their toes into solving mathematical programs (for example, by doing the projects at the end of this chapter), our suggestion is to first identify the class of mathematical program you are solving. For most novices, this is likely to be a linear program or mixed integer linear program. Next, identify which solvers available on NEOS are suitable for that class of problem; the vast majority of linear programming and mixed integer programming solvers on NEOS accept input in either AMPL or GAMS, if not both. Choose an input format suitable to your desired solver, and express your mathematical program in that format.

For the more sophisticated programmer, an open-source collection of solvers is provided by *COIN-OR* (Computational Infrastructure for Operations Research). These solvers are written in popular programming languages such as Python, C++, and Julia. In addition to providing access to high-quality open-source solvers, the COIN-OR project supports algorithm development research by providing the source code of these solvers and a peer review process for evaluating improvements made to these solvers [104, 322].

Individual packages for certain classes of optimization problems also exist independently of the COIN-OR projects. For example, `cvxpy` is a Python package for solving convex programs [7, 129, 459], and `picos` is a Python API for conic and integer programming problems [418].

The Institute for Operations Research and Management Science (INFORMS) publishes a biannual optimization software survey in the magazine, *OR/MS Today* [165], which covers solvers and modeling languages, and includes both commercial and open-source tools.

6.6.2 Simulation software and packages

The simulation methods described above are easily implemented in common programming languages such as R and Python. Discrete-event simulation packages exist for R (`simmer` [474–476]), Python (`SimPy` [326]) and Julia (`SimJulia` [303]) as well. Spreadsheet-based tools such as Microsoft Excel also have the ability to generate random numbers from common distributions to be used in Monte Carlo simulations. A challenge in agent-based simulation is the need to customize the simulation to the particular problem context. As a result, one generally needs to program an agent-based simulation from scratch. However, some commercial software provides simulation capabilities tailored to particular applications, such as manufacturing, supply chains, and hospital operations. The INFORMS Simulation Software Survey lists several commercial options for the major classes of simulations we have discussed [451].

6.6.3 Stochastic optimization software and packages

Because stochastic optimization captures a broad array of problem types, spanning multiple disciplines of research, it is difficult to point the reader to a small number of standard or widely used tools. For stochastic multistage linear or integer programs (known as *stochastic programming*), the Stochastic Programming Society (SPS) maintains a list of commercial and open-source tools, including AIMMS (commercial), PySP (Python-based) [496], the Stochastic Modeling Interface (developed through COIN-OR), and the NEOS Server [446]. Stochastic optimization frameworks based on dynamic programming, such as reinforcement learning, tend to be problem-specific, precluding the use of standardized software. However, OpenAI has developed the `Gym` platform for "developing and comparing reinforcement learning algorithms" [373]. Similarly, `Horizon` is an open-source reinforcement learning platform developed and in use by Faceboook [179].

6.7 Looking to the future

This chapter has provided a broad outline of several areas of operations research that connect to data science. We've discussed the types of mathematical programming that typically underlie common machine learning algorithms and heuristics, as well as techniques of operations research that are themselves methods for making decisions from data. An emerging area of research examines statistical and machine learning tools *as* optimization problems. We summarize recent research here and refer the reader to a more detailed treatment of this topic in the textbook by Bertsimas and Dunn [41].

Feature selection refers to the task of identifying which predictor variables to include in a statistical or machine learning model. As will be discussed in Chapter 7, with the growing prevalence of very large datasets, some having tens of thousands of possible predictors, it is becoming less realistic to use iterative exploratory methods to identify the best subset of predictors to include. Common dimension reduction techniques, such as principal components analysis, suffer from lack of interpretability: they can help a machine learning model make good predictions, but the user cannot always understand the relationship between the predictors and the response. Recent research in the OR community has developed mixed integer linear and nonlinear optimization methods to solve the best subset problem in linear [457] and logistic [44] regression and in support vector machines [34]. As an example, Bertsimas and

King [44] use mixed integer nonlinear optimization techniques to identify best subsets of variables (including possible nonlinear transformations) to include in a logistic regression model to achieve the sometimes-competing objectives of model parsimony, avoidance of multicollinearity, robustness to outliers, statistical significance, predictive power, and domain-specific knowledge. Moreover, they demonstrate the efficacy of their approach on datasets having over 30,000 predictor variables.

Optimal machine learning refers to casting machine learning problems (which seek to predict the values of unseen test data in a way that minimizes some error function) within a traditional optimization framework. For example, traditional classification methods such as the CART method for binary classification trees (discussed further in Section 8.9) use iterative heuristics that somewhat greedily partition the data to minimize in-class variability at each step. By contrast, an optimal classification tree solves a mixed integer program to find the provably optimal binary partitioning of the data along its features [40].

Prescriptive analytics uses modeling not only to predict variables but also to guide decision-making. When relying on observational data, it is often necessary to integrate prediction into prescriptive decision-making. As we look forward, we can expect methods to more seamlessly bridge predictive and prescriptive analytics. For example, in personalized health, one observes an outcome in response to a particular behavior or treatment, but one does not observe the *counterfactual*: the outcome that one would have observed had a different treatment been given. Recent research attempts to integrate the prediction of the counterfactual into determining the optimal treatment [37, 42]. Likewise, randomized clinical trials often desire early identification of patients who respond exceptionally well or exceptionally poorly to a new treatment. Bertsimas et al. [45] present a mixed integer programming formulation that can identify subsets of the population for whom a particular treatment is exceptionally effective or ineffective, and show that the formulation leads to subsets that are interpretable.

Machine learning problems are often rife with uncertainty due to missing values in the data, imprecise values of predictor variables, or incorrect class labels. *Robust optimization* is an optimization framework that explicitly accounts for the uncertainty of estimated parameter values [31, 38]. Recent research applies robust optimization techniques to machine learning problems, including optimal decision trees [43]. (Decision trees will be discussed in Section 8.9.) Additionally, optimization frameworks have been proposed for multiple imputation of missing data in several machine learning contexts [46].

Operations research concerns itself with leveraging data to improve decision-making. As data science sits at the interface of mathematics, computer science, statistics, it is only natural that operations research paradigms and methodologies will continue to play a prominent role in the field.

6.8 Projects

6.8.1 The vehicle routing problem

Vehicle routing problems are the class of problems related to routing a fleet of vehicles to visit a set of clients. In the *Capacitated Vehicle Routing Problem* (CVRP), the goal is to optimally route a fleet of vehicles to deliver a set of packages to clients subject to vehicle capacity. This exercise will lead the reader through some classic techniques for solving the CVRP. For more details about this problem and vehicle routing in general, see [469]).

In the CVRP, we are given an origin o, a destination d, and a set of clients $N = \{1, 2, \ldots, n\}$ each with demand q_i (for simplicity we set $q_o = q_d = 0$). We represent this set using a complete, directed graph $G = (V, A)$, where $V = N \cup \{o, d\}$ and let the arc cost $c_{(i,j)}$ be the travel cost from i to j for all $(i, j) \in A$. The goal is to route a fleet of K vehicles starting from the origin and ending at the destination so that each client is visited by at least one vehicle, and the total travel cost is minimized. Further, each vehicle has a fixed capacity Q; if a vehicle visits the subset $S \subseteq N$ then we require $\sum_{i \in S} q_i \leq Q$.

1. Below is an integer program for the CVRP, where $\delta^+(S)$ (resp. $\delta^-(S)$) is the set of arcs (i, j) such that $i \notin S$ and $j \in S$ (resp. $i \in S$, $j \notin S$). The variables x_a each represent the number of routes that use arc $a \in A$, while u_i represents the cumulative quantity of goods delivered between the origin o and node i along the chosen route. Explain why this is a valid integer program for the problem. That is, show that any feasible solution to the IP corresponds to a valid solution to the CVRP of equal cost and vice versa. For a referesher on integer programs, see Section 6.2.2.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,j) \in A} c_{(i,j)} x_{(i,j)} \\
\text{subject to} \quad & \sum_{a \in \delta^-(\{o\})} x_a = K \\
& \sum_{a \in \delta^+(\{d\})} x_a = K \\
& \sum_{a \in \delta^+(\{i\})} x_a = 1 \quad \forall i \in N \\
& \sum_{a \in \delta^-(\{i\})} x_a = 1 \quad \forall i \in N \\
& u_i - u_j + Q x_{(i,j)} \leq Q - q_j \quad \forall (i,j) \in A \\
& x_a \geq 0, \ \text{integer} \quad \forall a \in A \\
& q_i \leq u_i \leq Q, \quad \forall i \in V
\end{aligned}
$$

2. As stated in Section 6.6, `picos` is a Python package that can solve integer programs. Use picos to write a function that takes in an $(n+1)\times(n+1)$ matrix of edge costs, an integer K, a vector of n demands, and a capacity Q and returns the total travel cost of the solution to the above IP.

3. The website *http://www.vrp-rep.org/variants/item/cvrp.html* contains data for instances of the CVRP. Download a few instances and compute the corresponding optimal travel cost using the function you wrote in the previous part.

4. Instead of solving the CVRP optimally using integer programming, we can also develop a heuristic algorithm to find a good solution, as described in Section 6.2.4. Design a *local search* algorithm that iteratively tries to improve the current solution using local moves. (E.g., one move might be removing a client from a route and reinserting it at the optimal place among all K routes.) Compare the algorithm's performance to that of the IP. Is there a way to improve the performance of the heuristic? If so, what is the trade-off in runtime?

5. Now suppose that the demands are stochastic rather than deterministic. That is, assume that the demand for client i is a random variable drawn from $N(q_i, \sigma^2)$. Use the *random* package in Python and the IP you wrote above to estimate the expected total travel cost for different values of σ^2 using the techniques seen in Section 6.3.2.2.

6.8.2 The unit commitment problem for power systems

The *unit commitment* problem is an optimization problem that determines which generation sources an electric power system should turn on and off to meet demand at minimal cost, subject to various technical constraints. In this problem, we examine a simple *day-ahead market* in which an independent system operator (ISO) forecasts the next day's energy demand on, e.g., an hourly basis, and schedules the power plants accordingly to meet that demand. A more detailed presentation of such problems can be found in [234].

Using the notation of [234], consider a set G of generators, and a set T of timesteps in a day (e.g., hours). For each generator $g \in G$ and timestep $t \in T$, we must determine the binary start-up decision v_{gt} (indicating whether or not generator g is started up at time t), the binary shutdown decision w_{gt} (indicating whether or not generator g is shut down at time t), and generator g's production rate p_{gt} at each time step between start-up and shut-down. When unit g is started up, we incur a fixed start-up cost SU_g, and when unit g is shut down, we incur a fixed shut-down cost SD_g. Additionally, there is a fuel cost function $F_g(p_{gt})$ for the use of unit g that is linear[7] in the production

[7] In reality, the production costs are modeled as being quadratic. The method for linearizing quadratic costs is described in [234] but is omitted from this exercise for simplicity.

TABLE 6.3: Sample generator parameter data for the deterministic unit commitment problem.

	G1	**G2**	**G3**	**G4**
Min-ON (L_g) (h)	2	1	2	2
Min-OFF (l_g) (h)	2	2	2	1
Ramp-Up Rate (RU_g) (MW/h)	30	15	60	15
Ramp-Down Rate (RD_g) (MW/h)	15	15	60	15
Min-Gen P_g^{min} (MW)	50	30	20	45
Max-Gen P_g^{max} (MW)	150	110	90	110
Startup Cost SU_g (\$)	500	500	800	300
Shutdown Cost SD_g (\$)	500	500	800	300
Fuel Cost a_g (\$)	6.78	6.78	31.67	10.15
Fuel Cost b_g (\$/MWh)	12.888	12.888	26.244	17.820

rate p_{gt} of unit g at time t. That is, $F_g(p_{gt}) = a_g + b_g p_{gt}$ for known constants a_g and b_g. We seek to schedule the generators to minimize total costs subject to the following constraints.

- Each generator has a minimum duration of time that it must stay on once powered on, L_g, and a minimum duration of time that it must stay off once powered off, l_g.
- When powered on, each generator g has a minimum generation limit P_g^{min} and a maximum generation limit P_g^{max}.
- Additionally, each generator has a ramp-down rate RD_g and a ramp-up rate RU_g that limit fluctuations in the production rate p_{gt} from one time period to the next.
- We assume that total production in each time period must meet or exceed forecasted demand D_t.

This project then has the following parts.

1. Formulate the deterministic unit commitment problem described above as a mixed integer linear programming problem, as introduced in Section 6.2.2.2.
2. Solve the deterministic unit commitment problem using the sample data on four generators given in Table 6.3; the hourly demands are given in Table 6.4 (adapted from [234, Table 2.2 and Figure 2.3]). You will need to use optimization software capable of solving mixed integer linear programs. See Section 6.6.1 for suggestions.
3. Next we incorporate variable demand. Assume, as above, that the generation schedule must be set a day in advance, but while the availability of

TABLE 6.4: Sample hourly demand data for the deterministic unit commitment problem.

Hour	1	2	3	4	5	6
Demand (MW)	142	165	152	140	138	152
Hour	7	8	9	10	11	12
Demand (MW)	204	191	193	200	207	214
Hour	13	14	15	16	17	18
Demand (MW)	199	202	211	239	248	342
Hour	19	20	21	22	23	24
Demand (MW)	345	317	209	199	184	175

energy sources is deterministic, demand is variable. Specifically, assume that there are three possible demand scenarios: With 25% probability, demand will be 10% lower than expected in every hour. With 50% probability, demand will be as expected in every hour, and with 25% probability, demand will be 10% higher than expected in every hour. Due to this variability, if demand is not met in a given hour, a penalty cost q is incurred per megawatt of unmet demand. Formulate a mixed integer linear programming model that minimizes total expected costs (with the expectation taken over the three possible scenarios) while satisfying the production constraints for all three scenarios. (We call this a *stochastic programming model*, as introduced in Section 6.4. The definition of expected value is given in Section 6.3.1.)

4. Solve the stochastic programming model using the sample data in Tables 6.3 and 6.4. First let $q = 60$. In this case, all of the demand in the low- and medium-demand cases is satisfied, but not in the high-demand case. By how much does the optimal objective function value increase over the optimal objective function value found in the deterministic case? How does the chosen production schedule differ from that found in part (b)? Next vary q and assess the sensitivity of the chosen production schedule to its value. How high must the penalty on unmet demand be so that demand in the high-demand scenario is always met?

5. Now, let's incorporate generation variability. Renewable energy sources, such as solar and wind, have production capacities that vary with environmental conditions. Consider three scenarios: with 30% probability the minimum and maximum production levels of generators 3 and 4 are 15% less than expected; with 50% probability the minimum and maximum production levels of generators 3 and 4 are as expected; with 20% probability the minimum and maximum production levels of generators 3 and 4 are 10% more than expected. Formulate a mixed-integer linear programming model that minimizes total expected costs (with the expectation taken over the nine possible combinations of demand and

generation scenarios) while satisfying the constraints for all nine scenarios. Solve the stochastic programming model using the sample data in Tables 6.3 and 6.4. What is the cost of uncertainty in the generators?

6.8.3 Modeling project

There are numerous extensions and generalizations to the previous project. Here are two examples.

1. Assume that the expected hourly demand is known, but that each hour's demand independently has a variance σ_t^2 associated with it and that each generator's hourly min and max production level have a variance of σ_{gt}^2. (Variance is defined in Section 6.3.1.) Program a simulation or dynamic program (or both) that determines the lowest cost generation schedule using the sample data in Tables 6.3 and 6.4. (Hint: This task is deliberately ill-posed. For example, you will need to assume distributions on the hourly demands, and make assumptions about the allowable percentage of demand that goes unmet.)

2. As described in detail in [234], there are many ways to extend this formulation. For instance, prices can exhibit variability; demand might fluctuate as a function of price variability (*price elasticity*); there can be network effects. Add increasing levels of complexity to your model and examine how the optimal allocation of generators and total cost varies at each stage of model progression.

6.8.4 Data project

The U.S. Energy Information Administration (EIA) collects hourly data on the U.S. electrical system operations [479]. Download the January-June 2019 BALANCE data from the U.S. Electric System Operating Data tool [481]. (Instructions can be found at [477].)

1. Focus on CISO (California ISO). Using a statistical analysis software such as `R`, begin by exploring and cleaning the data. For example: Which columns are present in the data? Are generation values stored correctly as numerical variables? If not, re-code them appropriately. Do the values make sense? If not, handle the nonsensical values appropriately. Are any values missing? If so, handle them appropriately. Your analysis should be easily reproduced by another person or your future self, so keep track of the analysis you perform and data cleaning decisions you make.

2. Conduct exploratory data analysis, using graphical visualizations and summary statistics, on hourly demand and the amount of demand met by each generation source. Using the methods of Section 6.3.2.1, which

distributions seem appropriate? How would you estimate the expected values and variances that you used in earlier iterations of your model?

3. The dataset does not provide information about fuel costs, generator capacities, start-up/shut-down costs, or ramp-up/ramp-down requirements. Additional EIA data can be found at [478] and [480]. The California ISO website also provides monthly energy production statistics [77]. Using these and other sources of data, estimate the parameters required to solve your day-ahead stochastic model for the CISO market.

4. Solve your day-ahead stochastic model. For parameters that you were unable to precisely estimate from available data, use sensitivity analysis to vary those parameters and explore how the outputs of your model vary in response. Does your portfolio of energy sources used in the optimal solution resemble the portfolio reported by [77]? Why or why not?

Chapter 7

Dimensionality Reduction

Sofya Chepushtanova
Colorado State University

Elin Farnell
Amazon Web Services

Eric Kehoe
Colorado State University

Michael Kirby
Colorado State University

Henry Kvinge
Pacific Northwest National Laboratory

7.1 Introduction

In mathematics the notion of *dimension* has a variety of interpretations. As beginning students, we are initially confronted with the dimension of a vector space, and then the dimension of an associated subspace. When we leave the setting of linear spaces we meet the idea of local linearization and recapture the concept of dimension from a basis for the tangent space, i.e., the topological dimension. A related idea is the term *intrinsic dimension*, which is taken to be the minimum number of variables required by a modeling function to characterize the data locally. Intrinsic dimension can be computed in the setting of the implicit function theorem, for example, if we envision capturing a dataset as the graph of a function. We see that all of these interpretations of dimension involve the number of parameters required, locally or globally, to characterize a dataset. These widely varied notions of dimension allow us to quantify the geometry and complexity of a dataset, making the estimation of dimension an attractive first step in the process of knowledge discovery from data.

The transition from continuous spaces to sets, either finite or infinite, generates new considerations when it comes to the definition of dimension. This of course is the arena of central interest when we are estimating dimension from observed or simulated data. Now one can introduce the idea of non-integer, or fractal dimensions. These include Hausdorff, box counting, and the more recently proposed persistent homology fractal dimension. All of these definitions are *well-defined* mathematically. They have precise geometric frameworks and emerge from a mathematician's view of the world.

The data scientist is confronted with the task of trying to leverage the mathematician's notions of dimension to help characterize the complexity or information content of a dataset. Dimension gives a measure of how many degrees of freedom are at work in an underlying process that generated the data. Estimating this can be extremely useful when one is interested in quantifying the changes in the behavior of a system with an external parameter, e.g., temperature of a heated fluid, or speed of an airplane. A partial differential equation (PDE) model of a physical phenomenon can faithfully capture first principle conservation laws but, in and of itself, does not reveal the complexity of the system. A PDE is a form of mathematical compression that is decoded by numerical simulations that produce complex data. In high dimensions we require phenomenological modeling approaches to exploit the fact that data itself reveals additional principles of interactions among variables.

We have several complementary objectives with this chapter. First, we propose to provide an overview of several foundational mathematical theorems that speak to the idea of dimension in data, and how to capture a *copy* of a dataset of interest in a potentially low-dimensional vector space. We present a range of computationally robust techniques for estimating dimensions of a dataset and provide a comprehensive set of references to additional methodologies. We seek to further establish the mindset that the playground of data analysis is not for only statisticians and computer scientists. Ideas of shape through geometry and topology may provide insight into data that can't be captured using statistical invariants or deep learning. We propose that dimension and its estimation provide the starting point for a mathematician's data science toolkit.

This chapter is organized as follows. Section 7.2 provides conceptual foundations, linking geometric motivations to the concept of dimension. In Section 7.3, we look at Principal Component Analysis (PCA), the most widely-used dimensionality reduction technique, which readers have already seen used in some earlier chapters of this text. Before diving into other methods, Section 7.4 asks how we can know when a dimensionality reduction technique has done a good job.

Because some datasets exhibit non-integer dimensions, we outline fractal and correlation dimensions in Section 7.5. Section 7.6 describes the Grassmannian as a framework for robust data processing and dimension reduction. We show the powerful benefits of exploiting symmetry in data in Section 7.7. We then find a surprising application of category theory to data analysis (the UMAP algorithm for data visualization) in Section 7.8 and give a brief summary of several additional methods in Section 7.9.

For those interested in the mathematical underpinnings of the chapter, Section 7.10 presents some landmark theorems in pure mathematics and connects them to the problem of dimension estimation of data. Finally, Section 7.11 provides concluding remarks, citations to related work, and suggestions for exploring the chapter's methods on real data.

7.2 The geometry of data and dimension

The goal of this section is to expand on the notions introduced in Section 7.1; we understand dimension through varying lenses, each of which is related to underlying geometry. Depending on the context of the dataset of interest to a data scientist, one of these characterizations of dimension may be more useful than others.

A common starting point for the concept of dimension arises from an understanding of dimension in a linear sense. We naturally understand $\mathbb{R}^3$ as a three-dimensional space by recognizing that it is a three-dimensional vector space. In this characterisation, we note that any basis for $\mathbb{R}^3$ has three distinct elements that together generate the space via linear combinations over $\mathbb{R}$, and thus this notion of dimension reflects our perception of dimension as a measure of degrees of freedom.

In a data science context, we may often consider data which is very "close" to linear or which "behaves well" in a linear sense; it is in these cases that it will be meaningful to utilize this definition of dimension. Consider for example, the set of pictures of a person under varying illumination. Images of a person can be thought of as vectors in a high-dimensional vector space; when we construct linear combinations of these vectors, the results are not only well-defined mathematically, but also capture meaning in the context of the data. Linear combinations appear to be new pictures of the same person and we can even have a well-understood notion of the average of a set of pictures. In fact, it has been demonstrated that, under minor assumptions, the set of images of an object under all possible illumination conditions forms a convex cone in the space of images, which is well-approximated by a low-dimensional linear space [28, 183]. The geometry of illumination spaces has been characterized in the setting of the Grassmannian, which is shown to provide a powerful and robust framework for pattern recognition [48, 91].

On the other hand, there are certainly times when a linear notion of dimension would be an unwise approach to understanding and analyzing data. Consider data drawn from $X = \{(x, y, z) : x^2 + y^2 = 1, z = 0\}$, a circle in the xy-plane in $\mathbb{R}^3$ of radius one, centered on the z-axis. We can, of course, count the number of elements in a basis for the subspace in $\mathbb{R}^3$ generated by these points; in doing so, we might conclude that the space from which our data was drawn is two-dimensional. But we all know that treating the data in this way would be ignoring fundamental information. In this case, we would be missing the fact that the dataset itself is not well-behaved under taking linear combinations, so linear spaces are not ideal for understanding the behavior of our data. Furthermore, we'd be failing to observe that there is a different characterization of dimension that would better capture degrees of freedom in this setting.

As the previous example of a circle embedded in $\mathbb{R}^3$ suggests, such cases are better handled by considering the data in more generality, in this case as being drawn from a manifold rather than from a vector space. Then we can recognize that the circle is actually a one-dimensional object via its local identification with $\mathbb{R}$. While one may reasonably detect if data is well-approximated by a low-dimensional linear space via, e.g., singular values, detecting whether data is close to being drawn from a manifold can be more challenging. This topic is addressed further below where we consider the computation of the topological dimension of a manifold from samples.

The methodology that supports extracting a manifold representation for a particular dataset is often referred to as nonlinear dimension reduction or manifold learning and contains a wide variety of techniques. It is worth pausing for a moment to establish that expending the effort to find such a manifold is a worthwhile endeavor. Often, after applying these techniques, one obtains a visualization of the data in a low-dimensional space which then guides intuition for analysis. Techniques that are commonly used for visualization in addition to dimension reduction include Multidimensional Scaling (MDS), Isomap, Laplacian eigenmaps, t-distributed Stochastic Neighbor Embedding (t-SNE), and Uniform Manifold Approximation and Projection (UMAP), some of which are explored below. In addition to gaining intuition via visualization, such dimension reduction often provides context that drives other computations on the data; in the example of the embedded circle data, it would be appropriate to compute distances between points along the circle rather than the default choice of Euclidean distance. In biology, for example, understanding the nonlinear evolution of data trajectories is critical to decoding processes [185].

Finally, the so-called "curse of dimensionality," first introduced by Bellman in [30] tells us that we must use caution when attempting to complete analysis in high-dimensional spaces. Because volume increases exponentially fast with dimension, in general, data is sparse in high-dimensional spaces. In order to learn a model, the amount of data required to do so with confidence frequently grows exponentially with dimension. Consequently, obtaining a meaningful low-dimensional representation of the data is often a fundamental step in successful data analysis [53].

Manifold learning comprises a wide range of techniques, each of which seeks to find a low-dimensional representation for a dataset on the basis that, while the data may reside in a high-dimensional ambient space, it often lies on or close to a low-dimensional manifold. For example, as discussed in [29], consider a dataset of grayscale pictures of an object taken at a resolution of $n \times n$ by a moving camera under fixed lighting, which is naturally contained in $\mathbb{R}^{n^2}$. Yet the dataset must have the structure of a low-dimensional manifold embedded in $\mathbb{R}^{n^2}$ because the set of all images of the object under all camera positions has degrees of freedom defined exclusively by the movement of the camera. We often think of "intrinsic dimension" as precisely corresponding to this notion of degrees of freedom, and manifold learning provides a tool

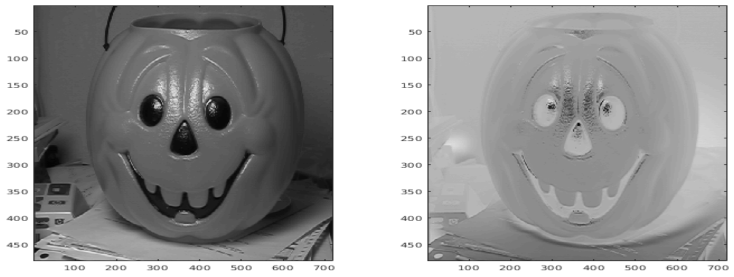


FIGURE 7.1: Left: The mean image. Right: The first principal component.

for attempting to discover that dimension. As a related example, we collected grayscale images of the plastic pumpkin that appears on the left in Figure 7.1 as it spun on a record player through approximately three rotations. This is an analogous setting, and we should expect that while the data resides in a high-dimensional ambient space corresponding to the number of pixels, the intrinsic dimensionality of the data must be one because there was only one degree of freedom in the construction of the dataset that arose from the angle of rotation. In Figure 7.2, we show the projection to three dimensions of this dataset using PCA, introduced in Section 7.3; note that it does appear to exhibit the structure of a one-dimensional manifold.

In general, we may think of dimension estimation and reduction of data in terms of maps to a lower-dimensional space that preserve some quantity of interest. Some manifold learning techniques emphasize preservation of pairwise distances between points; e.g., MDS and Isomap seek to preserve pairwise distances and pairwise geodesic distances, respectively. Others, such as t-SNE, characterize point similarity/dissimilarity via probability distributions and emphasize finding a low-dimensional representation that is faithful in this regard. Many highlight preservation of local structure, e.g., Locally Linear Embeddings (LLE) and Laplacian eigenmaps. PCA, which is a linear dimension reduction technique, can also be thought of in this regard; the projection to a lower-dimensional space via PCA maximizes (seeks to preserve) variance. Taken as a whole, we often learn about the dimension and structure of our data by mapping to a lower-dimensional space, especially when such maps are possible with relatively "small" loss in terms of that which we seek to preserve. When we have a manifold representation for a dataset, we take the dimension of the manifold to be the dimension of the data, which agrees in a local sense with the degrees of freedom characterization of dimension.

Finally, we should note that for certain datasets, fractal dimension becomes the appropriate notion of dimension. In these cases, it is common that the data does not lie on a sub-manifold and thus the context of the problem necessitates a different approach to understanding dimension. Popular real-world settings

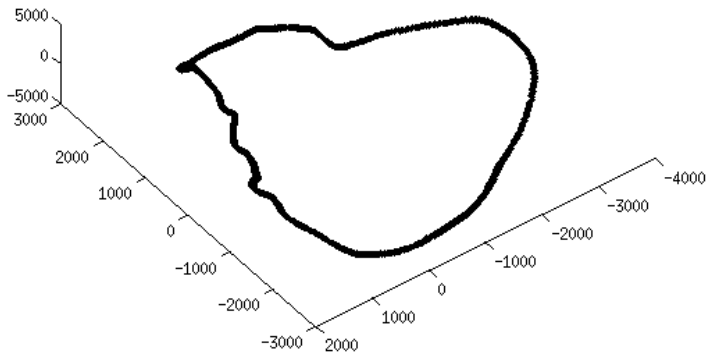


FIGURE 7.2: The projection to three dimensions using Principal Component Analysis of the dataset of pumpkin images taken as the pumpkin rotated on a record player. Note that the dataset does appear to be one-dimensional. Image from [153], used with permission.

that have seen applications of fractal dimension include coastlines, fault geometry, vegetation, and time series [218, 332, 353, 369]. Fractal dimensions, too, can be viewed as a count of degrees of freedom, tying back to our earlier intuitive understanding of dimension.

There are many definitions of fractal dimension; these definitions often agree, especially on well-behaved examples, but they need not agree in general. Hausdorff proposed a definition that he utilized in the context of the Cantor set, and his and other definitions of fractal dimension agree on the Cantor set having fractal dimension $\log_3(2)$ (an intuitively reasonable result since the Cantor set ought to have dimension greater than zero but less than one). The Hausdorff dimension of a set is considered an important definition of dimension, but it can be difficult to compute in practice. A related fractal dimension is the box-counting or capacity dimension, which arises from a relaxation of the definition of Hausdorff dimension. The box-counting dimension considers how the infimum of the number of cubes of fixed side length required to cover the set scales as the side length goes to zero (equivalently, one may use alternatives such as closed balls of fixed radius). A second alternative to Hausdorff dimension, and one of the most popular choices for computationally feasible fractal dimension estimation, is correlation dimension, which utilizes statistics of pairwise distances. For more details on fractal dimension, see [151].

In the next section, we dive deeper into a selection of specific dimension estimation and reduction methods.

7.3 Principal Component Analysis

Principal Component Analysis (PCA) provides an optimal data-driven basis for estimating and reducing dimension and is one of the main tools used for that purpose. It is also widely used as a technique for visualizing high-dimensional datasets. The method can be traced back to Pearson [383] although it has been rediscovered many times and is also referred to as the Karhunen-Loève Transform [259, 318] and Empirical Orthogonal Functions (EOFs) [319]. It is intimately related to the Singular Value Decomposition (SVD) and the numerical linear algebraic algorithms for computing the SVD may be used for computing PCA. Readers who would like a review of SVD (or the related linear algebra in general) may wish to read Section 3.2.6 before diving into Section 7.3.1 below. Interestingly, the Fourier transform can be viewed as a special case of PCA for translationally invariant datasets [268].

7.3.1 Derivation and properties

PCA provides a principled technique for changing coordinates of data with the goal of revealing important structure. Given a dataset of points $\{x^{(i)}\}$, $i = 1, \ldots, n$, the goal is to determine the best k-dimensional subspace of $\mathbb{R}^m$ where m is referred to as the *ambient* dimension of the data, i.e., each $x^{(i)} \in \mathbb{R}^m$. Let's write the approximation of a point $x^{(i)}$ as a decomposition into the sum of the component a in the projected space and the residual b in the orthogonal complement, so

$$x = Px + (I - P)x,$$

where we have set $a = Px$ and $b = (I - P)x$. Here $P = UU^T$ and the columns of U are an orthonormal matrix $U^T U = I$. Assuming that our projections are orthogonal, we have the Pythagorean Theorem to decompose the lengths squared

$$\|x\|^2 = \|a\|^2 + \|b\|^2.$$

It is convenient to define the data matrix $X = [x^{(1)}|\cdots|x^{(n)}]$. Using the identity $\sum_i \|x^{(i)}\|^2 = \|X\|_F^2$ we can compute the decomposition over the entire dataset

$$\|X\|_F^2 = \|A\|_F^2 + \|B\|_F^2$$

where $A = [a^{(1)}|\cdots|a^{(n)}]$ and $B = [b^{(1)}|\cdots|b^{(n)}]$. (Recall that the Frobenius norm of a matrix is the square-root of the sum of the squares of the entries.) We can seek an orthogonal projection that maximizes the sum of the projected lengths

$$\underset{U^TU=I}{\text{maximize}} \|UU^TX\|_F^2. \tag{7.1}$$

It is readily shown using the technique of Lagrange multipliers that the extrema for this problem satisfy

$$XX^Tu_i = \lambda_i u_i. \tag{7.2}$$

Since XX^T is symmetric and positive semi-definite, it follows that we can order the basis in terms of the real eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n \geq 0.$$

Note that since

$$\lambda_i = u_i^T XX^T u_i,$$

the eigenvalue λ_i measures the total projection of the data onto the i^{th} coordinate direction. If the data is mean subtracted, i.e., the data has zero mean in the centered coordinate system, then the eigenvalues are in fact the statistical variance captured in that coordinate direction. The eigenvalue spectrum measures the distribution of information in the space. This dispersion of information can be measured using entropy. Shannon's entropy is defined as

$$E = -\sum_i \lambda_i \ln \lambda_i. \tag{7.3}$$

This quantity E is at a maximum when all the eigenvalues are equal indicating that there is no preferred direction in the dataset, i.e., no data reduction is possible. In contrast, if only one eigenvalue is nonzero, then the data can be reduced to one dimension and is maximally compressible. It can be shown that the PCA basis is the one that minimizes Shannon's Entropy. In summary, over all orthogonal bases the PCA basis [268, 495]

- maximizes the length-squared of the projected data,
- equivalently, minimizes the length-squared of the residuals,
- equivalently, maximizes the statistical variance of the retained data,
- equivalently, minimizes Shannon's entropy, and
- produces coordinates in which the dimension-reduced data are uncorrelated.

7.3.2 Connection to SVD

The Singular Value Decomposition (SVD) for a matrix is connected to PCA in a simple way. Recall that every rectangular matrix X can be decomposed into the product of orthogonal matrices U, V, and a diagonal matrix Σ. The matrices U and V contain the left and right singular vectors of X, respectively, and Σ contains the ordered singular values. Thus we can write the SVD of X as

$$X = U\Sigma V^T. \tag{7.4}$$

It can be shown that the left singular vectors are the eigenvectors arising in PCA. This follows from the fact that

$$XX^T = U\Sigma^2 U^T, \tag{7.5}$$

or an $m \times m$ eigenvector problem. Analogously, the right singular vectors are the eigenvectors of the $n \times n$ problem

$$X^T X = V\Sigma^2 V^T. \tag{7.6}$$

Note that the relationship between U and V through the SVD allows one to solve only the smaller of the two eigenvector problems. Given its relevance for points in high dimensions, e.g., high-resolution digital images, solving equation (7.6) is referred to as the *snapshot method* while solving equation (7.5) is referred to as the *direct method*; see [269, 437]. Note that, in practice, if the sizes of m, n allow, it is most desirable to solve the "thin" SVD so that the matrices XX^T or $X^T X$ are never actually computed. For additional details, see [268].

7.3.3 How PCA is used for dimension estimation and data reduction

In this section, we consider an application of PCA/SVD for dimension reduction and estimation.

We consider the variation of illumination on an object by waving a light in front of a stationary plastic pumpkin. In the left of Figure 7.1 we have plotted the mean image in a sequence of 200 images. The first eigenvector, shown on the right, contains 85% of the total data variance even though the ambient dimension is 345,600. The first 10 dimensions contain over 98% of the total variance.

The projection of the image sequence onto a two-dimensional subspace is shown in in the left of Figure 7.3; this representation contains over 94% of the variance. While this is not exactly an isometric embedding, it reveals the qualitative motion of the lamp being swung back and forth by hand. The log of the eigenvalues shown on the right of this plot show the traditional *scree*

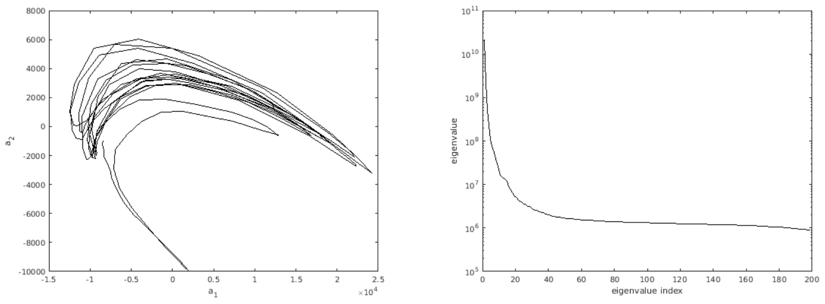


FIGURE 7.3: Left: Projection of the images in the illumination sequence to two dimensions, created using the code in Figure 7.4. Right: The nonzero eigenvalues of the matrix XX^T.

plot. Note that the steep portion begins to flatten out around ten dimensions and this would be taken as the estimate for the dimension of this data.

Note that there are 200 images and that they were mean subtracted to perform this analysis. This is done effectively using

$$\tilde{X} = X(I - ee^T/n)$$

where e is the vector of ones. As a result of this mean subtraction there are 199 eigenvectors associated with nonzero eigenvalues. But, according to the discussion above, about 189 of the eigenvectors are not required for representing the data since the images only sit in these dimensions with very small amplitude.

MATLAB code for creating the image on the left of Figure 7.3 using SVD appears in Figure 7.4.

7.3.4 Topological dimension

An important question for geometric data analysts is how to estimate the dimension of a manifold from sampled data. There is an elegant solution based on computing the singular values $\sigma_i(\epsilon)$ of a local ϵ-ball of data as a function of the radius of the ball. The algorithm begins with a small data ball centered at a point of interest, and that contains enough points for the SVD to be informative. The algorithm proceeds by enlarging the size of the data ϵ-ball $B_\epsilon(p)$ centered at a point p on the manifold [65,67]. The key idea is that, for small ϵ, the singular values that scale linearly with the radius of the ball are associated with the singular vectors that span the tangent space of the manifold. The remaining singular values reflect the noise distribution in the data. As the radius of the ball increases, curvature effects begin to dominate as reflected by the appearance of singular values that scale quadratically. Extensions of

```
%Input: data1 is the data matrix of the color image sequence
%with dimensions 480 (#rows)
%               x 720 (#cols)
%               x 3 (R,G,B)
%               x 200 images in sequence.
%initialize the data matrix,
%whose columns will be "vecced" images:
Q = [];
for i = 1:200
    %convert color images to gray level:
    A = rgb2gray(squeeze(data1(:,:,:,i)));
    %turn each image matrix into a vector:
    x = reshape(A,size(A,1)*size(A,2),1);
    %collect images as columns in one matrix:
    X = [X double(x)];
end
MM = X*ones(200)/200;
MX = X-MM; %complete the mean subtraction
[U S V] = svd(MX,0); %thin SVD
A = U(:,1:2)'*MX; %find 2D reduction
```

FIGURE 7.4: MATLAB code for generating the image shown on the left of Figure 7.3, taking as input the sequence of 200 images described in Section 7.3.3.

these ideas to function spaces have been presented in [66]. Explorations of computational and algorithmic strategies are considered in [238].

We note that the obvious question—do the singular vectors that scale quadratically with ϵ reveal additional geometric information—can be answered in the affirmative [15, 16, 525]. You can also compute generalized curvatures using singular values [14].

7.3.5 Multidimensional scaling

In the above algorithms, we assumed that our dataset X was given with coordinates in some field (so that each $x \in X$ is a point in a vector space). However, in many cases of interest, data may not be available as points in a Euclidean space. Instead, we may have access to relationships between data points in a set, i.e., we may possess a similarity matrix S associated with a dataset that satisfies

$$S_{ij} \leq S_{ii} \text{ and } S_{ij} = S_{ji}.$$

Defining $D_{ij} = S_{ii} + S_{jj} - 2S_{ij}$ gives rise to a distance matrix [334]. Alternatively we may have distances between data points as a starting point. MDS answers the question concerning how to determine a *configuration* of points whose Euclidean distances are as faithful as possible to the provided distance matrix.

One natural strategy is to then try to find a Euclidean approximation $\widetilde{X} \in \mathbb{R}^n$ of the metric space (X, m), for some $n \geq 1$, where the corresponding metric on $\widetilde{X}$ is the usual Euclidean metric. Once we have $\widetilde{X}$, we also have all the tools of linear algebra at our disposal.

The loss function minimized by MDS is the *strain* of the Euclidean approximation $\widetilde{X}$ relative to X. Minimizing the strain is equivalent to minimizing

$$\left(\sum_{x_1, x_2 \in X} (b_{ij} - x_1 \cdot x_2)^2 \right)^{\frac{1}{2}}$$

where b_{ij} comes from a double mean-centered version of the distance matrix D. Note that MDS accesses Euclidean distance via its relationship with the inner product in the identity:

$$||x_1 - x_2||^2 = ||x_1||^2 - 2x_1 \cdot x_2 + ||x_2||^2.$$

For a more detailed discussion on MDS the reader is referred to [50], or to the example code in Figure 7.5.

```
%Multidimensional Scaling Algorithm

%INPUTS:
%   D -- a distance matrix D(i,j) is the distance
%        between points i and j
%   dim -- the dimension of the linear embedding space
%OUTPUTS:
%   X -- a data matrix of size dim x number of points

A = -0.5*D.*D;%Hadamard product
n = size(D,1)%n is the number of points
onevec = ones(n,1);
H = eye(n)-onevec*onevec'/n;%centering matrix
B = H*A*H %double centering of A
[V E] = eig(B)
VN = V*sqrt(E);
X = (VN(:, 1:dim))'
```

FIGURE 7.5: Implementation of the MDS algorithm in MATLAB.

7.4 Good projections

We return now to datasets consisting of points in a vector space V. In most situations a dimensionality reduction algorithm should approximately preserve the distance between distinct points x_1 and x_2 in a dataset $X \subset V$ even as pairs of points in V, but not in X, are collapsed. Another way to say this is that we would like to find functions that preserve the secant set associated with X. The *secant set of* X is defined as

$$S := \{x_1 - x_2 \mid x_1, x_2 \in X,\ x_1 \neq x_2\}.$$

Note that the secants of S will in general have different lengths. Depending on what aspects of X we want to preserve during reduction, we may either normalize our secants (to put them on an equal footing) or discard secants below a certain length threshold (where noise has a larger effect).

One example of a secant-based dimensionality reduction algorithm, which operates on the normalized secant set $\widetilde{S}$ and which is related to PCA, is found in [68]. Here, one solves the optimization problem

$$\arg\max_P \sum_{s \in \widetilde{S}} ||Ps||, \tag{7.7}$$

to find an orthogonal projection P, which projects the data from $\mathbb{R}^n$ to a k-dimensional subspace (for $k < n$) that optimally preserves normalized secants of the dataset. This problem can be solved by taking the singular value decomposition of the matrix whose columns (or rows) are the elements of $\widetilde{S}$.

We saw in Section 7.2 that it can be valuable to estimate the intrinsic dimension of a dataset. In this case, the constructive proof of Whitney's Embedding Theorem suggests that instead of solving (7.7), we should solve

$$P^* = \arg\max_P \left(\min_{s \in \widetilde{S}} ||Ps|| \right), \tag{7.8}$$

which tries to find a projection such that the most diminished secant is maximized. Intuitively, if our projection drastically decreases the length of a secant, and hence collapses two distinct data points, then we have most likely intersected or folded the data manifold on itself and hence the dimension we are projecting into may be insufficient for embedding the manifold. As Whitney's Embedding Theorem gives an upper bound on the number of dimensions required to embed an m-dimensional manifold, studying the value of $\min_{s \in \tilde{S}} ||P^* s||$ for the orthogonal k-projection solution P^* of (7.8), as k varies, can give one approximate bounds on the intrinsic dimension of X. The *secant-avoidance projection (SAP) algorithm* [291] is an example of an algorithm that can efficiently solve (7.8). For a hierarchical approach to (7.8) see [293]. The word "approximately" must be used in this case because whereas (7.7) is convex, (7.8) is not.

To see the difference between secant-based methods and PCA, we consider the simple example of a circle S^1 embedded into $\mathbb{R}^{2n}$ via the trigonometric moment curve $\phi_{2n} : S^1 \to \mathbb{R}^{2n}$. Parameterized by $t \in [0, 2\pi]$ this curve is given by

$$\phi_{10}(t) := (\cos(t), \sin(t), \cos(2t), \dots, \cos(nt), \sin(nt)).$$

Let U be a dataset obtained by randomly selecting 800 points from $\phi_{10}(S^1) \subset \mathbb{R}^{10}$. Figure 7.6 shows the result of using PCA to project U into $\mathbb{R}^3$. Notice that in order to optimize for variance, portions of $\phi_{10}(\mathbb{R})$ become tightly knotted in the course of dimensionality reduction. Put another way, PCA is willing to collapse a few secants if it can preserve overall variance. We contrast this with the result of using a projection that satisfies (7.8). Notice that since (7.8) focuses on maximizing the most collapsed secant, the resulting projection does not have any tight knots in it.

If we solve equation (7.8) over the normalized secant set $\widetilde{S} \subset \mathbb{R}^n$, the k-projection solution P^* provides an additional statistic

$$\kappa_k = \min_{s \in \tilde{S}} ||P^* s||,$$

that describes the extent to which the least well-preserved secant is diminished under the optimal projection. We call the collection of all such values

$$\boldsymbol{\kappa} = (\kappa_1, \kappa_2, \dots, \kappa_n),$$

the κ-profile for the dataset X. It has been shown that changes in the κ-profile often reflect fundamental changes in a dataset [292]. For example, different soundscapes [162] exhibit fundamentally different κ-profiles as shown in Figure 7.7, reflecting their differing complexities.

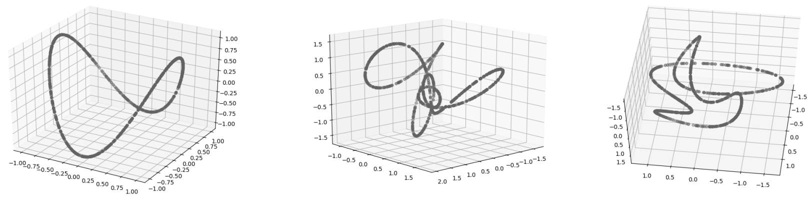


FIGURE 7.6: Visualizations of the result of projecting randomly sampled points from the trigonometric moment curve in $\mathbb{R}^{10}$ to $\mathbb{R}^3$ using the first three coordinates (left), using PCA (center), and using the SAP algorithm (right). Note that since PCA tries to maximize only overall variation, the projection that optimizes this objective function leaves some points knotted together. The SAP projection, on the other hand, finds a projection that keeps the points more evenly spaced throughout the whole curve.

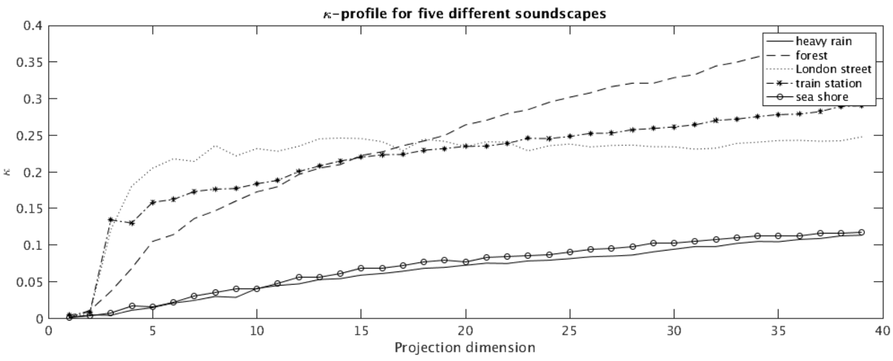


FIGURE 7.7: The κ-profile for five different soundscapes. A recording of each soundscape [162] was chopped into fixed-length "vectors" and these were treated as points in a high-dimensional Euclidean space where their κ-profile could be calculated. Note that, unsurprisingly, environments with more relatively uniform, incoherent noise, such as the seashore and heavy rain, appear to be higher dimensional than environments with more structured noise.

7.5 Non-integer dimensions

The non-integer definitions of dimension in this section initially arose in the study of dynamical systems. We have included a short detour through some of these basic concepts. Readers primarily interested in computing dimension from data can return to the technical details related to dynamics later if interested. We begin with a brief review of dynamical systems, but readers familiar with the material may skip ahead to Section 7.5.2.

7.5.1 Background on dynamical systems

Here we provide the foundation to precisely define what is meant by an attractor. We first give some basic working definitions from dynamical systems theory.

Definition 7.1 (Autonomous Differential Equation) *An autonomous differential equation on the open phase space $M \subset \mathbb{R}^N$ is an equation of the form,*

$$\frac{du}{dt} = F(u)$$

where $u : \mathbb{R} \to M$ is a smooth path, also called an orbit, and $F : M \to \mathbb{R}^N$ is a smooth function on M.

Associated to a differential equation is its flow. Here we provide the definition of a complete flow to avoid subtleties involved in defining the flow domain.

Definition 7.2 (Complete Flow) *The flow $f : \mathbb{R} \times M \to M$ associated to the above differential equation is the unique smooth map satisfying the following properties.*

1. $f(0, u) = u$
2. $f(s, f(t, u)) = f(t + s, u)$
3. $\frac{\partial f}{\partial t}(t, u) = F(f(t, u))$

Note that properties 1 and 2 define a smooth action of $\mathbb{R}$ on M. We then wish to characterize those sets which are invariant under the dynamics.

Definition 7.3 (Invariant Set) *A set $X \subset M$ is invariant if for any $x \in X$ and any $t \in \mathbb{R}$, $f(t, x) \in X$.*

Building to our definition of attractor, attractors should attract in some sense. We make this precise.

Definition 7.4 (Attracting Set) *An attracting set is a closed invariant subset $X \subset M$ such that there exists a neighborhood U of X which has the property that if $u \in U$ then $f(t, u) \in U$ for all $t \geq 0$ and $f(t, u) \to X$ as $t \to \infty$.*

We call $\bigcup\limits_{t \leq 0} f^t(U)$ the domain of attraction, where $f^t : M \to M$ is the diffeomorphism defined by $f^t(u) = f(t, u)$. An attractor has the unique property that it must also be generated by an orbit of the dynamical system. By an orbit of the dynamical system we mean a path $x(t) = f(t, p)$ where $p \in M$ is fixed.

Definition 7.5 (Attractor) *An attractor X is an attracting set which contains a dense orbit, i.e., an orbit whose image is dense in X.*

7.5.2 Fractal dimension

The simulation of dynamical systems leads to interesting objects with non-integer dimensions known as *strange attractors*, as in Figure 7.8. How does one define and compute a definition of dimension that agrees, in some sense, with our geometric intuition? Is the notion of dimension even unique? How do different definitions of dimension relate to one another?

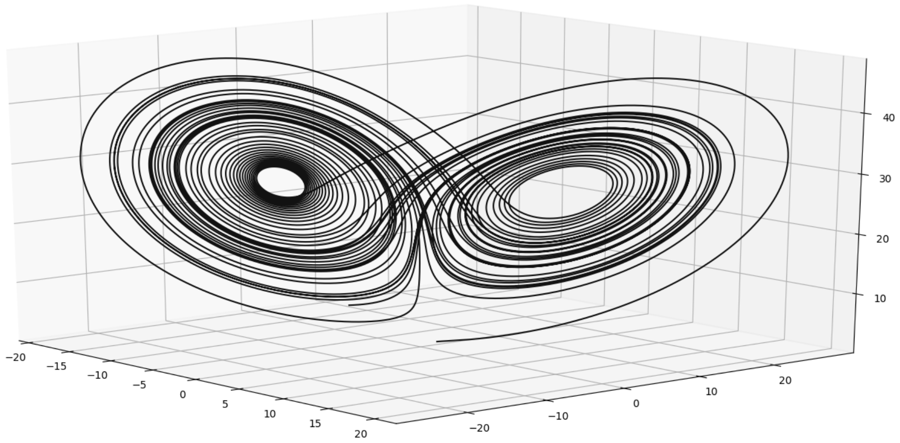


FIGURE 7.8: The Lorenz attractor in state space with parameters $\rho = 28$, $\sigma = 10$, and $\beta = 8/3$ chosen by Lorenz so that the system will exhibit chaotic behavior. The code to generate this image is in Figure 7.9. See [320] for more details.

For a set $X \subset \mathbb{R}^N$ whose closure is compact, let $\mathcal{N}_\varepsilon(X)$ be the minimal number of ε-balls needed to cover X and define

$$\mu_F(X, d, \varepsilon) = \mathcal{N}_\varepsilon(X)\varepsilon^d \tag{7.9}$$

and

$$\mu_F(X, d) = \limsup_{\varepsilon \to 0^+} \mu_F(X, d, \varepsilon). \tag{7.10}$$

If we consider, for example, a two-dimensional surface in three-dimensional space and a covering of the surface with the minimal number of three-dimensional ε-balls, we would observe that as ε approaches zero the cover would decrease in volume to zero. This is due to the fact that a two-dimensional surface has measure zero in $\mathbb{R}^3$. To find that the dimension of the surface is in fact two, we would want to reduce the dimensionality of the ε-cover by reducing d in the definition of μ_F above until we see that the ε-cover has nonzero d-volume in the limit as ε goes to zero. This motivates the definition of the fractal dimension.

Definition 7.6 (Fractal Dimension) *Let X be a set whose closure is compact. The fractal dimension of X is the quantity*

$$\dim_F(X) = \inf_{d \geq 0} \left\{ d : \mu_F(X, d) = 0 \right\},$$

where $\mu_F(X, d)$ is defined by Equations 7.9 and 7.10.

The fractal dimension, also known as the Hausdorff dimension, is often impractical to compute because it requires finding the minimal number of ε-balls needed to cover the set X. A more practical measure of dimensionality, which can be more readily computed, is the box-counting dimension. For the box-counting dimension, instead of minimally covering the set X with ε-balls and counting them, we partition $\mathbb{R}^N$ into a uniform grid of N-dimensional hypercubes with side length ε and then count the number of hypercubes that intersect X.

Let $M_\varepsilon(X)$ denote the number of hypercubes with side length ε that intersect X, and define

$$\mu_B(X, d, \varepsilon) = M_\varepsilon(X)\varepsilon^d.$$

With some basic algebra we obtain the equality

$$d = \frac{\log\left(\mu_B(X, d, \varepsilon)\right)}{\log(\varepsilon)} + \frac{\log\left(1/M_\varepsilon(X)\right)}{\log(\varepsilon)}.$$

Similar to the fractal dimension, we consider only those d where $\lim_{\varepsilon \to 0^+} \mu_B(X, d, \varepsilon) = 0$. Taking a limit on the expression above we obtain

$$d = \lim_{\varepsilon \to 0^+} \frac{\log\left(1/M_\varepsilon(X)\right)}{\log(\varepsilon)}.$$

Hence we have the following definition.

Definition 7.7 (Box-counting Dimension) *Let X be a set whose closure is compact. The box-counting dimension of X is the quantity*

$$\dim_{Box}(X) = \lim_{\varepsilon \to 0^+} \frac{\log\left(1/M_\varepsilon(X)\right)}{\log(\varepsilon)},$$

where $M_\varepsilon(X)$ is the number of hypercubes with side length ε in a uniform grid that intersect X.

In general, the box-counting dimension is at least as large as the fractal dimension, but equality will hold when X satisfies the so-called open-set condition; see [152] for more details.

7.5.3 The correlation dimension

The fractal dimension gave us a pure geometric measure of dimensionality for sets that are noisier than sets that are locally Euclidean-like (i.e., manifolds, CW-complexes, etc...). Many times in practice, such a set arises as an attractor for some dynamical system. The correlation dimension is built to measure dimensionality of attractors, sets which are generated by the dense orbit of a dynamical system, as in Figure 7.8. Nevertheless, the definitions of the correlation dimension do not prohibit one from calculating it for an arbitrary dataset. In fact the correlation dimension is often very close to the fractal dimension of a set and generally far simpler to compute.

Now consider a generic time series $\{X_i\}_{i=0}^{\infty}$ in $\mathbb{R}^N$. (For an arbitrary dataset $X \subset \mathbb{R}^N$ we can just take a random ordering of the points. A brief introduction to time series in general appears at the end of Section 4.6.4.) We would like a definition of dimension for X that takes into account its dynamical complexity rather than just its geometric degrees of freedom. To form this definition we first define the correlation integral associated to the time series.

Definition 7.8 *For any $\varepsilon > 0$, the correlation integral with respect to ε is the value*

$$C(\varepsilon) = \lim_{n\to\infty} \frac{1}{n^2} \sum_{i,j=0}^{n} \theta(\varepsilon - |X_i - X_j|), \tag{7.11}$$

where θ is the Heaviside function, which is 0 on $(-\infty, 0)$ and 1 on $[0, \infty)$.

$C(\varepsilon)$ gives the probability of finding two points in the full time series that are closer than ε to one another. The intermediate values,

$$C(\varepsilon, n) = \frac{1}{n^2} \sum_{i,j=0}^{n} \theta(\varepsilon - |X_i - X_j|),$$

give the probability that there will be two points in the truncated time series less than ε distance away. Grassberger and Procaccia [199] establish that $C(\varepsilon)$ behaves as a power law for small ε so that $C(\varepsilon) \sim \varepsilon^{\nu}$ for some $\nu \in \mathbb{R}$. This yields the following definition.

Definition 7.9 (Correlation Dimension) *The correlation dimension of a set X, when it exists, is the quantity*

$$\dim_{\text{Corr}} X = \lim_{\varepsilon\to 0^+} \frac{\log C(\varepsilon)}{\log \varepsilon},$$

where $C(\varepsilon)$ is the correlation integral defined in Equation 7.11.

To get an idea of the correlation dimension, it is enlightening to see how it relates to the box-counting dimension. From the definitions given in the fractal dimension section along with some analysis, if $D_B = \dim_{\text{Box}}(X)$ then

$$C(\varepsilon) \geq \frac{\varepsilon^{D_B}}{\mu_B(X, D_B, \varepsilon)}.$$

By taking the log of each side and dividing by $\log(\varepsilon)$ we obtain

$$\frac{\log C(\varepsilon)}{\log \varepsilon} \leq D_B - \frac{\log \mu_B(X, D_B, \varepsilon)}{\log \varepsilon}.$$

Now we take the limit to obtain

$$\begin{aligned}\dim_{\text{Corr}}(X) &\leq \lim_{\varepsilon \to 0^+} \left(D_B - \frac{\log \mu_B(X, D_B, \varepsilon)}{\log \varepsilon} \right) \\ &= D_B - \lim_{\varepsilon \to 0^+} \frac{\log\left(\mu_B(X, D_B, \varepsilon)\right)}{\log \varepsilon} \\ &= \dim_{\text{Box}}(X).\end{aligned}$$

If we assume a uniform distribution of points on the set X with a little analysis [199] we can obtain

$$\dim_{\text{Corr}}(X) = \dim_{\text{Box}}(X).$$

Hence we can roughly think of the correlation dimension as a weighted fractal dimension in the sense that regions of the set X that are more frequented by the time series are weighted more heavily than regions that are frequented less so. The correlation dimension offers perhaps a more relevant measure of the dimension in the context of time series (more specifically dynamical systems) because it takes into account the actual dynamical coverage of X.

7.5.4 Correlation dimension of the Lorenz attractor

To see an application of computing the correlation dimension we will approximately compute the correlation dimension of the well-known Lorenz attractor. The Lorenz attractor sits within three-dimensional phase space and is generated by the Lorenz equations

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z.\end{aligned}$$

These equations form a three-dimensional coupled system of ordinary differential equations (ODEs). Lorenz, a mathematician and meteorologist, used these equations as a basic model for the dynamics in weather systems. Specifically, these equations relate the temperature variations and rate of convection of a two-dimensional cross-section of fluid uniformly warmed from below and cooled from above; see [438] for a thorough account. For parameter values near $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$, the Lorenz system has chaotic solutions, and the attractor assumes its famous butterfly shape, as in Figure 7.8. For the computation of the correlation dimension we will use a variety of tools and packages provided by Python, most notably `numpy`, `sklearn`, `scipy`, and `matplotlib`.

We first generate the Lorenz attractor by initializing a point in phase space and then using `odeint` from the `scipy` package to evolve the point in space along an orbit. At the end we plot the orbit to give a visualization of the attractor. The code is in Figure 7.9, and the resulting image we have already seen in Figure 7.8.

To calculate the correlation dimension of the attractor we will want to restrict our calculation to those points towards the end of the orbit so that the trajectory has had time to converge upon the attractor. We then define the correlation integral and compute it for a range of ε near zero. We then fit a linear model to the log-log plot, which tells us that the slope of the fitted model, and thus the correlation dimension, is approximately 2.04; see the code in Figure 7.10.

We can see how well a linear model fits the data by generating the aforementioned log-log plot, using the code in Figure 7.11, resulting in Figure 7.12. Our calculation yields that the attractor's correlation dimension is approximately 2.04 which agrees with our geometric intuition. It should be noted that that there is an upper bound to correlation dimension that one can calculate based on the number of points in X. Eckmann and Ruelle [139] show that if one uses n points to calculate correlation dimension of X then the maximum dimension one can calculate is approximately $2\log_{10}(n)$.

An efficient box-counting-based method for the improvement of fractal dimension estimation accuracy is introduced in [311]. A method to estimate the correlation dimension of a noisy chaotic attractor is proposed in [424]. This method is based on the observation that the noise induces a bias in the observed distances of trajectories, which tend to appear farther apart than they are. Fractal-based methods that are assigned to the global estimators and geometric approaches are explored in [258].

7.6 Dimension reduction on the Grassmannian

One of the major challenges in data science is capturing the essence of a phenomenon using data as a proxy. For example, changes in illumination can make the same object look very different to pattern recognition algorithms which is problematic for machine learning. One solution is to collect a set of points in Euclidean space and to compute their span, and hence their vector subspace. As evidenced by our pumpkin example in Section 7.3.3, the variation in illumination state of the object is now encoded into a low-dimensional subspace. Now the task is to exploit the geometry of this framework.

```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
from mpl_toolkits.mplot3d import Axes3D

# Set default parameters
rho = 28
sigma = 10
beta = 8/3

# Define the system of ODEs
def f(X, t):
    x, y, z = X
    return (sigma * (y - x),
            x * (rho - z) - y,
            x * y - beta * z)

# Pick initial point
X0 = [1.0, 1.0, 1.0]

# Specify time range
# (time_start, time_end, time_step)
t = np.arange(0.0, 1000.0, 0.01)

# Integrate the ODE to obtain an orbit
Xt = odeint(f, X0, t)

# Plot the Lorenz attractor
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(Xt[:, 0], Xt[:, 1], Xt[:, 2], c='b')
plt.show()
```

FIGURE 7.9: Python code for generating the Lorenz attractor shown in Figure 7.8

The Grassmannian (or Grassmann manifold) $G(k, n)$ is a $k(n-k)$-dimensional parameterization of all k-dimensional subspaces of $\mathbb{R}^n$. It is natural tool for representing a set of points as a single abstract point, i.e., a subspace, on this manifold. Note now that we have moved data to a manifold and do not require that the data itself initially reside on a manifold, as many manifold learning methods require. As an example, a set of 10 images in $\mathbb{R}^{345600}$ creates a point $\alpha \in G(10, 345600)$ represented by a basis for this

```
# Import additional libraries
import sklearn.metrics as metrics
from sklearn.linear_model import LinearRegression

# Define the correlation integral
def C(distances,eps):
    n = distances.shape[0]
    return np.sum(distances<eps)/(n**2)

# Specify range for epsilon
eps_range = np.arange(.3, 5, .05)

# Initialize vector of correlation integrals
corr_int = np.zeros_like(eps_range)

# Compute distance matrix from time-series
# From t=750 to t=1000
d = metrics.pairwise_distances(Xt[75000:])

# Compute correlation integrals for a range of epsilons
for i, eps in enumerate(eps_range):
    corr_int[i] = C(d, eps)

# Compute respective logarithms
x = np.log(eps_range)
y = np.log(corr_int)

# Fit linear model for approximating correlation dimension
X = x[:,None] # Refit to 2-d array for LinearRegression
lin_reg = LinearRegression().fit(X, y)

# Compute approximate correlation dimension
corr_dim = lin_reg.coef_ # Slope of line
print("Correlation dimension is", corr_dim)
```

FIGURE 7.10: Python code for computing the correlation dimension discussed in Section 7.5.4. It assumes the code from Figure 7.9 has been run.

set. Another set of 10 images creates another point, again represented by a spanning set, $\beta \in G(10, 345600)$. This leads us to be interested in measuring the distance $d(\alpha, \beta)$.

Why is this framework useful? A set of k n-dimensional data points in Euclidean space is modeled as a subspace, i.e., a point on $G(k, n)$, as in Figure 7.13. Computing distances between these points on the manifold depends

```
# Import additional library
from matplotlib import rc

# Setup TeX environment with pyplot
rc('font',**{'family':'sans-serif',
             'sans-serif':['Helvetica']})
rc('text', usetex=True)

# Plot correlation integrals with linear model
plt.plot(x, y, 'go--', linewidth=2,
         markersize=5,label="Correlation Integral")
plt.plot(x, lin_reg.predict(X), '-', linewidth=2,
         markersize=5, c='k', label="Linear Regression")
plt.legend()
plt.xlabel("log($\epsilon$)")
plt.ylabel("log(C($\epsilon$))")
plt.annotate('slope= '+str(np.around(corr_dim.item(),2)),
             (1, -8), size=12)
```

FIGURE 7.11: Python code for plotting the data and model discussed in Section 7.5.4. It assumes the code from Figure 7.10 has been run.

on computing principal angles between subspaces [192]. For example, given two $n \times k$ data matrices X and Y, we determine a representation for the span of each set of vectors on $G(k, n)$ with the QR-decomposition

$$X = Q_X R_X, \quad Y = Q_Y R_Y,$$

where the matrices Q_X, Q_Y are orthonormal bases for the column spaces $\mathcal{R}(X)$ and $\mathcal{R}(Y)$, respectively. An SVD computation produces the principal angles θ_i $(i = 1, \ldots, k)$ between $\mathcal{R}(X)$ and $\mathcal{R}(Y)$, i.e.,

$$Q_X^T Q_Y = U \Sigma V^T,$$

with

$$\cos\theta_i = \Sigma_{ii}.$$

(Readers who would benefit from a review of the QR decomposition should see Section 3.2.5.2 and for a review of SVD, see Section 3.2.6.)

The geometric structure of the Grassmann manifold allows us to compute pairwise distances between its points (subspaces) using one of the metrics or even pseudometrics defined there [1, 91, 141]. For instance, if $0 \leq \theta_1 \leq \theta_2 \leq \ldots \leq \theta_k \leq \pi/2$ are the principal angles between the column spaces $\mathcal{R}(X)$ and $\mathcal{R}(Y)$, then the chordal distance is given by

$$d_c(\mathcal{R}(X), \mathcal{R}(Y)) = \sqrt{\sin^2\theta_1 + \sin^2\theta_2 + \cdots + \sin^2\theta_k},$$

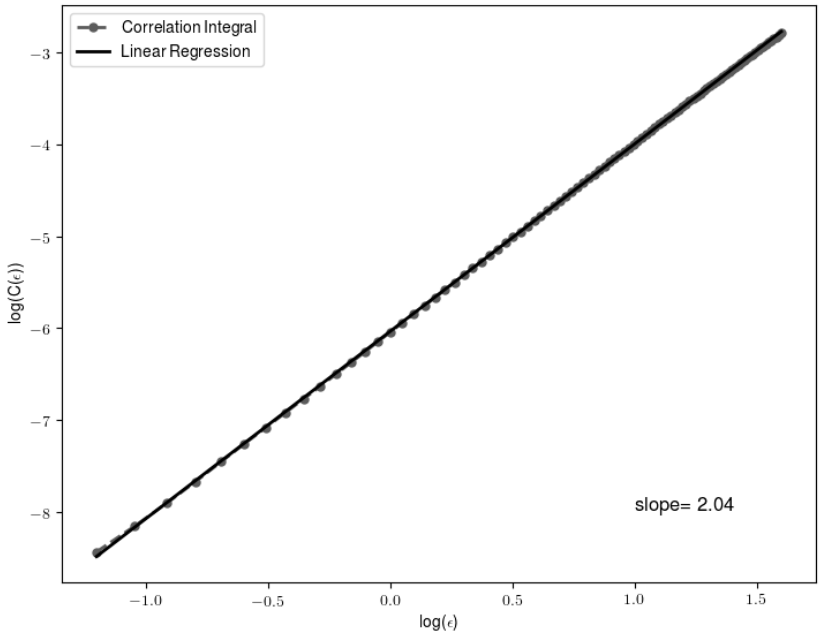


FIGURE 7.12: log-log plot of the correlation integrals $C(\varepsilon)$ vs. ε for the Lorenz attractor. The slope here is the approximate correlation dimension of the attractor.

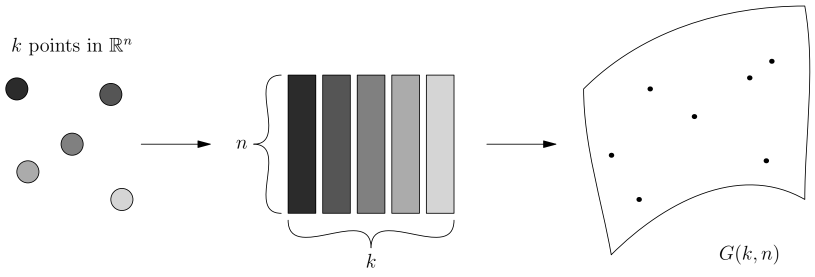


FIGURE 7.13: Schematic construction of points on a Grassmannian manifold. From [94], used with permission.

the geodesic distance is given by

$$d_g(\mathcal{R}(X), \mathcal{R}(Y)) = \sqrt{\theta_1^2 + \theta_2^2 + \cdots + \theta_k^2},$$

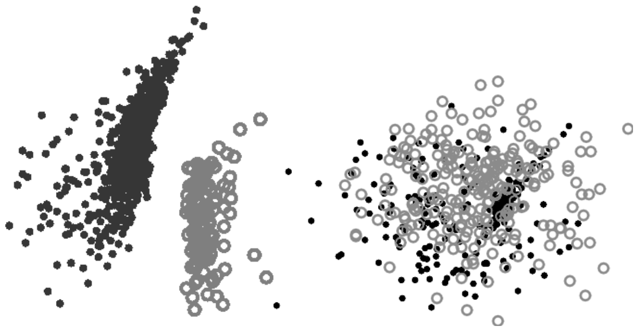


FIGURE 7.14: An example of two-dimensional data representations via the Grassmannian framework (a) that separates the two data classes versus (b) PCA projection of individual points, which leaves the two classes mixed in two dimensions. From [94], used with permission.

and for $0 < l < k$ a pseudometric can be defined as

$$d_l(\mathcal{R}(Y), \mathcal{R}(Y)) = \sqrt{\theta_1^2 + \theta_2^2 + \cdots + \theta_l^2}.$$

See [108] for additional details.

Since we can measure distances between points on the Grassmannian, we can use multidimensional scaling as described above to build a configuration of points in Euclidean space. The striking result of merging these ideas is shown in Figure 7.14. Hyperspectral pixels in 200 dimensions associated with two different classes of material are completely mixed when projected using PCA. In contrast, if the variability of the data is captured using the Grassmannian trick, the data is separable. See another example in [94]. This represents a very powerful preprocessing for subsequent machine learning algorithms.

Using this approach we can actually "see" subspaces as low-dimensional points in Euclidean space. Moreover, according to [108], $G(k, n)$ with the chordal distance d_c can be isometrically embedded into $\mathbb{R}^D$, where D is independent of k. Note that even if d_c provides distance-preserving embeddings, while d_g and d_l do not, it is also computationally useful to use both the geodesic metric and pseudometric to generate the distances between points on $G(k, n)$ for our setting. It is interesting that a pseudometric d_l, computed from the smallest l principal angles, can result in better class separability in low-dimensional Grassmannian embeddings compared to the other distances. See [94] as an example of a classification problem on Grassmanians.

Having organized data on an embedded Grassmanian, one can apply machine learning techniques for further experimentation in the convenient setting of low-dimensional Euclidean space. Additional dimension reduction can be achieved by using feature selection approaches such as sparse support vector machines (sparse SVM or SSVM) [94]. Note that if SVM, SSVM, or any other supervised machine learning method is used, one can utilize Landmark MDS (LMDS) to be able to embed new testing points in the same Euclidean space as training data [122]. This can be done by applying MDS to the manifold distances computed between landmark points only, and then by applying distance-based triangulation to find configuration for the rest of the points, including both training points and new points for testing, whenever needed.

The Grassmannian framework described above can be used in combination with topological data analysis (TDA), in particular, with its powerful tool *persistent homology* (PH), which has gained a lot of attention in recent years [143, 375, 523]. PH is a method that measures topological features of the underlying data space, such as the numbers of connected components, loops, and trapped volumes, persistent at multiple scales. Since PH can be used to study metric spaces, the Grassmannian, equipped with its metrics, provides a convenient setting for the following approach: first, organize (potentially) large volumes of high-dimensional data as points on $G(k, n)$, retaining data variability and structure, then generate and interpret PH output. See, for example, [95], as well as Chapter 10 in this book.

We conclude this section by remarking that application of the Grassmannian has become widespread in computer vision and pattern recognition. For example, we find applications to video processing [214], classification, [90, 208, 489, 490], action recognition [24], expression analysis [312, 454, 472], domain adaptation [286, 382], regression [227, 430], pattern recognition [328], and computation of subspace means [88, 335]. More recently, Grassmannians have been explored in the deep neural network literature [235]. (This text covers neural networks in detail in Chapter 9.) An interesting related idea concerns subspace packing problems [108, 289, 448]. The geometric framework of the Grassmannian has been made more attractive by the development of numerical tools [2, 141, 175].

7.7 Dimensionality reduction in the presence of symmetry

Over the last century and a half, a continual theme in both mathematics and physics has been the use of symmetry to simplify problems. The beginnings of a similar story can be told in data science where it has become increasingly clear that exploiting the symmetries inherent in a data type can lead to more robust and efficient algorithms. One of the most celebrated examples of this phenomenon is the convolutional neural network (CNN), whose exceptional performance on image recognition tasks helped spark the deep learning revolution. The key property of a CNN is that its architecture is designed to be *invariant* to translations of objects in a sample image, as we will cover later in Section 9.4. For example, once the network learns to recognize a dog on the right side of an image, it will automatically learn to recognize a dog on the left side of an image. Thus even before training, the model understands that it should ignore certain variations in the data that are unimportant to classification. Since representation theory is the study of symmetry using the tools of linear algebra, many of the ideas from this domain play a key, if understated, role in data analysis.

In this section, we give some specific examples of how representation theory can play a guiding role in dimensionality reduction. Our presentation is unconventional in that we stick to an algebraic description, since it is from this perspective that the transition to more diverse groups is easiest. We start by acknowledging the most ubiquitous example of representation theory in dimensionality reduction: the discrete Fourier transform (DFT). For diverse modalities where data is represented in the form of vectors (a prominent example being time series) it is common to take a DFT and then cut out frequencies that are known to correspond to noise or unwanted signal. Suppose we have a data point $x = (x_0, \dots, x_{n-1}) \in \mathbb{C}^n$ and that C_n is the cyclic group of order n. Then C_n contains n elements which we denote as $C_n = \{1, g, g^2, \dots, g^{n-1}\}$. Recall that the group algebra $\mathbb{C}[G]$ of a finite group G is the algebra of formal sums of elements of G with multiplication given by the usual multiplication of group elements [301, II, §3]. We can interpret x as an element of the group algebra $\mathbb{C}[C_n]$

$$x = \sum_{k=0}^{n-1} x_k g^k.$$

More commonly x would be interpreted as a function in $\mathrm{Hom}(C_n, \mathbb{C})$, but as algebras $\mathbb{C}[G] \cong \mathrm{Hom}(G, \mathbb{C})$ for any finite group G, where the usual algebra multiplication holds on the left and the convolution of functions holds on the right. The DFT is nothing other than the well-known decomposition of the group algebra of a finite group into a product of matrix algebras [428, Part II, Section 6]. (In the most general setting this is known as the Artin-Wedderburn decomposition.) Of course in this case, since the symmetry group C_n is commutative, the result is just a direct product of numbers (the Fourier coefficients). Algebraically then, removing unwanted frequency components of a signal is equivalent to zeroing out certain matrix algebras in the Artin-Wedderburn decomposition.

Of course, harmonic analysis is much more than a specialized area of representation theory, but the ways that Fourier theory is used in dimensionality reduction serve as a model for the ways that the representation theory of other groups can be applied in this area. But note that for a noncommutative finite group G, the data at the other end of the Fourier transform is a direct product of nontrivial matrices. As is usual in representation theory, many of the observations made here and below have analogues when the finite group G is replaced by a compact Lie group.

In the remainder of this section, we will show how the ideas above interact with two classical dimensionality reduction algorithms in the presence of symmetry. Following [295], we first consider the most fundamental technique for dimensionality reduction, PCA (Section 7.3). Suppose that we have a dataset X that sits in a $\mathbb{C}$-vector space V. Like many dimensionality reduction algorithms, PCA can be reduced to finding the eigendecomposition of a specific linear operator, in this case, the ensemble average covariance matrix $C : V \to V$. Now suppose that a group G (finite or compact) acts on V and that the set of possible data points U (not restricted to the data points X that have actually been observed) is stable under this action (so that possible data points get sent by C to other possible data points). This symmetry assumption implies that C is G-equivariant on V. It then follows from basic representation theory that subrepresentations of G in V are stable under the action of C. Thus if we understand the representation theory of G, then we automatically have a partial PCA decomposition of V with respect to dataset X, since the eigenspaces of C are constrained to be direct sums of the irreducible representations of G in V. As a further consequence, the possible dimensions of the eigenspaces of C are bounded below by the dimensions of the irreducible representations of G appearing in V.

The authors of [295] go on to show that symmetry gives a definite computational advantage when applying PCA. In the common case where V decomposes into a direct sum of regular representations of G, then the ratio $\alpha(G)$ of the number of flops[1] required to do PCA without utilizing the symmetries of X over the number of flops required when utilizing the symmetry of X is

$$\alpha(G) = \frac{|G|^3}{\sum_\rho \dim(\rho)^3},$$

where the term in the denominator is the sum of cubes of the dimensions of all irreducible representations of G.

A similar phenomenon occurs when one considers the MDS algorithm (Section 7.3.5) in the presence of symmetry. Recall that the input to this algorithm is a distance matrix D that gives the pairwise distances between each point in a dataset X. Let $m : X \times X \to \mathbb{R}_{\geq 0}$ be the metric which gives D. In [290] it is shown that when a finite group G acts on X and m is invariant with respect to the action of G so that for $g \in G$ and $x, y \in X$, $m(gx, gy) = m(x, y)$, then the eigenspaces of the MDS operator are in fact representations of G. Again, as a consequence of understanding the representation theory of G, we can automatically make statements about the best way to embed X in Euclidean space while approximating m by the Euclidean metric.

We end by listing a few basic examples of datasets that naturally have the action of a group attached.

[1]A flop, or "floating point operation," was introduced briefly in Section 3.2.2.4.

- Generalizing from the example above, there is an action of $C_n \times C_n$ on any set of $n \times n$ images. In fact X can be identified with elements of $\mathbb{C}[C_n \times C_n]$ in a manner analogous to above. The two-dimensional Fourier transform is the result of the corresponding decomposition of $\mathbb{C}[C_n \times C_n]$ into matrix algebras.
- Let X be an n-question survey where each question requires a binary answer. The set X is acted on by $C_2 \times \cdots \times C_2$ (where there are n terms in this product corresponding to the n questions). The analogue of the DFT basis in this case is called the Walsh-Hadamard basis.
- Let X be a set of rankings of n objects, then X may be identified with elements of the symmetric group $\mathcal{S}_n$.
- It has become increasingly clear that many types of data can be realized as points on a Grassmann manifold $G(k, n)$ with $0 < k \leq n$. Being a homogeneous space, there is a natural action of the orthogonal group $O(n)$ on $G(k, n)$. An example of a type of dataset that can naturally be interpreted as points on the Grassmannian are illumination spaces. See [91] for further details, and the next section in this chapter for many additional applications.

The reader interested in a more in-depth treatment of the applications of representation theory to dimensionality reduction and data science in general should consult the foundational work in this area [128]. For a more recent survey on how representation theory appears in machine learning, see [275].

7.8 Category theory applied to data visualization

UMAP stands for Uniform Manifold Approximation and Projection. In contrast to other dimension reduction algorithms such as PCA that preserve the global structure of the dataset, UMAP, along with t-SNE and Laplacian Eigenmaps, seeks to instead preserve the local geometry of a dataset. The advantage is that individual structures sitting within the dataset will remain intact in the projected space. For example, if the dataset contains a set of points which lie on a topological torus they will remain on a torus when projected, as in Figure 7.15. In the context of visualization, one may be able to recognize separate and familiar topological structures within the data. In the context of classification and preprocessing, each class may live on its own separate submanifold of the original data space. Hence, we would like to retain that classification by submanifolds when moving to a lower dimensional space where further analysis is to be performed.

UMAP is unlike other popular dimension reduction algorithms in that it will project a dataset so that it is approximately uniformly distributed in

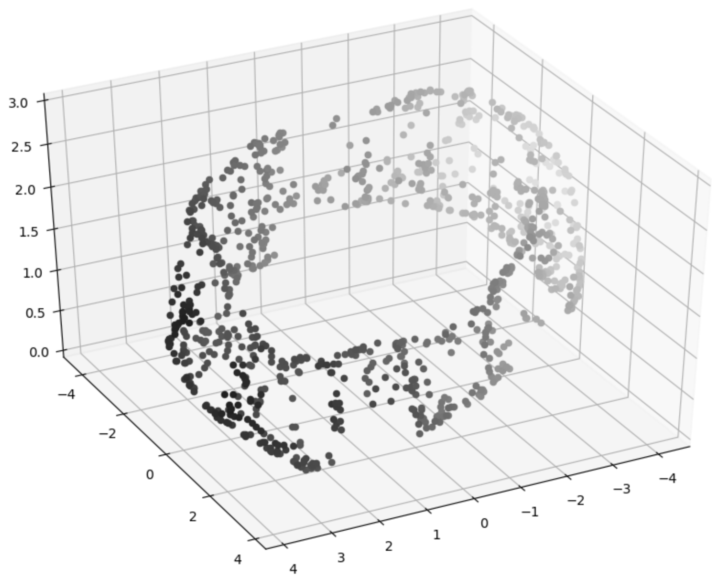


FIGURE 7.15: Three-dimensional projection via UMAP of a 2-torus embedded into 100-dimensional space. The torus was generated by first independently sampling two angle coordinates uniformly from 0 to 2π, parametrizing the standard torus in three-space with inner radius $r = 3$ and outer radius $R = 10$, and then applying an orthogonal transformation to isometrically embed the torus into 100 dimensions.

the embedding space regardless of the distribution of the original data. A mathematician may be intrigued to discover that such a practical algorithm is built from abstract category-theoretic foundations. There are several key components to the UMAP algorithm:

ULA: **U**niform **l**ocal **a**pproximations to the global Reimannian distance at each point.

LFR: Association of a **f**uzzy simplicial **r**epresentation to each uniform **l**ocal approximation.

GFR: Generation of a **g**lobal **f**uzzy **r**epresentation by taking a fuzzy union over all local fuzzy representations.

PCE: **P**rojection of the original data to a lower-dimensional space that minimizes the **c**ross-**e**ntropy between corresponding fuzzy representations.

ULA: Let $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^m$ be a dataset and assume that X lies on some embedded submanifold $\mathcal{M}$ of $\mathbb{R}^m$, not necessarily distributed uniformly with respect to the induced metric. The goal is to put a Riemannian metric g_X (smooth choice of inner product on the tangent spaces) on the manifold $\mathcal{M}$ so that X is uniformly distributed with respect to the geodesic distance d_X. One approximate way to do this is to have it so the unit balls B_i centered about each x_i contain the same number of points of X. UMAP does this by locally approximating the geodesic distance $d_X(x_i, x_j)$ out of x_i via inversely scaling the Euclidean distance $d_{\mathbb{R}^m}(x_i, x_j)$ by the maximum distance σ_i from x_i to its k nearest neighbors,

$$d_X(x_i, x_j) \approx \frac{1}{\sigma_i} d_{\mathbb{R}^m}(x_i, x_j).$$

Therefore a unit ball in $\mathcal{M}$ around x_i will almost always contain exactly k points. In order to ensure local connectivity of the embedded manifold we modify the scaled Euclidean distance by subtracting off the distance, ρ_i, from x_i to its nearest point in X. At this point we've approximated only distances out of the point x_i, so a star of distances where x_i is the center of the star. Since all of X will be covered by these star-like distances, and we plan to assimilate them into a global construction, we define the distances between the outer points of each individual star as infinity. The choice of infinity will be justified later when constructing the global fuzzy representation; essentially an infinite distance corresponds to a lack of 1-simplex joining the associated points in the fuzzy representation. We can now give the definitions of extended pseudo-metrics d_i on X, which capture the geometry local to each x_i. Explicitly, define the extended pseudo-metric spaces $d_i : X \times X \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ as

$$d_i(x_j, x_k) = \begin{cases} \frac{1}{\sigma_i}\left(d_{\mathbb{R}^m}(x_j, x_k) - \rho_i\right) & \text{if } j = i \text{ or } k = i \\ \infty & \text{otherwise.} \end{cases}$$

LFR: Having our different metrics d_i which capture the geometry near each point x_i, we would like to combine these into a single metric. In general, there is no natural way to build a metric space from other metric spaces with the same underlying set. In our case, this is due to incompatibilities between local representations. For example, in general,

$$d_i(x_i, x_j) \neq d_j(x_j, x_i),$$

due to the differences in scale for each neighborhood. By associating to each metric space d_i the "fuzzy" simplicial set $\tilde{X}_i$ via the "functor" **FinSing**, we can losslessly transfer the metric data to a different category where assimilating all of the local distances becomes a simple operation. Let **Fin-sFuzz** denote the category of fuzzy simplicial sets over a finite set and **Fin-EPMet** be the category of extended pseudo-metric spaces over a finite set. Then **FinReal** is the functor that takes a fuzzy simplicial set to its associated metric space, and **FinSing** is the functor that takes a metric space to its associated fuzzy simplicial set.

Abstractly, a fuzzy simplicial set F is a contravariant functor $F : \Delta \times I \to$ **Set** where I is the poset category on the unit interval $[0, 1]$, Δ is the simplex

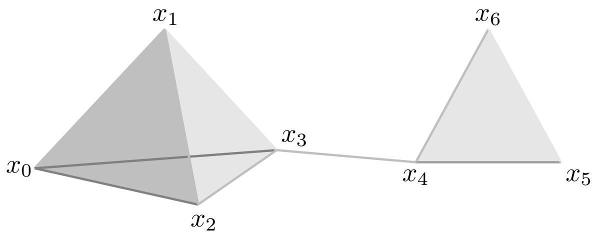


FIGURE 7.16: An example of a three-dimensional fuzzy simplicial set with probabilities on the faces represented by various degrees of shading. Here we omit a simplex if the associated probability is 0.

category, and **Set** is the category of sets. To each $([n], a) \in \Delta \times I$ we associate the set $F_n^{\geq a}$ of n-simplices with membership strength at least a. Practically, we can think of a fuzzy simplicial set as a simplicial complex where each simplex in the space has a probability associated to it, as in Figure 7.16. The actual mathematical theorem given in [339, 439] that makes this lossless transfer precise is that there are natural bijections between the hom-sets

$$\hom_{\textbf{Fin-EPMet}}(\textbf{FinReal}(X), Y) \cong \hom_{\textbf{Fin-sFuzz}}(X, \textbf{FinSing}(Y))$$

for any $X \in$ **Fin-sFuzz** and $Y \in$ **Fin-EPMet**.

Whereas traditional simplicial complexes tell you about the topology of a space, the added data of probability extracts the geometric, or distance, structure of the space. We can construct $\tilde{X}_i := \textbf{FinSing}(X_i)$ explicitly by first building an $n-1$-dimensional simplex with vertex set X, and a probability assigned to each face F via the formula

$$\mathbb{P}(F) = \min_{x_i, x_j \in F} \left\{ e^{-d_i(x_i, x_j)} \right\}. \tag{7.12}$$

Notice that a face must have probability less than or equal to the probability of any one of its subfaces. For UMAP this has the implication that each $\tilde{X}_i$ is a 1-dimensional simplicial set, or a weighted graph, due to the presence of infinite distances in (X, d_i) $(e^{-\infty} = 0)$. Hence, the problem of joining various metric spaces into a single metric space becomes the problem of combining the edge probabilities of various weighted graphs.

GFR: Since each weighted graph $\tilde{X}_i$ has the same vertex set X, we can take a fuzzy, or probabilistic, union of the weighted graphs $\tilde{X}_i$ by superimposing the $\tilde{X}_i$ on top of one another and resolving the probabilities on each edge. Thus we require a function that will combine different probabilities of the same edge to a single probability. Such a function $\xi : [0,1] \times [0,1] \to [0,1]$ is called a t-conorm. UMAP uses the probabilistic t-conorm

$$\xi(a, a') = a + a' - a \cdot a'.$$

The fuzzy representation of X will then be the weighted graph $\tilde{X} := \bigcup_i^{\xi} \tilde{X}_i$. Since each edge $[x_i, x_j]$ will have only two probabilities associated to it, namely $e^{-d_i(x_i,x_j)}$ and $e^{-d_j(x_j,x_i)}$, we can compute the above union with simple matrix operations. If we let P denote the matrix with weights $p_{ij} := e^{-d_i(x_i,x_j)}$, then $\tilde{X}$ will be the weighted graph with vertex set X and weighted adjacency matrix given by

$$\bar{X} = P + P^T - P \circ P^T$$

where $P \circ P^T$ denotes the Hadamard (point-wise) product of P with its transpose P^T. Notice that $\tilde{X}$ will represent a metric space (X, d_X) where X appears to be approximately uniformly distributed. Essentially UMAP averages each uniform local representation to obtain an approximately uniform global representation.

PCE: Pick an embedding dimension d. Let $F : X \to \mathbb{R}^d$ be any mapping with $y_i := F(x_i)$ and $Y := \operatorname{Im}(F)$. We can view Y as a metric space with the induced Euclidean metric. Again, to ensure local connectedness of our embedded space, we adjust the Euclidean distance by subtracting off the expected minimum distance σ between points. The parameter σ chosen is problem-dependent and can have a significant effect on the embedding; see [339]. Let $\tilde{Y} := \textbf{FinSing}(Y)$ be the fuzzy representation of Y. Note that since Y has finite distances defined between every pair of points, $\tilde{Y}$ will be a $(n-1)$-dimensional fuzzy simplicial set. To compare the two fuzzy representations, we approximate $\tilde{Y}$ with its 1-skeleton to obtain another weighted graph. UMAP seeks an optimal projection F_* in the sense that F_* minimizes the cross-entropy between the weighted graphs $\tilde{X}$ and $\tilde{Y}$,

$$F_* = \arg\min_F \left\{ \sum_{x_i \neq x_j} \left(\tilde{X}_{ij} \log \left(\frac{\tilde{X}_{ij}}{\tilde{Y}_{ij}} \right) + (1 - \tilde{X}_{ij}) \log \left(\frac{1 - \tilde{X}_{ij}}{1 - \tilde{Y}_{ij}} \right) \right) \right\}.$$

Note: $\tilde{Y}$ depends on the embedding F.

UMAP initializes the embedding F with Laplacian eigenmaps and then uses stochastic gradient descent (SGD, covered in Chapters 8 and 9) along with negative sampling in order to minimize the cross-entropy. Like UMAP, Laplacian eigenmaps also approximately preserve the local geometry so they serve as a good initial configuration for the descent. SGD amounts to sampling edges $[x_i, x_j]$ at random and then moving the pair of points (y_i, y_j) in the direction of the gradient of the associated summand in the cross-entropy. In order to calculate the gradient of any summand we need to approximate the weight function $\Phi : \mathbb{R}^d \times \mathbb{R}^d \to [0,1]$ on the edges of $\tilde{Y}$ given by

$$\Phi(x,y) = \begin{cases} e^{\|x-y\|_2 - \sigma} & \text{if } \|x-y\|_2 \geq \sigma \\ 1 & \text{if } \|x-y\|_2 < \sigma \end{cases}$$

with a differentiable function $\Psi : \mathbb{R}^d \times \mathbb{R}^d \to [0,1]$. UMAP smoothly approximates Φ with the function

$$\Psi(x,y) = \left(1 + a\left(\|x-y\|_2^2\right)^b\right)^{-1},$$

where the hyperparameters a and b are chosen by nonlinear least squares fitting. This completes the algorithm.

Notice that in the UMAP algorithm all the work of minimization happens in the category of **Fin-sFuzz**, and not in **Fin-EPMet**. Actually calculating the metric space (X, d_X) is not required to produce a lower dimensional embedding of it. This is the power of categories; the ability to transfer a problem from one mathematical framework to another, using tools more natural for solving the problem. See Figures 7.15 and 7.17 for some examples of UMAP embeddings.

UMAP can be implemented via the Python package `umap-learn`. Figure 7.18 shows the code to generate a dataset on a torus embedded in $\mathbb{R}^{100}$. Then Figure 7.19 shows the code that applies the UMAP algorithm to reduce to $\mathbb{R}^3$, recovering the torus shape from the data, resulting in Figure 7.15. In Figure 7.19, `n_neighbors` denotes the k in k-nearest neighbors, `min_dist` denotes the expected minimum distance σ, `n_components` denotes the dimension d of the projected space, and `metric` denotes the metric for measuring distances in the original space.

7.9 Other methods

In this section we briefly survey some additional methods of interest. They can be read in any order, or returned to after perusing Section 7.10.

7.9.1 Nonlinear Principal Component Analysis

An auto-encoder feed-forward neural network is a mapping that has been optimized to approximate the identity on a set of training data [277, 278, 368]. The mapping is comprised of a dimension-reducing mapping G followed by a dimension-increasing reconstruction mapping H, i.e.,

$$F(x) = (H \circ G)(x).$$

We call $G : U \subset \mathbb{R}^n \to V \subset \mathbb{R}^m$ the *encoder* and $H : V \subset \mathbb{R}^m \to U \subset \mathbb{R}^n$ the *decoder*. The network learns the functions G and H to minimize the loss function

$$\sum_i \|x^{(i)} - F(x^{(i)})\|_2^2 \tag{7.13}$$

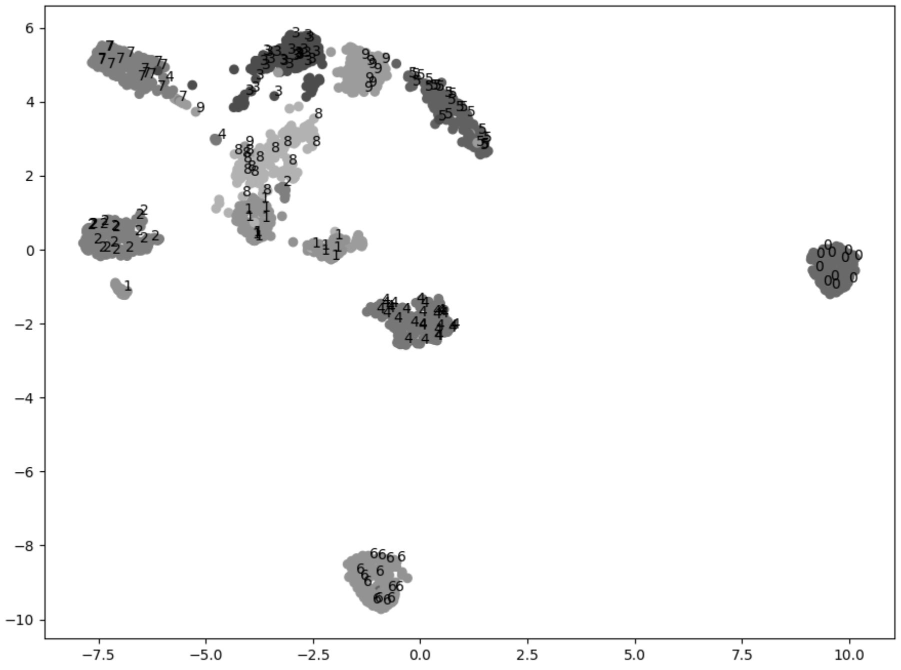


FIGURE 7.17: Two-dimensional UMAP of the NIST digits dataset provided by Python module scikit-learn. The compressed 8×8 gray scale images of handwritten digits came from a total of 43 people. The images are then flattened and represented as vectors in 64-dimensional space where distances are measured by the Euclidean distance metric.

```
# Import libraries
import numpy as np
from scipy.stats import ortho_group
import matplotlib.pyplot as plt
import matplotlib
from mpl_toolkits.mplot3d import axes3d
import umap

# Initial parameters
r = 3 # Inner radius
R = 10 # Outer radius
xyz = np.array([]) # Initialize (x,y,z) coordinates of torus
N = 1000 # Number of points
# Random orthonormal frame in 100-dimensional space:
orth = ortho_group.rvs(100)
orth3 = orth[0:3, :] # 3-dimensional subframe

for i in range(N):
    # Generate random angles for parameterization
    theta = 2*np.pi*np.random.rand()
    tau = 2*np.pi*np.random.rand()

    # Calculate position vector for center ring
    xy = np.array([R*np.cos(theta), R*np.sin(theta)])

    # Calculate (x,y,z) coordinate for parameterization
    z = r*np.sin(tau)
    w = r*np.cos(tau)
    xy = xy + (w/R)*xy

    # Append point to array of points
    if xyz.shape[0] == 0:  # If array is empty
        xyz = np.array([xy[0], xy[1], z])
    else:  # If array is non-empty
        xyz = np.vstack((xyz, np.array([xy[0], xy[1], z])))

# Rotate into 100-dimensional space
# (recall, orth3 is some orthogonal transformation)
X = np.matmul(xyz, orth3)
```

FIGURE 7.18: Python code to sample data from a torus and embed it randomly in $\mathbb{R}^{100}$, so that we might attempt to recover the structure with the UMAP algorithm, as in Figures 7.19 and 7.15.

```
# UMAP algorithm
model = umap.UMAP(
        n_neighbors=50,
        min_dist=0.1,
        n_components=3,
        metric='euclidean',
        )
u = model.fit_transform(X)

# Initialize plot
fig_umap = plt.figure()
ax_umap = fig_umap.add_subplot(111, projection='3d')

# Color by scaling (x,y,z) coordinates to RGB values
c = np.zeros_like(xyz)
c[:,0] = ((xyz[:,0]+r+R)/(2*(r+R)))
c[:,1] = ((xyz[:,1]+r+R)/(2*(r+R)))
c[:,2] = ((xyz[:,2]+r)/(2*r))

# Plot
for i in range(N):
    color = matplotlib.colors.to_hex(c[i,:])
    ax_umap.scatter3D(u[i, 0], u[i, 1], u[i,2],c=color)

plt.show()
```

FIGURE 7.19: Python code to apply the UMAP algorithm to the data generated by the code shown in Figure 7.18. This code assumes that the code in that figure has been run. The resulting image appears in Figure 7.15.

for a given set of data points in the ambient space $\{x^{(1)}, x^{(2)}, \ldots, x^{(P)}\}$. Hence, the autoencoder learns a function $F = H \circ G$ such that $F(x^{(i)}) \approx x^{(i)}$. The reduction mapping G transforms the point $x^{(i)}$ to a latent representation $y^{(i)} = G(x^{(i)}) \in V \subset \mathbb{R}^m$.

If we are interested in visualization, then we take $m = 2$ or $m = 3$. The decoder ensures that the encoder is faithful to the data point, i.e., it serves to reconstruct the reduced point to its original state $x^{(i)} \approx H(y^{(i)})$. If the mappings H and G were restricted to be linear, then this approach discovers the same subspaces as PCA, hence the term "nonlinear PCA." For more information on how neural network functions are nonlinear, see Chapter 9.

7.9.2 Whitney's reduction network

From a mathematician's perspective, Hassler Whitney might arguably be viewed as the world's first dimension reducer. His theoretical work provides insight into the mechanisms of dimension reducing mappings such as the autoencoder described in the previous section. For reasons we shall see, Whitney restricts the mapping G above to be linear. Whitney's data was a manifold and his famous *easy embedding theorem* (which we review in Section 7.10.1 for interested readers) identifies the conditions required to realize a copy of a manifold in Euclidean space. It is also the blueprint for a data reduction algorithm that bears some resemblance to the autoencoder neural network described in Section 7.8. To a data scientist, Whitney's formulation can be interpreted as representing a dataset as the graph of a function [68, 69].

Loosely speaking, a manifold of dimension m can be projected into a Euclidean space of dimension $2m + 1$ such that the image of the projection can be reconstructed in the ambient space without loss; for details see [222]. If G is linear and H is nonlinear in the autoencoder neural network, then we are matching Whitney's blueprint for data reduction [68, 69]. The secant methods described above for estimating the embedding dimension are then very effective for finding good projections. Now we assume that $P = UU^T$ is a projection found using secant a method. Then a data point can be decomposed as

$$x = Px + (I - P)x.$$

We are interested in the dimension-reduced representation $\hat{p} = U^T x$. The residual of this projection, i.e., the part of x not captured in the linear space spanned by the columns of U is $\hat{q} = U_\perp^T x$ where $U_\perp$ is a mapping to the null space of P. In this setting we use a neural network to encode the mapping $\hat{q} = f(\hat{p})$. In other words, we have the reconstruction

$$x \approx U\hat{p} + U_\perp f(\hat{p}).$$

This can be thought of as parameterizing the data in terms of a linear operator L and nonlinear operator N as

$$x \approx L(\hat{p}) + N(\hat{p}).$$

Or, again, in terms of a graph $(\hat{p}, f(\hat{p}))$ in the reduced coordinate system. The conditions under which the mapping $\hat{p} = f(\hat{q})$ exists are governed by Whitney's theorem. For additional details see [68, 69].

7.9.3 The generalized singular value decomposition

The generalized singular value decomposition is an important but less widely used technique for signal processing [192].

Theorem 1 GSVD [484]. *Let $A \in \mathbb{R}^{n_a \times p}$ and $B \in \mathbb{R}^{n_b \times p}$. Then there exist $U \in \mathbb{R}^{n_a \times n_a}$, $V \in \mathbb{R}^{n_b \times n_b}$, $X \in \mathbb{R}^{p \times p}$, as well as diagonal matrices $C \in \mathbb{R}^{n_a \times p}$, and $S \in \mathbb{R}^{n_b \times p}$ such that*

$$A = UCX^T \tag{7.14}$$

and

$$B = VSX^T \tag{7.15}$$

with

$$U^TU, = I, \quad V^TV = I, \text{ and } S^2 + C^2 = I.$$

Here the diagonal matrices consist of non-negative entries $C = \operatorname{diag}(c_1, \ldots, c_p)$ and $S = \operatorname{diag}(s_1, \ldots, s_p)$.

In what follows we will make some restrictive assumptions that will simplify our presentation. First, we assume that $p \leq n_a + n_b$ so that X is square. Further we assume X is invertible and define $Z = (X^T)^{-1}$ and we write the i^{th} column of Z as z_i. Also, we will assume that A and B have the same number of rows, so $n = n_a = n_b$.

The GSVD equations can be merged into the statement

$$s_i^2 A^T A z_i = c_i^2 B^T B z_i.$$

We can now simultaneously estimate dimension in the row spaces of A and B using the properties $z_i^T A^T A z_i = c_i^2$ and $z_i^T B^T B z_i = s_i^2$. Since $c_i^2 + s_i^2 = 1$ we see that each z_i contains parts of the rows of A and B as measured by c_i and s_i. A mathematical overview of the GSVD is available in [142]. The GSVD also provides an elegant solution to signal separation. In this setting we assume that there is an observed matrix X that is the result of linearly mixing a signal matrix of interest S, i.e.,

$$X = SA$$

The GSVD method allows you to recover S from X under certain assumptions; see [239] for details.

7.9.4 False nearest neighbors

A point in the delay embedding of the time series $x(t)$ into dimension M is given by

$$\mathbf{z}(t) = (x(t), x(t-T), \ldots, x(t-(M-1)T) \in \mathbb{R}^M.$$

If the embedding dimension M is too small, then it is possible that moving to dimension $M+1$ may lead to nearest neighboring points blowing apart from each other, indicating that they were *false nearest neighbors* [265]. Define the distance between a point $z \in \mathbb{R}^M$ and its nearest neighbor $\tilde{z} \in \mathbb{R}^M$ to be

$$d_M^2(t) = \sum_{k=0}^{M-1} (z(t-kT) - \tilde{z}(t-kT))^2.$$

A measure of the distance correction δ_c that results from adding the $M+1^{\text{st}}$ dimension is then

$$\delta_c = \frac{|x(t-TM) - \tilde{x}(t-TM)|}{d_M(t)},$$

where $\tilde{z}$ is the delay embedding of $\tilde{x}$. Following [265], if this measure is greater than a critical threshold $\delta_c > \tau$ one may conclude that dimension M is too small for a faithful embedding of the data. One also has to account for isolated points that do not have close nearest neighbors, as well as the presence of noise. Additional implementation details can be found in [216], and more on delay embeddings of time series is reviewed in Section 7.10.2 for interested readers.

7.9.5 Additional methods

In closing, we point the reader to useful surveys and additional algorithms on dimension estimation and reduction that reflect different perspectives from the one followed here. An overview of several basic methods for dimension reduction is given in [82] and [74]. A survey of intrinsic dimensionality, topological dimension, Fukunaga-Olsen's algorithm, fractal dimension, and MDS is provided in [78]. More recent review of the state-of-the-art methods of intrinsic dimension estimation, underlining recent advances and open problems, is given in [79]. An algorithm to estimate the intrinsic dimension of datasets based on packing dimension of the data that requires neither parametric assumptions on the data generating model nor input parameters to set is presented in [264]. Many novel applications for local intrinsic dimension estimation have been explored; see, e.g., [83]. An alternative to the UMAP algorithm for data visualization is given by centroid-encoders, a data-labeled version of autoencoders [184].

7.10 Interesting theorems on dimension

Mathematicians have already proven a lot of fundamental things about dimension. Developing algorithms from these is an active area of research. This section briefly states several interesting theorems related to dimension estimation and data reduction with the hope that the reader is induced into pursuing these ideas further in the literature.

7.10.1 Whitney's theorem

The statement of the theorem presented below follows [222]. Recall that a C^1 map is called an immersion if for every point $x \in \mathcal{M}$ we have that the derivative map $T_f : \mathcal{M}_x \to \mathcal{N}_{f(x)}$ is one-to-one.[2]

Definition 7.10 (Embedding [222]) *A map $f : \mathcal{M} \to \mathcal{N}$ is an embedding if f is an immersion which maps $\mathcal{M}$ homeomorphically onto its image.*

The following theorem can be interpreted as being about data reduction on sampled manifolds.

Theorem 2 (Whitney's Easy Embedding Theorem [499]) *Let $\mathcal{M}$ be a compact Hausdorff C^r n-dimensional manifold, $2 \leq r \leq \infty$. Then there is a C^r embedding of $\mathcal{M}$ in $\mathbb{R}^{2n+1}$.*

The proof of Whitney's theorem introduces the unit secant set. Projections away from these secants allow the realization of a diffeomorphic copy of the manifold. So, ignoring noise, any n-dimensional manifold can be realized in a Euclidean space of dimension $2n+1$ via a linear map. The reconstruction mapping is a nonlinear inverse of the projection.

7.10.2 Takens' theorem

Often data is observed as a sequence of points in time. Consider, for example, the value of the temperature outside, the velocity of a stream, or the voltage fluctuations at a point on the human scalp. These data streams can be represented as the sequence $(x(t_1), x(t_2), \ldots, x(t_n))$. This scalar sequence may be viewed as a projection of a higher-dimensional object with a distinct geometric structure. Imagine a point traveling along the unit circle but all you get to observe is the x-coordinate of the point. One can reassemble this scalar information using a device known as a *time-delayed embedding* that was proposed as a means to recover geometry from a time series [378]. So, the time series data is used to build delay vectors

$$\mathbf{z}(t) = (x(t), x(t-T), \ldots, x(t-(M-1)T)) \in \mathbb{R}^M,$$

[2]A map is said to be C^r if the first r derivatives exist and are continuous.

where T is the delay time and M is the embedding dimension. One can verify that a good choice of T and $M = 2$ make it possible to recover a topological circle from the above projection using this trick. In general, the delay T can be found from the data by looking at the first zero crossing of the autocorrelation function. The embedding dimension M can be estimated in a variety of ways, including the false-nearest-neighbor algorithm we covered in Section 7.9.4. The major theoretical underpinning of the property that delay embeddings can be used to reconstruct geometry was proved by Takens [455].

Theorem 3 (Takens [455]) *Let $\mathcal{M}$ be a compact manifold of dimension m. For pairs (φ, y), $\varphi : \mathcal{M} \to \mathcal{M}$ a smooth diffeomorphism and $y : \mathcal{M} \to \mathbb{R}$ a smooth function, it is a generic*[3] *property that the map $\Phi_{(\varphi,y)} : \mathcal{M} \to \mathbb{R}^{2m+1}$, defined by*

$$\Phi_{(\varphi,y)}(x) = (y(x), y(\varphi(x)), \dots, y(\varphi^{2m}(x)))$$

is an embedding. By smooth we mean at least C^2.

A very readable proof of this theorem may be found in [237]. This work set off a stream of papers and launched the field in dynamical systems referred to as *embedology* [421]. In the context of applications to real data, this modeling approach has at its core a dimension estimation problem.

7.10.3 Nash embedding theorems

There are differing philosophies about which aspects of geometry one should preserve when embedding datasets in lower dimensional spaces. As above with Whitney's theorem, it is appealing to seek answers in mathematical theory. The hope is that there may be the ingredients of a potentially useful algorithm.

Nash solved the open problem related to the general conditions under which a Riemannian manifold can be isometrically embedded into Euclidean space.

Theorem 4 (Nash [357]) *Any closed Riemannian n-manifold has a C^1 isometric embedding in $\mathbb{R}^{2n}$.*

Theorem 5 (Nash [358]) *Every Riemannian n-manifold is realizable as a sub-manifold of Euclidean space.*

If the manifold is compact Nash provides an embedding dimension of $n(3n + 11)/2$ while if it is noncompact his estimate is $n(n + 1)(3n + 11)/2$. These estimates were further improved in [203].

Direct numerical implementation of Nash's C^1 embedding algorithm is feasible, but it is not well behaved numerically. Another interesting direction

[3]The term "generic" restricts the theorem to a certain sane set of measurement functions; the reader should refer to [455] for details.

is to learn the geometry in the data by computing an appropriate Riemannian metric, a technique referred to as *metric learning on manifolds* [387].

There is a considerable literature on isometric embeddings that is not for the faint of heart [200]. However, it is possible that there exists current theory that could be converted into new embedding algorithms.

7.10.4 Johnson-Lindenstrauss lemma

If one would like to map a high-dimensional dataset into a lower dimensional Euclidean space while approximately preserving the relative distances between any two of the data points, the existence of such a mapping is asserted by a fundamental result named after William Johnson and Joram Lindenstrauss:

Theorem 6 (Johnson-Lindenstrauss Lemma [253]) *Let $0 < \epsilon < 1$, X be a set of m points in $\mathbb{R}^n$, and k be a positive integer such that $k \geq O\left(\frac{\ln(m)}{\epsilon^2}\right)$. Then there is a Lipschitz mapping $f : \mathbb{R}^n \to \mathbb{R}^k$ such that for all $x_i, x_j \in X$,*

$$(1-\epsilon)||x_i - x_j||^2 \leq ||f(x_i) - f(x_j)||^2 \leq (1+\epsilon)||x_i - x_j||^2.$$

In other words, the lemma says that that any dataset of m points in $\mathbb{R}^n$ can be embedded in $k \ll n$ dimensions without distorting the distances between any pair of the data points by more than a factor of $1 \pm \epsilon$. Note that k is independent of n and logarithmic in m, but naturally increases as ϵ decreases. There are versions of the lemma with improved bounds for k; see, for instance, the discussion in [118], which also contains a readable proof of the statement.

In practice, to construct such an embedding, most existing methods project the input points onto a spherically random hyperplane through the origin; consider, e.g., a technique known as random projections [51]. In this method, the original n-dimensional data is projected to a k-dimensional ($k \ll n$) space through the origin, using a random $k \times n$ matrix P whose columns are unit vectors. In matrix notation, if $X_{n\times m}$ is a data matrix of m n-dimensional points, then the projection of $X_{n\times m}$ onto a lower k-dimensional space is given by the $k \times m$ matrix $P_{k\times n}X_{n\times m}$. Note that in practice the random matrix P does not have to be orthogonal, since that may be computationally expensive, and hence, strictly speaking, $P_{k\times n}X_{n\times m}$ is not a projection if this is the case. The choice of matrix P is an important part of the method, and, typically, the entries of P are chosen to be Gaussian distributed. While this approach is conceptually simple, in practice, multiplying the input matrix with the matrix P with particular properties can be computationally challenging; see further discussions in [3, 51, 118].

The applications implied by the lemma include, for example, the above-mentioned random projections method and its faster version called the fast Johnson-Lindenstrauss transform (FJLT) [10], learning mixtures of Gaussians [117], and approximate nearest-neighbor search [287].

7.11 Conclusions

7.11.1 Summary and method of application

In this chapter we have presented a range of theory and algorithms that relate to the problem of dimension estimation and reduction. Our perspective has its origins in mathematical theory. It concerns primarily how mathematical definitions and concepts can be exploited by data scientists to explore the geometry and topology of a dataset. Algorithm development for understanding data is now well evolved and the resulting toolset is quite powerful. It is our thinking that clues for further development of these algorithms and additional inspiration for new algorithms can be found in the depths of mathematical theory. Hopefully this article will encourage mathematicians, from across the spectrum of applied to pure, to explore mathematics for the purposes of knowledge discovery in data.

Given all the techniques of this chapter, how does a data scientist know what to use to make sense of dimension estimation and data reduction? Given the diversity of applications a precise flowchart is unrealistic, however, it is possible to provide some general guidelines. The first thing to try is almost always Principal Component Analysis. This linear method provides both an estimate of dimension and a basis for the associated reduced subspace. While PCA captures maximum variance of the dataset, one may opt for secant projection methods to preserve more geometric structure. Secant-based methods prevent the collapsing of distinct points, which may be more attractive for some applications. Nonlinear methods for reducing the dimension of data are potentially of interest if the eigenvalue distribution of PCA suggests that a linear mapping to a low-dimensional space is not possible. This can happen, for example, if a topological circle explores all dimensions in the ambient space. A nonlinear reduction technique might be used to reduce such a dataset to three dimensions with the Whitney Reduction Network. Other methods provide a dimension estimate without an actual data reduction mapping. These include correlation dimension, fractal dimension and false nearest neighbors. Lastly, if similarities or differences are available rather than actual data points, methods such as MDS still apply. Other manifold learning techniques such as Laplacian eigenmaps are also potentially useful; see [305] for a unifying view of these methods.

7.11.2 Suggested exercises

In the spirit of the above blueprint for dimension estimation and data reduction, we encourage the reader to explore these methods further by applying them to real data. A variety of interesting datasets are hosted at the UCI Machine Learning Dataset Repository [136]. Download one of these datasets and estimate its dimension by applying one or more of the data reduction algorithms detailed in this chapter, using Python or MATLAB code as shown in the chapter's figures, or by looking up appropriate packages in your language of choice. Consider attacking these particular challenges:

1. How many dimensions does it require to retain 95% of the variance in your data using PCA?

2. Plot the length of the minimum projected secant as a function of dimension using the PCA basis.

3. How does this graph corroborate your PCA estimate?

4. Would the particular dataset you chose be better served by the properties of one of the other methods introduced in the chapter?

Or if you prefer a less data-driven exercise, try replicating the experiment shown in the Python code in Figures 7.19 and 7.18, but for a shape other than the torus.

Have fun!

Chapter 8

Machine Learning

Mahesh Agarwal
University of Michigan-Dearborn

Nathan Carter
Bentley University

David Oury
True Bearing Insights

8.1 Introduction

Machine learning is the discipline of programming computers to learn from experience (known data) in order to make predictions and find insights on new data. Applications of machine learning are distributed across several fields such as marketing, sales, healthcare, security, banking, retail, and dozens more. While there are also dozens of specific applications within those fields, a few examples include spam detection, product recommendation, fraud detection, image recognition, and text classification.

These two goals—predictions and insights—and the machine learning problems they solve can be broadly categorized into two classes, **unsupervised learning** and **supervised learning**. Chapter 5 gave detailed coverage of the most common type of unsupervised learning, clustering, so this chapter focuses on supervised learning. In order to introduce the elements of a common machine learning workflow, we start by reviewing a familiar example of supervised learning, linear regression. We then consider more advanced methods such as logistic regression, Bayesian classifiers, decision trees, and support vector machines, giving you several powerful new tools in your data science toolbelt.

8.1.1 Core concepts of supervised learning

In supervised machine learning, data takes the form of datasets in which one column is called the "target," "response," or "dependent" variable (and is referred to as y) and the remaining columns are called "feature," "predictor," or "independent" variables (and are referred to as $\mathbf{x}$). The target column contains the values to predict and the feature columns contain those values used as input to make these predictions. Below, the term "feature-target dataset" refers to datasets with an identified target column. The goal of supervised learning can be summarized as finding, from a feature-target dataset, an underlying function f such that $f(\mathbf{x}) \approx y$. This typically happens without using knowledge of the full dataset, which we'll explain further in Section 8.2.

In machine learning, instead of explicitly programming rules for prediction, the approach is to use data to **train** a model to discover a pattern in the data. Consequently, a single observation from our dataset, which we denote with $(\mathbf{x}^{(i)}, y^{(i)})$ for row i, is called a **training** example. A collection of such examples, $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, \ldots, n\}$ is a **training set**, and is used to learn the "best" function f such that

$$f(x_{\text{new}}) \approx y_{\text{new}},$$

when we encounter a previously unseen data point $(x_{\text{new}}, y_{\text{new}})$, i.e., it is not among the $(\mathbf{x}^{(i)}, y^{(i)})$.

We will distinguish carefully between the terms "model" and "fit model." When we say model, we are referring to a function that takes a feature-target dataset as input and produces a fit model as output, typically by seeking optimal values of the model parameters. This mapping (from dataset to fit model) is known as "training." A fit model is a function that takes as input a set of features and predicts the corresponding target value. Equivalently, we can see the fit model as taking an entire dataset with feature columns as input and adding to that dataset a full column of target values, by performing its prediction operation on each row.

Let's clarify the distinction between a model and a fit model with an example. Assume we have a dataset of house characteristics and the sale prices of those houses, and the sale price is the target. If we were to apply linear regression (as introduced in Section 4.3.1), then the model would be just that—linear regression. Linear regression is a technique that takes as input the known dataset of sale price and house characteristics and produce a fit model, which in the case of linear regression is a collection of coefficients (one for each feature column) and an intercept. In other words, before the coefficients are determined, we just have the general model of linear regression, but once they are determined, we have a fit model that has been trained on our data, and is a specific linear equation. The estimated sales prices (target) can be calculated by the fit model (one target value for each row) in the usual way, the intercept plus the product of each characteristic with its corresponding coefficient.

Typically, these estimates (and so the fit model that created them) are then scored to evaluate the fit model's quality. In the case of linear regression, its estimates are often scored using the root mean-squared error (RMSE). This process of creating the model, predicting with the model, and evaluating the model by scoring these predictions are essential components of the machine learning workflow we introduce in Section 8.3.

8.1.2 Types of supervised learning

Supervised learning models can be divided into two classes: those that produce fit models that make predictions for continuous numeric target variables, which is called **regression,** and those that produce fit models that make predictions for categorical target variables, which is called **classification.** The prediction of housing prices is an example of regression, because sales prices (the target) are numeric.

Determining whether emails are spam (or not spam) is an example of classification. In the context of classification problems, a dataset $\{(\mathbf{x}, y)\}$ is called **labeled** if every $\mathbf{x}$ is accompanied by a tag y which represents the group or class to which $\mathbf{x}$ belongs. Binary classification models are written to predict values of 0 or 1 (or, equivalently, 1 or -1). There are standard techniques that translate categorical variables into one or more binary numeric variables, as covered in Section 4.6.5.

Arguably two of the most basic requirements of machine learning for most models are that the target and feature columns are numeric and that these columns have no missing values. If values are missing from the original dataset, they need to be imputed before fitting most machine learning models; Sections 2.4.3.1 and 8.3.2.1 discuss the handling of missing values.

8.2 Training dataset and test dataset

The goal of supervised machine learning is to make accurate predictions on future (unseen) data. To do so, we create a model using known data and score this model using known data, with the intent that this score (on known data) is a good estimate of the score of predictions on unseen data. This section introduces two constraints inherent in the above stated goal and describes how they determine a **machine learning workflow**.

8.2.1 Constraints

In order that a model's score on known data is a good estimate of its score on unknown data, the data on which the model is scored should be unknown to the process that creates the model. This leads to the following two constraints.

1. A model should be applied to a **training dataset**, and the resulting fit model should be scored by its predictions on a **test dataset** that is separate from the training dataset.[1]

2. No information from the test dataset should be used to modify the training dataset, nor to modify any aspect of the model fitting process.

Failing to satisfy the second constraint is often referred to as **data leakage** [427]. Information from the test dataset includes summaries of columns (variables) from the test dataset, predictions made on the test dataset, scores of such predictions, and literally any other data computed based on the test dataset.

These constraints are used to preserve the use of the test dataset as a valid and believable proxy for unseen data and they determine the components of the machine learning workflow described below. It is a common error to use the training dataset (and the predictions made on it) as a proxy for unseen data, but the training dataset is "known" to the model, because the model was created from it. This is intimately related to the concept of overfitting a model, introduced in Chapter 4 and revisited in several other chapters of this text. If the data scientist trains the data too specifically to the quirks of the training set, that fit model is said to be "overfit" to the training data, and thus is likely to produce poor predictions on test (or other unseen) data.

The use of training and test datasets is standard practice. Many machine learning software packages have a facility to create training and test dataset pairs.

Even so, in practice, these constraints are sometimes disregarded, but doing so diminishes the expectation that model performance on the test dataset is an accurate estimate of model performance on unseen data. In sensitive domains (e.g., health care), reporting to others a model's score on a test dataset, without mentioning data leakage that has occurred, can be unethical, because it misrepresents the capabilities of the model.

In particular, note the temptation to iteratively modify a model based on its performance on the test set, until it performs well on future runs on that test set. This is a type of data leakage in which the data scientist's own activity is the medium through which the leakage occurs. This is likely to produce results whose accuracy is not representative of model accuracy on (actually) unseen data. Marco Lopez de Prado analyzed this process in financial machine learning in [121].

At face value, the data leakage constraint implies that the test dataset should be used only once to create and score predictions. The cross-validation technique mentioned in earlier chapters and described further in Section 8.2.2 allows the training dataset to be used to iteratively improve a model, by creating multiple "train-validate" dataset pairs to evaluate multiple models without

[1] And when we introduce validation datasets in Section 8.2.2, this constraint applies there as well. Models should be trained on the training data and evaluated on the validation data.

ever looking at the test data. Thus the test dataset is not used until the best model has been chosen after iterating over the train-validation dataset pairs. See Section 8.3.6 for details on integrating cross-validation into a machine learning workflow.

8.2.2 Methods for data separation

The previous section emphasized the importance of having separate training and test datasets, as well as the importance of using the test set only once, to evaluate the final model. This leaves us with the question of how to iteratively improve a model using only the training data. This section introduces two answers to this question.

Before we do so, note that the method we've described so far for training-test separation is called **holdout**. In holdout, the original labeled data is divided into two disjoint sets, typically about 70% in the training set and the other 30% in the test set. The model is trained on the training set and evaluated on the test set.

The limitation mentioned above is not the only concern with holdout. Because any data withheld for testing is not available for training, a model trained on only a part of the data is unlikely to be as good as a model trained on the full dataset. A model trained using the holdout method is highly dependent on the composition of the training set. This problem can be aggravated if the original dataset was small to begin with, or if the classes in the dataset are imbalanced. To see the impact of imbalanced data, consider a case such as fraud detection or rare disease discovery; imagine 10 cases of fraud out of 10,000 credit card transactions. It is possible that a test set with 3000 cases contains no fraudulent transactions, thus leading to a poor model with poor estimated performance. This issue can be partially addressed by ensuring that the training and test sets have the same distribution of response classes. For example, if analyzing survival in the Titanic dataset (Section 8.6), it is advisable to split the set so that the proportion of survivors is the same in both the training and test sets.

If using the holdout method, one way to get a better estimate of the **generalization error**, which is the expected error on previously unseen data, is to repeat the holdout method with different choices of training sets. Then the mean and variance of the errors across these repetitions can be computed and used to compute confidence intervals on the model's final score. This process of repeating the holdout approach multiple times is also called **random subsampling**. While this method does improve the performance estimate, it is still possible for some records to be used more often in training than others. And it doesn't address the original concern we raised in the previous section, of how to iteratively improve a model without ever looking at the test data.

To address that concern, we introduce **cross-validation**. In this very common approach, the labeled dataset of size n is divided into k (usually $k = 5$ or 10) disjoint subsets of approximately equal size $S_1, \ldots, S_k$, as in Figure 8.1.

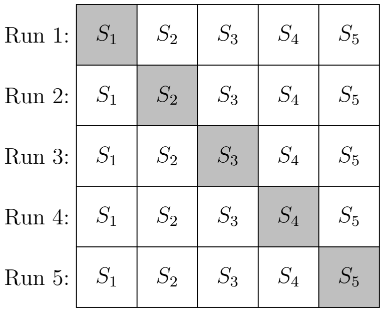

Run 1: S_1 S_2 S_3 S_4 S_5
Run 2: S_1 S_2 S_3 S_4 S_5
Run 3: S_1 S_2 S_3 S_4 S_5
Run 4: S_1 S_2 S_3 S_4 S_5
Run 5: S_1 S_2 S_3 S_4 S_5

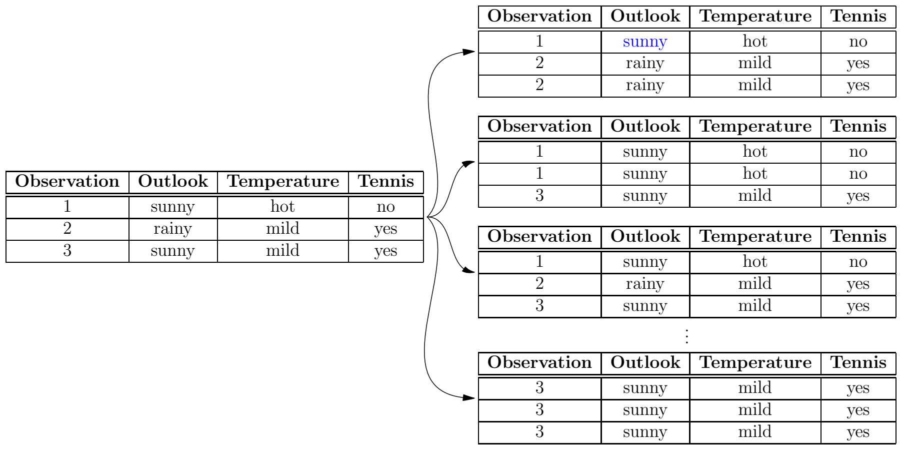
FIGURE 8.1: k-fold validation for $k = 5$. Each row represents one run of training and testing, with the shaded set S_i in that row set aside for testing and the remaining subsets united to form the training set.

The model is then fit to data k different times. During each run, one of these k subsets is used as a test set and the remaining $k - 1$ are used as a training set. This way, each record is used the same number of times for training and exactly once for testing.

If k is large, the training set is almost as large as the full labeled dataset, hence the error estimates are similar to those obtained when the full dataset is used for training. Error estimates from the k runs are averaged out to get an estimate of the generalized error. In the special case $k = n$, this method is called the **leave-one-out** approach. Cross-validation has the disadvantage of being computationally expensive, since the process has to be repeated k times.

Another common approach is **bootstrap**, in which a training dataset of the same size as the full labeled dataset is created, but by sampling *with* replacement from the full labeled dataset, as in the example in Figure 8.2. Hence a labeled record can appear more than once in the training set. If the labeled dataset is large, then an observation has about a 63% chance of being in the training set.[2] The records that are not included in the training set become a part of the test set. The model is trained on the training set and the performance is recorded on the test set. This sampling process can be repeated multiple times to generate multiple **bootstraps**. The accuracies from each of the resulting test sets can be averaged to get an estimate of the generalized error.

[2]Each of the n records has probability $1 - (1 - \frac{1}{n})^n$ of getting selected; as $n \to \infty$, this becomes $1 - \frac{1}{e} \approx 0.63$.



Observation	Outlook	Temperature	Tennis
1	sunny	hot	no
2	rainy	mild	yes
3	sunny	mild	yes

Observation	Outlook	Temperature	Tennis
1	sunny	hot	no
2	rainy	mild	yes
2	rainy	mild	yes

Observation	Outlook	Temperature	Tennis
1	sunny	hot	no
1	sunny	hot	no
3	sunny	mild	yes

Observation	Outlook	Temperature	Tennis
1	sunny	hot	no
2	rainy	mild	yes
3	sunny	mild	yes

⋮

Observation	Outlook	Temperature	Tennis
3	sunny	mild	yes
3	sunny	mild	yes
3	sunny	mild	yes

FIGURE 8.2: Four different bootstrap samples from the dataset on the left.

8.3 Machine learning workflow

This section introduces a machine learning workflow, which details the basic steps in a machine learning project. As Chapter 1 stated, there are many ways to describe the organization of a data science project, but the chronological workflow we suggest here pays particular attention to the constraints in Section 8.2.1. We describe the steps of the workflow first in general terms and then, in separate sections, dive into each step and connect it to a running example, using a dataset from the Boston housing market and linear regression as the model.

The "machine learning workflow" takes a raw dataset as input and produces a fit model as output optimized to produce good predictions on future unseen data. We list the steps here and illustrate them in Figure 8.3. As you read the steps below, feel free to compare them to the figure.

1. Obtain the initial raw dataset, which may come from multiple sources and take some wrangling to form into a single table.

2. Preprocessing: Create a feature-target dataset from the initial dataset, which entails data cleaning, handling missing values, feature engineering, and selecting a target.

3. Split the feature-target dataset into a training set and a test set.

   To keep our first example simple, we will use holdout, but will later return to cross-validation in Section 8.3.6.

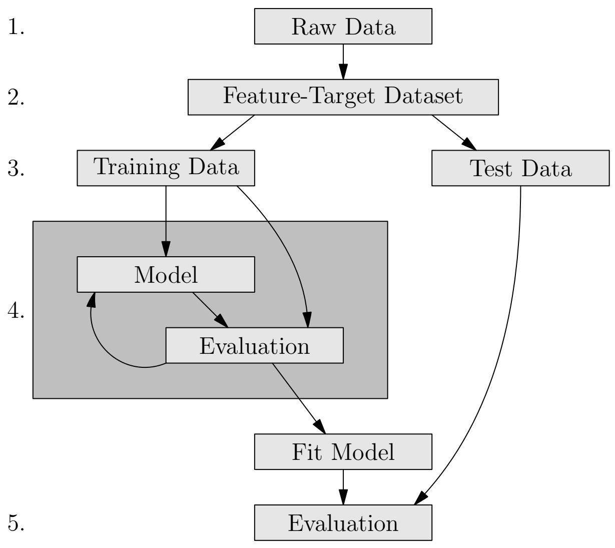


FIGURE 8.3: A flowchart illustrating the workflow introduced in Section 8.3. The numbers on the left of the figure correspond to the steps listed in the text.

4. Modeling: Create a model from the training dataset.

   Again, we will keep our first example of this workflow simple, and form just one model without seeking to iteratively improve it. But as Figure 8.3 shows, this step is typically an iterative process, leveraging tools like cross-validation or bootstrap.

5. Use the model to create predictions from the features in the test dataset and score those predictions by comparing them to the corresponding target values in the test dataset.

Most of the work involved in machine learning projects is contained in step 2 (preprocessing) and step 4 (modeling) and thus those steps make up the majority of this section. Step 1 is highly dependent on the sources and formats of the dataset and often involves writing code to address the needs of the specific datasets, including those sources and formats. Step 3 involves one of the techniques from Section 8.2.2, but we will begin with holdout in our first example below, randomly designating a portion (typically 70%) of the feature-target dataset to the training dataset and the remaining portion to the test dataset. Step 5 is also standard. Because our example will involve a linear regression model, we can use the R^2 (or adjusted R^2) metric from Section 4.6.3; if we were doing classification, we could use one of the scoring techniques we introduce later in Section 8.6.3.

Data scientists may spend as much as 80%–90% of their time on steps 1 and 2. Without good data preparation, it is impossible to build good models. Furthermore, the steps can be highly domain dependent, and an understanding of the context of the data can be very helpful in building good models.

We demonstrate the workflow using a running example from a dataset based on the U.S. Census Service data for houses in Boston, MA, originally published in 1978 [210]. The dataset has 1012 rows, each representing a 1970 U.S. Census tract in the Boston area, and 13 attributes per tract, some numeric and some categorical, described in Table 8.1. An additional attribute, median value (MEDV), is often used as the response variable and is suitable for a regression problem because it is numeric. These attributes are described in the table. The MEDV variable is censored, in that median values at or over \$50,000 are set to \$50,000.

TABLE 8.1: Description of variables in the Boston housing dataset.

Variable	**Description**
CRIM	Per capita crime rate by town
ZN	Proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	Proportion of nonretail business acres per town
CHAS	Charles River dummy variable (1 if the tract bounds the river, 0 otherwise)
NOX	Nitric oxide concentration (parts per 10 million)
RM	Average number of rooms per dwelling
AGE	Proportion of owner-occupied units built prior to 1940
DIS	Weighted distances to five Boston employment centers
RAD	Index of accessibility to radial highways
TAX	Full-value property tax rate per \$10,000
PTRATIO	Pupil-teacher ratio by town
B	$1000(B_k - 0.63)^2$, where B_k is the proportion, by town, of people described as black
LSTAT	Percentage of the population classified as lower status
MEDV	Median value of owner-occupied homes, in \$1000s

8.3.1 Step 1: obtaining the initial dataset

The first step of the workflow is obtaining the initial raw dataset. Because we have chosen to use a well-known dataset, this step is easy for our example. The dataset is easy to obtain freely on the Internet, and comes built into several popular statistical and machine learning software systems.[3] In other cases, though, this stage entails reading and merging multiple datasets that may be

[3]It appears in the R package `mlbench` [310] and is included in the Python library scikit-learn [385].

in different formats. This may involve a wide range of coding techniques, some of which were discussed in Chapter 2.

The initial dataset produced at this stage should be checked to ensure it meets expectations. This check may be informal and performed manually, which is called exploratory data analysis, or it may be an automated monitoring process used to check dataset quality against formal specifications. In both cases variable ranges and distributions are checked and missing values are identified.

Chapters 2 and 4 have both discussed exploratory data analysis (EDA) techniques, so we do not review them in detail here. But note that it is common to explore the entire dataset in this phase before the training and test sets have been separated. This is often important to check data quality, but it is important to avoid letting information from the (future) test dataset influence the modeler's strategy too much, or data leakage has occurred.

Some situations provide much more strict boundaries. For instance, in a data science competition (for example, using the Common Task Framework introduced in [132]), participants are only ever shown the training data, and the test data is reserved for evaluating winners. For a less contrived example, imagine deploying a model into a software environment where it will be automatically applied to new data; clearly, the data scientist does not get to see that new data when building the model.

The first goal of EDA is understanding the functional types (categorical, numerical) and the data types (string, integer, decimal, logical) of the variables in the dataset. Visualizing variables can provide insight into single variables and into relationships between multiple variables. Common summary statistics fall into this category, including the mean and standard deviation for continuous variables and finding counts and proportions for categorical variables. In addition to these univariate measurements, bivariate measures are common for finding correlation between variables, to identify redundant variables and multi-collinearity that could harm regression models. Two-way tables, χ^2 tests, and ANOVA are common techniques; the reader may refer to Chapter 4 for a review, and Section 4.6.3 for coverage of multicollinearity in particular.

For example, if a data scientist were exploring the variables in the Boston housing dataset, she might create a histogram of each variable. The result for the ZN variable is in Figure 8.4, showing that the distribution of this variable is highly left-skewed.

To check that the data seems reasonable, the data scientist might compute a summary of its values, like the one in Table 8.2. Because the variable is supposed to represent proportions, when we see that it has a minimum of zero and a maximum of 100, we learn two things. First, the values seem to be percentages (out of 100) rather than proportions (out of 1.0), which will obviously be important if we choose to do any computations based on this variable. Second, none of the values seem unreasonable (e.g., 110%). If any had been outside the sensible range, those rows may need to have been discarded.

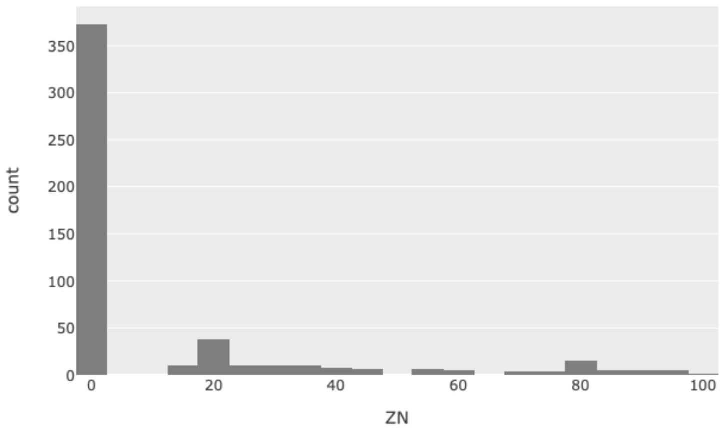


FIGURE 8.4: Histogram for the ZN variable in the Boston housing dataset.

TABLE 8.2: Numeric summary of the ZN variable in the Boston housing dataset.

	ZN
count	506.000000
mean	11.363636
std	23.322453
min	0.000000
25%	0.000000
50%	0.000000
75%	12.500000
max	100.000000

Plots are less reliable than numeric measures, however; for instance, it is not obvious from Figure 8.4 that the maximum value is 100. Not only is the bar near 100 very small and thus easy to miss visually, but even if we spot it, we know that a histogram clumps nearby data points together, and so the maximum might not be exactly 100.

During EDA, it is common to discover that some rows and columns have missing or outlier values. We discuss strategies for addressing these concerns as part of the preprocessing step, below.

8.3.2 Step 2: preprocessing

The preprocessing step in the workflow takes as input the initial dataset and produces a feature-target dataset using a series of data transformations.

This step has two goals. First, the resulting feature-target dataset should have no missing values. Second, some features may need to be created or modified, a process known as **feature engineering**. All data transformations in this step must be **non-aggregating**, a concept we explain and justify in the following section.

8.3.2.1 Missing values and outliers

Recall from Section 2.4.3.1 that many datasets have entries that are either unspecified (missing values) or are outliers (present but extreme values). For instance, if a disproportionate number of rows specify 99 for the age variable, that could indicate that 99 has been used as a flag indicating an unknown age. Outliers or missing values can be due to the nature of the domain from which the data is derived or due to issues with data collection. As mentioned in the previous section, these properties of the data are typically discovered in the EDA phase. Then there are several options: remove observations with missing or outlier values, remove features with too many missing values, replace missing or outlier values with valid values, or use a model that is robust to missing or outlier values.

The first two options require a threshold of the number or proportion of missing or outlier values. For example, in the Titanic dataset introduced in Table 8.4 of Section 8.6, the column "body" has too many missing values; a data scientist might consider discarding it. Removing rows or columns that exceed the threshold should not be done lightly as there may be "signal" in the missing or outlier values.

The third option, replacing missing or outlier values, is called **data imputation** and should be done carefully, so as not to adversely affect the results of the model. Missing data can be imputed using domain expertise or leveraging statistical tools. Two of the most common imputation techniques are to replace missing values with a specified value based on an understanding of the domain or replace them with the mean, median, or mode of the variable. Another approach is to replace a missing value with a random number within one standard deviation of the mean value of the variable. A more involved form of imputation is to create a model that predicts the replacement value based on other features of the dataset.

If the data scientist chooses to do data imputation, it is important to consider subtle means of data leakage. Consider a medical study, where some patients' blood pressure data are missing, and we choose to impute those missing values with the mean blood pressure numbers across all patients in the dataset. If this action is taken during Step 1 or 2 of the workflow, before the training and test sets have been separated, then the test set rows have impacted this mean value, which is likely to be imputed into some rows in the training data, a subtle form of leakage.

Thus any imputation that **aggregates** data from an entire column should be avoided in this preprocessing step, and saved until Step 4 of the workflow, once the training and test sets have been separated. However, imputation that uses a static value (for example, filling in zeros in a sales column for weeks when we have no record of sales) is a **non-aggregating** data transformation, raises no such concerns, and can be performed here in Step 2. Thus these terms are important because of the second constraint from Section 8.2.1.

Once we have omitted any columns that contained too many missing values or outliers (assuming we elected to do so), we can then consider whether any rows should be omitted for a similar reason. Notice that order is important; considering rows before columns could result in the removal of an excessive number of rows in the dataset. In both cases the threshold is determined from requirements of the domain, prediction model, and application.

One of the reasons the Boston housing dataset is so popular is because it is very clean. It does not contain any missing values that require imputation, so we will not need to apply the techniques of this section to our example dataset.

8.3.2.2 Feature engineering

Feature engineering is the process of designing features that are most useful to the modeling process, that is, they help create fit models that make good predictions. There are standard techniques for creating additional features to add to a dataset, and we describe a few below. They fall into two broad categories: using requirements of the model and using our understanding of the problem domain. The examples below include some of each type.

When using the linear regression model, we are assuming linear relationships between the dataset's features and its target. But we may know or suspect that some features have a nonlinear relationship with the target, and thus it might be appropriate to add a new feature computed by transforming one or more existing ones. This includes what Section 4.6.3 calls "interaction terms," that is, the product of two features. These are sometimes referred to as **polynomial features** for obvious reasons, and are motivated both by the constraints of a linear model and our knowledge of the domain.

Because most machine learning models assume features are numeric, categorical features must be converted to one or more so-called "binary" or "dummy" features, which are numeric. Methods for doing so were covered in Section 4.6.5, and are another type of feature engineering, this one motivated exclusively by the requirements of the model.

Dates and times typically need to be converted to another form to be useful in modeling. Although dates are often composed of numeric components (e.g., the three numbers in 7/4/1776) this is not often a useful form for computation. For some domains, just the month value lifted from the date may be useful for prediction, possibly treated categorically. For other applications, a more useful value may be the day of the week (possibly represented as 0–6, indicating Sunday through Saturday). For other applications, the elapsed time from some fixed starting point until the date in question, as a number of days or months, may be more useful, or the difference in time between two date features. All of these common date manipulations are examples of feature engineering, all motivated by the semantics of the data and its relationship to what we know about the domain.

Domain knowledge may also lead us to do feature engineering in non-mathematical ways. For example, in the Titanic dataset we will cover in Section 8.6, if we're trying to model a passenger's likelihood of survival, we might be able to do some feature engineering from the column that includes the passenger's name. The title recorded in the name can inform us about the passenger's rank in society; from "Astor, Col. John Jacob," we can conclude that he has the title "Col." This can be meaningful information that lets us ask new questions of the data, such as whether a person of higher rank has influence that impacts whether he or she gets on a life boat. This type of feature engineering, that is, based on text, can often be done with regular expressions [171], but in complex cases may require manual data processing.

Finally, all the techniques of dimensionality reduction we learned in Chapter 7 are methods of feature engineering designed to reduce a potentially intractable number of dimensions down to something more manageable, or to reveal to us the simple structure hidden in complex data. The most common of these is principal components analysis (PCA, introduced in Section 7.3). It creates features that are linear combinations of the original features, but are linearly independent of one another. These new features are sorted from highest to lowest variance, so that we might choose the subset with highest variance.

In the Boston housing dataset, we consider the variables listed in Table 8.1. The CHAS variable was already an indicator variable in the dataset when we received it. If MEDV were one of our features, we might consider the product MEDV $\cdot$ TAX as a useful new feature (after converting units), because it indicates the median amount of annual property tax paid for a residence of the tract; but since MEDV is our target, we cannot use it to form new features. If there had been hundreds of features, we might have needed PCA, but as is, the dataset probably does not require feature engineering.

8.3.3 Step 3: creating training and test datasets

Recall the importance of this step, first described in constraint 1 in Section 8.2.1, and that typically 70% of the records are placed in the training dataset and the remaining 30% are placed in the test dataset. In our example, the Boston housing dataset contains 506 observations, and we will use an 80%/20% split, so we will assign 404 to the training dataset and 102 to the test dataset.

For a numeric target variable, assuming it is not sequential (e.g., a stock price at regular time intervals), the observations are usually assigned randomly to the training and test datasets. For a categorical target variable, it is desirable to ensure that the class distribution of the target is similar in the training and test datasets. Most software packages for machine learning can automate the test-train set creation and have many options to ensure requirements like these are satisfied; we will see an example using scikit-learn in Section 8.4.

8.3.4 Step 4: model creation

By this point, the data scientist should know what model they aim to use on the data. This choice is informed by the data exploration and manipulation in steps 1 and 2, and the choice of model may also have informed some of the feature engineering work, as described above.

Recall the distinction between a model and a fit model. The data scientist chooses the model, and the computing environment trains that model on the training dataset, producing coefficients (in the case of linear regression, or model parameters in general) that give us a fit model, ready to be applied to new data points. That training process happens in this step.

In order to keep our running example simple, so that we might focus on the workflow, we will use a linear model to predict MEDV from the other features. While the model selection and evaluation process is typically iterative, using training and validation sets as described in Section 8.2.2, we will omit those details in this walk through the workflow.

This is perhaps the central step in the machine learning workflow, so we reiterate a central concept. Minimizing error on the training set does not, in general, guarantee good performance on unseen data. This is the danger called "overfitting" that was introduced in Chapter 4 and has been revisited in many chapters since then. This danger is more likely if the hypotheses of the model are too complex or specific. Such complexity must be considered carefully against the size of the training set; smaller sets are more prone to overfitting. These guidelines can help us select a model that is more likely to perform well on unseen data.

In Section 8.3.2 we forbade aggregating transformations, to prevent data leakage. Now that training and test datasets have been separated, if we need to apply any aggregating transformation (such as imputing missing values with a column mean), we can do so now, before training the model. Two classes of data transformations often take place at this point, one called scaling and normalization, the other called feature selection. We focus on those two activities in the following sections.

In models with hyperparameters, this model creation step would also include tuning those hyperparameters. (For a review of hyperparameters, see their introduction in Section 6.2.4.) But the simple model we've chosen for our Boston housing dataset does not have any hyperparameters.

8.3.4.1 Scaling and normalization

Linear regression typically does not require scaling the feature variables since the coefficients can automatically adapt to the scale of the features. But this section is crucial in other models, even though we will not need to use it for our Boston housing model.[4]

Some model training algorithms (such as penalized regression, mentioned in Chapters 4 and 6) compare features, which can be meaningless if the features in question have different scales. In general, for two different measurements (say, distance traveled and number of passengers), there is no reason to suspect the data have even remotely the same range of values. In practice, the term **normalization** has multiple meanings, but we use it in its most general sense, the processes for resolving these differences.

There are two common techniques to create features with common ranges. The first is **min-max scaling**, where each value $x_j^{(i)}$ of feature j is replaced with

$$\frac{x_j^{(i)} - \min_i x_j^{(i)}}{\max_i x_j^{(i)} - \min_i x_j^{(i)}},$$

where i ranges over all observations. Obviously, this creates values in the interval $[0,1]$, and both endpoints of the interval are achieved. The second technique is **standard scaling**, where each value $x_j^{(i)}$ of feature j is replaced with $\frac{x_j^{(i)} - \mu}{\sigma}$, where μ and σ are the mean and standard deviation of that feature and i ranges over all observations.

While scaling may seem like a type of preprocessing, recall that it cannot be done in step 2 without the danger of data leakage, and must therefore be done now, before training a model. This ensures that the aggregate data transformations involved operate only on the training dataset. But whatever scaling transformations are created for and applied to the training data must be applied unchanged to the test data as well, or the test data will not be sensible inputs to the fit model later.

8.3.4.2 Feature selection

It is not always best to use all available features when training a model. This may be because we prefer a simpler model for reasons of elegance or interpretability (applying Occam's razor, as in Section 4.3.2) or because we want to avoid overfitting.

The term **feature selection** is ambiguous. Many people use it to refer to the decisions made by the data scientist to select those features that seem to make the most sense for inclusion in the model. Others use it to refer to automated algorithms that can select a subset of the features of a dataset. In both cases, the goal is creating a better model that produces better predictions. It is this second meaning that we focus on here; other chapters have focused on mathematical modeling in detail, but machine learning is a frequently automated process, so we focus exclusively on automated methods.

[4]One does need to take care with the scale of features in linear regression if using penalized regression (ridge or lasso) or if using an optimization technique like gradient descent, which can be sensitive to significant differences in scale across features, as discussed in Section 9.3.2.

There are several general methods that distinguish currently available feature selection techniques. Filter methods select features based solely on their statistical properties. For example, features might be selected based on their variance or on their correlation with the target. Such techniques can be applied before training the model.

Embedded methods use models that, as part of the model training process, assign values to each feature that indicate their importance to the fit model. Regularization models such as random forest (Section 8.10.2), lasso (Section 8.5.4), and elastic net [400] fall into this category.

Wrapper methods evaluate subsets of features by automatically repeating the process of training the model and scoring the fit model's predictions, one repetition for each subset that needs to be evaluated. The feature set with the best score is chosen. This method is more computationally intensive than the others, but finds feature sets that directly contribute to better models and predictions. The caveat is that this method can lead to overfitting [323].

Let us return to our running example, fitting a linear model to the Boston housing dataset. There are two well-known algorithms for feature selection in such a situation. Both require a way to measure the goodness of fit for a model, such as R^2, adjusted R^2, or root mean-squared error (RMSE), which is defined as

$$\text{RMSE} = \sqrt{\frac{\sum_i (y^{(i)} - \hat{y}^{(i)})^2}{n}}. \tag{8.1}$$

This is a measure of model error because $\hat{y}^{(i)}$ represents the value predicted for observation $(\mathbf{x}^{(i)}, y^{(i)})$, in which $y^{(i)}$ is the actual value. The RMSE is in the units of y, and hence its scale depends on the meaning of y.

By contrast, the R^2 metric introduced in Section 4.6.3, which we will discuss further in Section 8.5.3, is unitless. It measures the proportion of the variability in y explained by $\mathbf{x}$, and thus takes values between 0 and 1. In two models with the same number of features, the one with a higher R^2 value would generally be considered better. As we add covariates, R^2 monotonically increases, but one can guard against overly complex models by using the adjusted R^2 instead.

Equipped with such metrics, we can return to the two algorithms for feature selection in a regression model. One algorithm begins with an empty model and adds features, one at time; the other begins with a model using all features and removes them, one at a time. These two methods are called **forward selection** and **backward selection**, respectively.

In Table 8.3, we see the output of the forward selection process based on RMSE. Each asterisk indicates the inclusion of the indicated feature in the model. We can see that initially there is significant gain in both R^2 and reduction in RMSE but as the number features increase, the returns do not keep up. If we were doing feature selection manually, fit model #4 in the table may be a good choice. An algorithmic process would use some objective criterion to choose the cutoff. When we revisit cross-validation in Section 8.3.6,

TABLE 8.3: Forward feature selection on Boston housing training dataset.

#	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	BLACK	LSTAT	R^2	RMSE
1													*	0.55	6.16
2						*							*	0.66	5.34
3						*					*		*	0.70	5.04
4						*		*			*		*	0.71	4.95
5					*	*		*			*		*	0.73	4.80
6				*	*	*		*			*		*	0.74	4.72
7				*	*	*		*			*	*	*	0.74	4.68
8		*		*	*	*		*			*	*	*	0.74	4.64
9	*	*		*	*	*		*			*	*	*	0.75	4.62
10	*	*		*	*	*		*	*		*	*	*	0.75	4.59
11	*	*		*	*	*		*	*	*	*	*	*	0.76	4.53

we will see ways for making a more definitive choice. For now, let us choose fit model #4, whose equation is

$$\text{MEDV} \approx 20.67 + 4.6219\text{RM} - 6749\text{LSTAT} - 0.9146\text{PTRATIO} - 0.5148\text{DIS}.$$

8.3.5 Step 5: prediction and evaluation

We now wish to evaluate whether the fit model produced in the previous step, which we optimized using training data, will perform well on data it has not seen. We therefore now use that fit model to make predictions from features of the test dataset. This process, using the fit model from the previous step, is entirely mechanical. For instance, in the case of the equation given above, software simply needs to apply this simple linear function to each row of the test dataset, generating predictions for the MEDV variable in each row.

Our scoring criteria in the case of our Boston housing example will be the same metric we used during forward stepwise regression in the previous section, the RMSE. We can ask our computing environment to determine the RMSE by comparing the predictions to the actual values on the test dataset, and in this case we find RMSE = 5.746. This is higher than the error of RMSE = 4.95 that we got on the training data, in row 4 of Table 8.3.

The challenge with interpreting the above number comes from the fact that, as one can see from equation (8.1), the units of RMSE are those of y. It would be nice to have a measure that is invariant to the scale of the response variable. Recall from Section 4.6.3 the decomposition of the sum of squared errors, which is the numerator in equation 8.1. In that section, we defined the total sum of squares $SST = \sum(y^{(i)} - \bar{y})^2$, the sum of squared error $SSE = \sum(y^{(i)} - \hat{y}^{(i)})^2$, and the variability explained by the model

$SSM = \sum(\hat{y}^{(i)} - \bar{y})^2$, and related them as

$$SSE = SST - SSM.$$

In that section, we also defined the **coefficient of determination**,

$$R^2 = \frac{SST - SSE}{SST} = \frac{SSM}{SST} = \frac{\sum_i(\hat{y}^{(i)} - \bar{y})^2}{\sum_i(y^{(i)} - \bar{y})^2}. \tag{8.2}$$

From this equation, we see that R^2 measures the proportion of the variability in y explained by $\mathbf{x}$ and it takes values between 0 and 1.

Even though R^2 is independent of the scale of y, as we add more covariates, the R^2 value monotonically increases. This can encourage one to build more and more complex models, perhaps at the cost of interpretability and generalizability to new data (overfitting). This gives rise to the alternative measure called the adjusted R^2,

$$\text{adj}R^2 = 1 - \frac{n-1}{n-d-1}(1 - R^2), \tag{8.3}$$

which penalizes for the number of covariates d. If the number of samples n is very large compared to the number of covariates, adjR^2 is almost the same as R^2.

We use these measures in the following section when we consider how to extend our housing example with cross-validation.

8.3.6 Iterative model building

Until now, we've kept our Boston housing example simple, forming and evaluating just one model. But in a more complex scenario, a data scientist might follow a more complex and iterative workflow. If you begin with a simple model, but it doesn't prove adequate when measuring it, you might return to the drawing board and add new features, or change to a model that uses a technique other than regression, and so on, until finding some model that can discover appropriate patterns of the data. This brings us up against a limitation of the simple example workflow shown so far.

The constraints from Section 8.2.1 implied that, in step 5, predictions should be created and scored only once on the test data. This limitation gets in the way of the iterative model-building process just described. The simple workflow example we just gave will need to be extended if this limitation is to be removed without violating those constraints.

Thus we return to cross-validation, introduced in Section 8.2.2, which addresses this limitation and provides the ability to evaluate a model, and to create and score predictions made by the model, using only the training dataset. This provides the capability to iterate over multiple models and model configurations. (The bootstrap technique introduced in that same section can also be used to accomplish a similar goal.)

Cross-validation takes as input a model and a training dataset. It creates a predefined number k of training and validation dataset pairs from the original training dataset, in the manner described in Section 8.2.2 and illustrated in Figure 8.1. Then for each such pair, it fits the model to the training dataset of that pair and evaluates that model by scoring its predictions on the validation dataset of the same pair. The final result is a collection of such prediction scores. Thus during step 4, the data scientist gets feedback on how the chosen model generalizes to data on which it has not been fit, without yet involving the test data.

Consider again the application of linear regression to the Boston housing data, using the same four variables from earlier. If we want to evaluate how well this model generalizes, without yet seeing the test dataset, we could apply cross-validation. As an example, let's choose $k = 5$ and use R^2 as the metric for scoring. When we did so, we achieved the following five R^2 scores, one for each training-validation pair.

$$0.75, 0.69, 0.68, 0.78, 0.79$$

Because there is randomness inherent in the process of generating the cross-validation partition, if you repeat this experiment, you may get different scores. (Chapter 2 discussed best practices for ensuring reproducibility of automated processes that involve randomness.)

The mean and standard deviations of the cross-validation scores are used to compare models. The mean measures the performance of the model (not the individual fit models). The standard deviation measures its consistency or predictability; a high standard deviation indicates that the data scientist is taking on more risk if adopting that model, because it could perform far better or worse than the mean suggests.

The workflow modifications we introduced here impact only step 4, which is now an iterative process and includes the following.

1. using cross-validation (or bootstrap sampling) to create a collection of prediction scores (one for each validation dataset) for a model
2. determining whether these scores (and potentially their means and standard deviations) are satisfactory[5]
3. starting over if the scores are not satisfactory, which may include engineering new features or choosing a different model

When this iterative process completes, the data scientist has chosen a model for the problem at hand. To convert it to a fit model, the training process is then typically applied to the entire training dataset, not merely one of the subsets of it created by cross-validation. Then that final fit model is scored

[5]For instance, in the case of the five scores shown above, is the data scientist content with a model that explains about 70%–80% of the variability in the data?

once on the test set, as usual, in step 5. There is no restriction on the number of iterations in step 4 as long as step 5 is not run.

It is also possible to automate the process of hyperparameter tuning through cross-validation. We refer interested readers to grid search and randomized search methods, both of which are implemented in scikit-learn [385], the software toolkit we introduce next.

8.4 Implementing the ML workflow

We now make the general ideas of Section 8.3 more concrete by showing how to implement them in a popular machine learning software toolkit, scikit-learn [385], which we access through Python. Readers who prefer to use another language for their machine learning projects,[6] or who do not yet wish to dive into the code may wish skip this section and move on to further mathematical content in Section 8.5. Throughout this section we will see the utility of scikit-learn as we import several of its tools to make an otherwise lengthy computation the matter of a single function call.

Scikit-learn is a highly object-oriented framework, but we present here only the essential material for working with a few of its objects and classes. For more depth on scikit-learn, see [385], and for a more thorough treatment of object orientation in Python in general, see [134].

8.4.1 Using scikit-learn

We give an implementation of the machine learning workflow using scikit-learn on the Boston housing dataset. The Python code for the workflow appears in Listing 8.1, and we will discuss each of its steps below.

Step 1 of the workflow is to get the raw data into a table, which is easy in the case of the Boston housing data, because it is one of the built-in datasets in scikit-learn. The code for doing so appears at the top of Listing 8.1; the first two lines read the dataset and store it in a variable.

Data obtained from most real-world sources has not been cleaned and prepared for analysis, as the built-in datasets in packages like scikit-learn have. So in most cases, Step 1 takes much more effort, as discussed in Chapter 2. We keep this step simple here so that we can focus on the machine learning steps instead.

Step 2 stores the features and target in different variables, for ease of reading the code, and so that we can easily pass them later to functions that

[6] As of this writing, there is a Julia library for accessing scikit-learn [442] plus a Julia-specific machine learning framework [55], and R has its own packages for machine learning as well [52, 283].

Listing 8.1: Code for the machine learning workflow described in Section 8.4

```
### Step 1: Obtain data

from sklearn.datasets import load_boston
boston = load_boston()

### Step 2: Create a feature-target dataset

import pandas as pd
features = pd.DataFrame(
    data=boston.data,
    columns=boston.feature_names)
target = boston.target

### Step 3: Split into training and test datasets

from sklearn.model_selection import train_test_split
(X_training, X_test, y_training, y_test) = \
    train_test_split(features,target,train_size=0.8)

### Step 4: Create a model from the training dataset

from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LinearRegression
estimator = Pipeline([
    ('select', SelectKBest(k=5)),
    ('model' , LinearRegression())
])
fit_model = estimator.fit(X=X_training, y=y_training)

### Step 5: Score the model using the test set

predictions = fit_model.predict(X=X_test)
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test,predictions)**0.5)
```

expect them to be separate. The code for doing so uses the pandas [340] package for Python, which stores row-and-column data in an object called a "DataFrame," named after a similar structure from R. The pandas package is ubiquitous in Python data science work.

The Boston housing dataset is a grid of numbers, but as we saw in Chapter 3, it is helpful to name the columns when working with matrix data, and thus the dataset also provides its column names, which our code uses when constructing the DataFrame.

Step 3 separates the dataset into training data and test data. This is such a common action that there is built-in support for it in scikit-learn. We import from scikit-learn's model selection package a function called `train_test_split`.

In this case, we've chosen to use 80% of the data for training (hence the 0.8 in the code for Step 3 in Listing 8.1). The `train_test_split` function returns a quadruple of training features, test features, training targets, and test targets, in that order, and the Python code shown uses the language's support for assigning to multiple variables at once using tuples, as shown.

Step 4 creates a model from the data. As discussed in Sections 8.3.4 and 8.3.6, this step is typically iterative, but we have kept it simple here for our first example, and automated a single model construction.

We do, however, wish to illustrate several important aspects of scikit-learn at this point. First, the toolkit is built on the notion of **pipelines**, which are chains of composed functions that pass data along, one to the next. The code in Step 4 in Listing 8.1 shows only a small (two-step) pipeline, but pipelines can be arbitrarily long.

The first function in the pipeline is the `SelectKBest` function with $k = 5$. This function chooses the k features from the training dataset that are most useful for predicting the target, by default measuring this with the F-statistic used in analyses of variance. (Other measures can be specified, though we have used the default in this example.)

The second function in the pipeline then takes those five features and creates a linear model. A pipeline that ends by fitting a model is called an **estimator** in scikit-learn parlance, in contrast to a pipeline that merely modifies the data, which is called a **transformer**. Notice that the code fits the model to the training data only, storing the result in a variable called `fit_model`.

We see once again why scikit-learn is popular; these feature selection and model fitting functions are built in, and the data scientist can load them with a single `import` command. No code needs to be written to define these mathematical or statistical concepts; this has already been done for us, and is thoroughly tested and documented [385].

While it may seem silly to use a pipeline to compose only two steps, the example can easily be extended for more complex models. For instance, SVMs (introduced in Section 8.8) assume that the features have been standardized. In such cases, we could add to the beginning of the pipeline the pair `('scale', StandardScaler())` to include this step. Each step in a pipeline can be used to transform one or more variables, compute new columns, select certain rows, impute missing values, fit models, make predictions, and more; the concept is quite flexible, as we'll discuss more in the next section.

Finally, Step 5 scores the model by applying it to the test dataset. This is the final section of the code in Listing 8.1. Once again, the necessary statistical concept (in this case MSE) has been defined in scikit-learn. Since we wish to report the RMSE, we simply raise the MSE to the 0.5 power before reporting it.

8.4.2 Transformer objects

Readers who wish to understand scikit-learn's internals a bit more will find an introduction in this section, which should help you if you plan to read the scikit-learn documentation [385]. Readers who wish to focus on the mathematics instead my skip this section.

Scikit-learn is designed within the object-oriented framework embraced by the Python language. In that paradigm, various concepts are implemented as **objects** in the language; in our case, those concepts include data transformations, models, etc. In object-oriented parlance, a **method** is a function that is associated closely with a specific object on which it operates. Thus for example, a model will have a method for fitting it to data and another method for using the result to predict new target values; these methods are called `fit` and `predict` in scikit-learn.

Like models, data transformations must also be fit to data before they can be used. For instance, if we wish to transform a feature by imputing the mean value to all missing entries, we first need to customize that transformation to our needs by finding that mean value. Thus data transformations in scikit-learn implement not only a `transform` method, as we might expect, but also a `fit` method, which must be run first. For both models and transformations, the `fit` method "learns" characteristics of the dataset and stores them as attributes of the (transformation or model) object. This corresponds roughly to what we've called training when we've been speaking mathematically.

For example, a `LinearRegression` object creates a linear model, and its `fit` method does what we think of mathematically when fitting a linear model to data; it computes the β_i coefficients and remembers them (as attributes of the object in this case). Similarly, a `SimpleImputer` object, which imputes missing values with the mean of the (non-missing) values for the feature, has a `fit` method that learns the mean value of each column to which it is applied and remembers that mean for later use. Its `transform` method then uses that mean to replace each missing value in any columns to which it is applied.

A scikit-learn pipeline calls the `fit`, `transform`, and `predict` methods for us in the correct sequence for all steps in the pipeline. In Listing 8.1, at the end of Step 4, we call `fit` on the estimator pipeline to create a fit model, and then use that model's `predict` method in Step 5 as we score the model. Each of these called the appropriate `fit`, `transform`, or `predict` methods in each step of the pipeline, and transmitted data among those steps appropriately.

8.5 Gradient descent

At this point in the chapter, we transition from big picture ideas to specific methods. Sections 8.2 through 8.4 covered topics that apply to machine learning projects in general, and how one should go about them. Now we introduce new models and techniques that a data scientist can include at various points in that workflow.

In this section, we discuss the general tool of **gradient descent**, which plays a major role in neural networks (Chapter 9), and we will use it to explain extensions to linear regression. (Readers who recall initial coverage of gradient descent in Section 6.2.3 may find portions of the following to be a review.) Subsequent sections introduce new types of models that one might call the bread and butter of machine learning, including logistic regression, the naïve Bayes classifier, support vector machines, decision trees, and ensemble methods.

8.5.1 Loss functions

As outlined in the introduction, the goal of machine learning is to learn from data. We need a metric to quantify the performance of this learning, and we often use a **loss function** $L(\hat{y}, y)$, which represents the loss from predicting $\hat{y}$ when the correct value (often called the **ground truth**) is y. The total or mean loss from the full training data is used to determine the overall loss during training and that total is usually called the **cost function** or (ambiguously, also) the loss function.

In regression problems, we've seen the common **squared loss** function $L(\hat{y}, y) = (\hat{y} - y)^2$. But in a classification problem, we use the loss function to penalize incorrect predictions. A standard example is the **zero-one loss** function given by $L(\hat{y}, y) = \mathbf{1}_{\hat{y} \neq y}$, where $\mathbf{1}$ is the indicator function, shorthand for

$$\mathbf{1}_{\hat{y} \neq y} = \begin{cases} 1 & \text{if } \hat{y} \neq y \\ 0 & \text{otherwise.} \end{cases}$$

In some settings, it makes more sense to use an asymmetric loss function. For instance, declaring a fraudulent transaction as correct is more damaging to a bank than the other way around.

While we wish to minimize the cost function during training, focusing only on minimizing the cost function can lead to overfitting, sometimes called "memorization." What we really want is a model that generalizes well to unseen data, just as students wanting to do well on an exam should focus on the concepts, rather than simply memorizing examples. In the context of machine learning, the cost function is sometimes modified to draw attention to the "concepts." The form of the cost function is also adapted whenever possible to yield a convex optimization problem.

8.5.2 A powerful optimization tool

Once we have a cost function associated with a machine learning problem, our next goal is to find the model parameters that minimize this cost. In some very nice (but rare) settings, this can be done analytically; we will see such an example in the next section. In most cases, this optimization is achieved using numerical techniques. We will describe one such powerful optimization technique, gradient descent.

Let $J(\theta)$ be a differentiable, convex function representing the cost as function of the parameters θ of our learning model. We wish to determine the θ that minimizes the cost $J(\theta)$. We start with an initial guess θ_0 for θ and repeatedly update that guess using

$$\theta_{i+1} = \theta_i - \alpha \nabla J(\theta_i), \tag{8.4}$$

until it converges (i.e., $|\theta_k - \theta_{k+1}| < \epsilon$, for some sufficiently small ϵ). Here α represents the **step size** or **learning rate**. Intuitively, θ is updated in the direction of the steepest descent, which moves θ towards lower values of $J(\theta)$. This is illustrated on the left of Figure 8.5.

Choosing the appropriate learning rate ensures that the iteration converges efficiently. If α is too small, it will take too long for $\{\theta_i\}$ to converge and if α is too large, each step may overshoot the minimum and $\{\theta_i\}$ may diverge. In practice, a sequence of alphas can be used, starting with larger values and moving to smaller ones. One suitable choice is $\alpha = \frac{1}{L}$, where L is the Lipschitz constant of the gradient, which satisfies

$$\|\nabla J(\theta) - \nabla J(\theta')\| \leq L \|\theta - \theta'\|, \quad \forall \theta, \theta' \tag{8.5}$$

for suitably nice functions J [21, 59]. In equation (8.4), for a single update to θ, the iteration has to compute $\nabla J(\theta)$, which requires looping over the full training set. Using the entire dataset in this way is called **batch gradient descent**, but in cases where the training set is very large, this loop can be time consuming. Alternatively, we can update θ based on a single training example at a time, and choose those examples in a random sequence. This approach is called **stochastic gradient descent** and is illustrated on the right of Figure 8.5. It works well in practice, once the algorithm has had multiple passes at the data.

Gradient descent can be used in some non-convex settings, but in that context it does not always converge to a global minimum; the solution may get "stuck" at a local minimum. There are many interesting variants on gradient

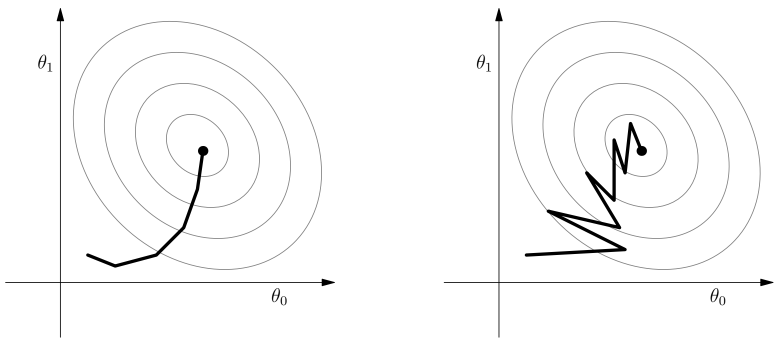


FIGURE 8.5: Gradient Descent vs Stochastic Gradient Descent . In each image, assume that the concentric ellipses are level curves and their center is a minimum.

descent that will be covered in detail in Chapter 9, because that chapter covers neural networks, which use gradient descent extensively in non-convex settings.

8.5.3 Application to regression

Linear regression is one of the simplest models that can be fit to data that presents a linear trend. It was covered in Section 4.3.1, but we briefly review it here to highlight applications of cost functions and gradient descent.

Recall that a representation of a linear function of $\mathbf{x}$ can be given by

$$f_\theta(\mathbf{x}) = \theta \cdot \mathbf{x} = \sum_{i=0}^{p} \theta_i \mathbf{x}_i,$$

where θ is a weight vector with same dimensions as $\mathbf{x}$. To simplify notation, we will assume that $\mathbf{x}_0 = 1$ so that θ_0 represents the intercept term, meaning both $\mathbf{x}$ and θ have length $p+1$. We will use the **ordinary least-squares** regression model, that is, given a training set we would like to pick θ to minimize the cost function,

$$J(\theta) = \sum_{i=1}^{n} (y^{(i)} - f_\theta(\mathbf{x}^{(i)}))^2, \tag{8.6}$$

the sum of squared errors, on the training set.

Let $\mathbf{X}$ be an $n \times (p+1)$ matrix of feature vectors, called the **design matrix**, and $\mathbf{y}$ a column vector of length $n \times 1$ of response values.

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Then we can express the goals of linear regression using the cost function framework we've used in the rest of the chapter, in matrix form as

$$J(\theta) = (\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}).$$

Assuming $\mathbf{X}$ is full column rank (i.e., the columns are linearly independent), $\mathbf{X}^T\mathbf{X}$ is positive definite, and setting $\nabla J(\theta) = 2(\mathbf{X}^T\mathbf{X}\theta - \mathbf{X}^T\mathbf{y})$ equal to zero yields the **normal equation**

$$\mathbf{X}^T\mathbf{X}\theta = \mathbf{X}^T\mathbf{y}.$$

Since $J(\theta)$ is a convex function, it has a unique minimum at

$$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{8.7}$$

So linear regression is one of the rare examples in which we have a closed form solution for the model parameters. Having estimated the regression parameter θ, we can predict the target value for any test vector $\mathbf{x}$ by evaluating $\mathbf{x} \cdot (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. But it is not always this easy.

Computing a matrix inverse can be computationally expensive or may not even be possible. Two columns of $\mathbf{X}$ may be perfectly correlated; for instance, when one column $\mathbf{X}_{\text{salary}}$ represents the salary of an employee and another $\mathbf{X}_{\text{retirement}}$ represents the retirement contributions made by the employer (perhaps $\mathbf{X}_{\text{retirement}} = 0.06 \cdot \mathbf{X}_{\text{salary}}$). In practice, if some columns are highly correlated, then it is advisable to remove the redundant columns. For instance, in the Boston housing data, Figure 8.6 shows that the variables RAD and TAX are highly correlated, with a correlation coefficient of 0.91. Keeping both the variables in the linear regression model can lead to **multi-collinearity**, which can cause the coefficient estimation to be unstable and make it difficult to separate the effect of one input from another.

If any of the above concerns arose in a linear regression model, we could apply (for example) stochastic gradient descent to estimate the model parameters θ by incorporating the linear regression loss function into the gradient descent update step.

$$\theta_{j+1} = \theta_j - \alpha\mathbf{x}^{(i)}(\theta_j \cdot \mathbf{x}^{(i)} - y^{(i)}) \tag{8.8}$$

8.5.4 Support for regularization

Earlier in this chapter, we mentioned some strategies for selecting a subset of predictors that are best related to the response variable. Readers may recall from Chapter 4 strategies for fitting a model using *all* p predictors yet performing very well on test sets, called **regularization** or **shrinkage**, because they shrink some of the estimated coefficients θ_i towards zero (when compared to their estimates from the least squares model). This shrinkage in turn reduces the variance of the model. There are two major types of regularization: **ridge**

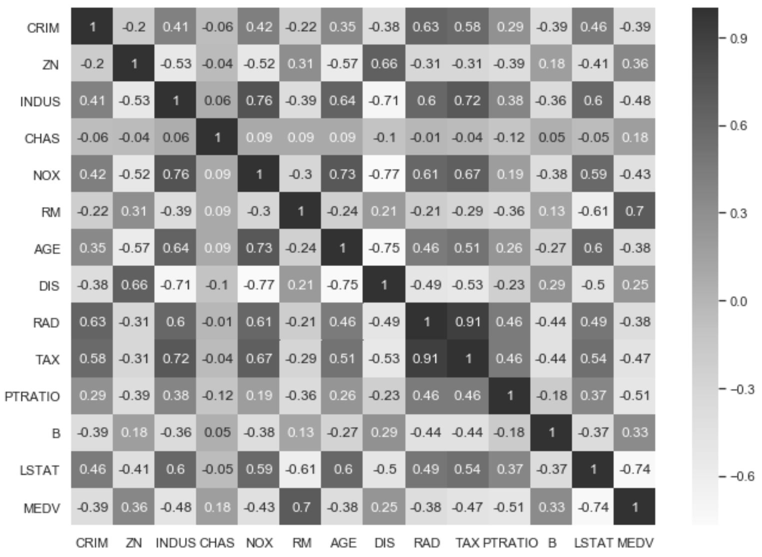


FIGURE 8.6: Correlation matrix for the Boston housing dataset variables, with each entry's background color darkened in proportion to the value of the entry.

regression, which shrinks all non-intercept coefficients but does not set any of them to be exactly zero, and **lasso**, which sets some coefficients *exactly* to zero. The latter can also be seen as another approach to model selection, since only some coefficients contribute to the model. Thus regularization is a way to prevent a model from overfitting, thus improving model performance on unseen data.

For convenience of the discussion below, we repeat the associated cost functions here; ridge uses the ℓ_2 norm to penalize each model parameter,

$$J_\lambda(\theta) = (\mathbf{X}\theta - y)^T(\mathbf{X}\theta - y) + \lambda \sum_{j=1}^{p} \theta_i^2,$$

through the tuning parameter λ. Lasso uses the ℓ_1 norm instead,

$$J_\lambda(\theta) = (\mathbf{X}\theta - \mathbf{y})^T(\mathbf{X}\theta - \mathbf{y}) + \lambda \sum_{j=1}^{p} |\theta_i|. \tag{8.9}$$

Ridge encourages more parameters to move towards zero as the data scientist increases the hyperparameter λ. In the same circumstance, lasso encourages more parameters to become exactly zero. This distinction is illustrated in Figure 8.7.

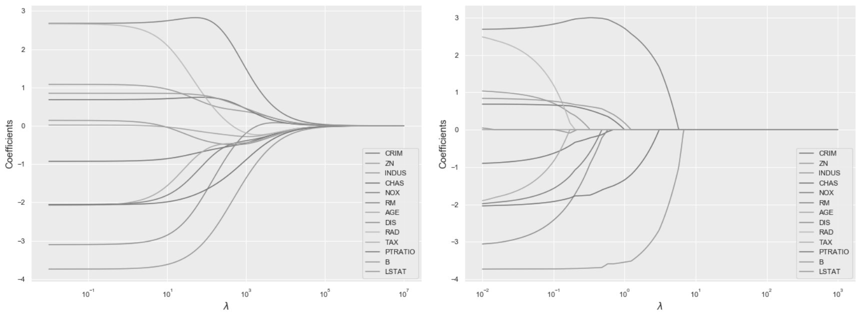


FIGURE 8.7: Ridge (left) and lasso (right) regression coefficients for a model fit to the Boston housing dataset. While this grayscale version does not permit the reader to distinguish the variables, the important pattern to observe is how they converge. In ridge regularization, the coefficients converge towards zero at varying rates, while in lasso regularization, each suddenly becomes zero at some value of λ.

To understand why lasso causes some coefficients to be exactly zero it will help to consider the contours of the error and the constraint functions. For simplicity, let us consider the two-dimensional case, illustrated in Figure 8.8. The constraint region for $|\theta_1|+|\theta_2| \leq t$ is a rhombus, while that of $\theta_1^2+\theta_2^2 \leq t$ is a circle. The contours of the least squares error function $(\mathbf{X}\theta-\mathbf{y})^T(\mathbf{X}\theta-\mathbf{y})$ can be represented by ellipses. Our goal is to minimize the sum of the magnitudes of the error function and the scaled shrinkage term. So in both approaches we seek to find the first point where the least-squares error function (the ellipses shown in gray) meets the constraint contour. In the lasso case, it is likely to be one of the corners of the rhombus centered at the origin, leading one coefficient to become zero entirely. In higher dimensions, similar results hold, except that more coefficients may become zero.

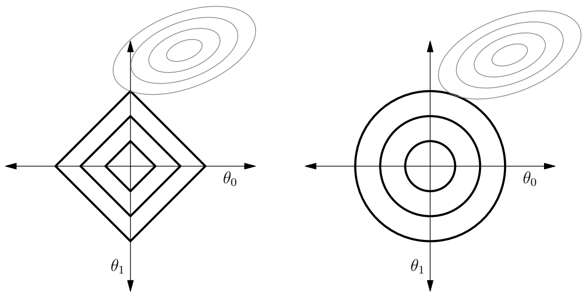


FIGURE 8.8: Contours of constraint and error functions in the lasso and ridge regression cases, as described in Section 8.5.4.

Section 4.7.2 pointed out that there is a closed-form solution only for ridge regression, not for lasso. We therefore find in regularization a new application of gradient descent; it can be used where no closed-form solution exists, because to express the penalties imposed by the regularization method we wish to employ, it requires modifications only to our cost function.

Yet even gradient descent has trouble with optimizing lasso regression, because the ℓ_1 norm in equation (8.9) is not differentiable. But a variation of the gradient descent method, called **sub-gradient descent**, discussed in Section 6.2.3, can be used in this case.

8.6 Logistic regression

On April 15, 1912, the Titanic, the largest passenger liner ever made at that time, collided with an iceberg during her maiden voyage. The ship sank, killing 1502 out of 2224 passengers and crew members. This tragedy led to enforcement of better safety conditions aboard ships thereafter.

The Titanic3 dataset[7] contains 1309 rows, each representing a real Titanic passenger, which constitute about 80% of the passengers on board [209]. The dataset contains 14 features, one of which indicates the survival status of the passenger. A description of the features appears in Table 8.4.

TABLE 8.4: Variables in the Titanic3 dataset.

Variable	**Description**
pclass	Type of passenger's ticket; values: "first," "second," "third"
survived	Survival status; values: 0 (did not survive), 1 (did survive)
name	Passenger's name
sex	Passenger's sex; values: "female," "male"
age	Age (in years) of passenger at the time of the voyage
sibsp	Number of siblings/spouses aboard
parch	Number of parents/children aboard
ticket	Ticket number
fare	Passenger fare
cabin	Cabin/room number
embarked	Port of embarkation; values: "C" (Cherbourg), "Q" (Queenstown), "S" (Southampton)
boat	Lifeboat identification number (if passenger survived)
body	Body identification number (if passenger did not survive)
home.dest	Passenger's home city

[7] So called because it improves upon earlier datasets named Titanic and Titanic2.

This dataset is included with many statistical software packages and is commonly used in statistics education. While the lack of sufficient lifeboats on the ship contributed to the high casualty rate, the dataset allows us to ask many other interesting questions about passengers' survival. For example, besides luck, were there other factors that predisposed some passengers to higher likelihood of surviving? Can we train a model that can predict a passenger's survival based on other features of that passenger?

Unlike a regression problem, where the goal is to predict a *continuous* response variable, in this problem we want to predict a *discrete* response variable (whether a passenger survived). Such a problem is called a classification problem. Logistic regression (first introduced in Section 4.7) is a binary classification technique that builds on linear regression through the logit function to deliver class probabilities for the response variable. Logistic regression is widely used in heath care and social sciences for its ease of interpretation, and we provide a brief review of it in the following section.

8.6.1 Logistic regression framework

We encode the two levels of the response variable as 0 and 1. Consider the **logistic function**

$$g(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}, \tag{8.10}$$

shown in Figure 8.9. It maps $(-\infty, \infty)$ to $(0, 1)$, hence its values can be interpreted as probabilities. We can define the formula for the logistic regression model as

$$h_\theta(\mathbf{x}) = g(\theta \cdot \mathbf{x}) = \frac{1}{1+e^{-\theta \cdot \mathbf{x}}}, \tag{8.11}$$

which embeds a standard linear model inside the logistic function. It is easy to check that

$$\log \frac{g(\theta \cdot \mathbf{x})}{1 - g(\theta \cdot \mathbf{x})} = \theta \cdot \mathbf{x}, \tag{8.12}$$

where the left-hand side is called the **log odds** or **logit**, because it contains the ratio of the probability of class 1 to the probability of class 0. If $\theta \cdot \mathbf{x} = \theta_0 + \theta_1 \mathbf{x}_1 + \cdots + \theta_p \mathbf{x}_p$, then we can interpret θ_i as the amount by which the log odds changes for one unit increase in $\mathbf{x}_i$, keeping everything else the same. So we can think of the logistic regression model as the usual linear regression model except that the response variable is a continuous variable representing the log odds of being classified into one of two chosen classes.

8.6.2 Parameter estimation for logistic regression

Since the logistic function is not linear, we do not have a closed form solution for θ and using the least squares estimate is also not convenient. We can instead use **maximum likelihood estimation** (MLE), introduced

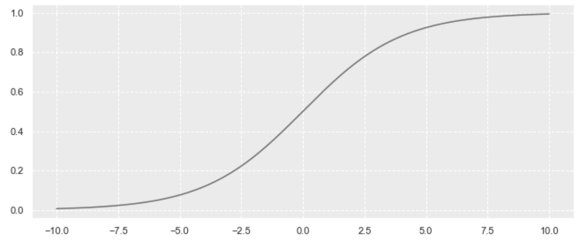


FIGURE 8.9: Logistic function $y = \frac{1}{1+e^{-x}}$, whose output (y) values can be used to represent probabilities.

in Section 4.3.5; we provide a brief review here. Using P to stand for the probability of an event, let

$$P(y = 1 \mid \mathbf{x}; \theta) = h_\theta(\mathbf{x})$$

and thus

$$P(y = 0 \mid \mathbf{x}; \theta) = 1 - h_\theta(\mathbf{x}).$$

Or if we use p to represent the probability mass function, we can write this more compactly as

$$p(y \mid \mathbf{x}; \theta) = h_\theta(\mathbf{x})^y (1 - h_\theta(\mathbf{x}))^{1-y}.$$

Assuming the n training examples are independently generated, we can express the likelihood of the parameters as

$$\prod_{i=1}^{n} h_\theta(\mathbf{x}^{(i)})^{y^{(i)}} (1 - h_\theta(\mathbf{x}^{(i)}))^{1-y^{(i)}}.$$

Maximizing this likelihood is equivalent to minimizing the negative log likelihood function instead. We can express this as a cost function

$$\begin{aligned} J(\theta) &= -\sum_{i=1}^{n} y^{(i)} \log(h_\theta(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(\mathbf{x}^{(i)})) \\ &= -\sum_{i=1}^{n} y^{(i)} \log(g(\theta \cdot \mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - g(\theta \cdot \mathbf{x}^{(i)})). \end{aligned}$$

Applying the gradient descent algorithm to $J(\theta)$, and using $g'(z) = g(z)(1 - g(z))$ when computing $\nabla J(\theta)$, we obtain the following form for applying stochastic gradient descent in this case.

$$\theta_{j+1} = \theta_j - \alpha \mathbf{x}^{(i)} (g(\theta_j \cdot \mathbf{x}^{(i)}) - y^{(i)}).$$

This equation can be expressed in terms of the design matrix $\mathbf{X}$ as

$$\theta_{j+1} = \theta_j - \alpha \mathbf{X}^T (g(\mathbf{X}\theta) - y), \tag{8.13}$$

where $g(\mathbf{X}\theta)$ means applying g componentwise to $\mathbf{X}\theta$.

Clearly the update rule is very similar to equation (8.8), which we obtained in the case of linear regression, except for the transformation of $\mathbf{X}\theta$ through the logistic function. Just as in the case of linear regression, one can regularize logistic regression using an ℓ_1 or ℓ_2 penalty.

To apply code like that shown in Listing 8.1 to a classification problem, the primary modification needed would be to change `LinearRegression` to `LogisticRegression`, which is another component built into scikit-learn.

The ideas in this section apply to classification into more than two response classes as well. For example, the performance of a student at the end of the year might be classified as outstanding, satisfactory, or unacceptable. For ease of notation let us call these classes 2, 1, and 0 respectively. We can carry out this classification by creating three different logistic regression classifiers, where

$$h_{\theta_i}(\mathbf{x}) = p(y = i \mid \mathbf{x}; \theta_i) \quad i = 0, 1, 2,$$

that is, $h_{\theta_i}(\mathbf{x})$ is the probability that $\mathbf{x}$ is in class i rather than either of the other two classes. (Here the other two classes are thought of as a single class which represents "not class i.") For a new input $\mathbf{x}$, we can predict its associated class as

$$\arg\max_{i=0,1,2} h_{\theta_i}(\mathbf{x}).$$

We have therefore now seen two models that can be used in Step 4 of the machine learning workflow from Section 8.3, linear and logistic regression. Both of these were reviews of techniques covered earlier in the text, and were used to provide a gentle introduction to the machine learning workflow and gradient descent. In Sections 8.7 and following, we introduce new models specific to machine learning that can be used in place of linear or logistic regression when they are more suited to the problem facing the data scientist.

But first, we must address the question that the previous section raises: When using a classification model like logistic regression in the machine learning workflow, how do we score the model in Step 5 of that workflow? Section 8.6.3 answers that question and discusses some related modifications.

8.6.3 Evaluating the performance of a classifier

Quantifying the performance of a classifier is necessary for understanding its effectiveness, comparing it with other models, selecting the best model from a list of models, tuning hyperparameters (like λ in regularization), and more. We address issues related to quantifying the performance of a classifier, and cover various metrics, including accuracy, F_1 score, and AUC.

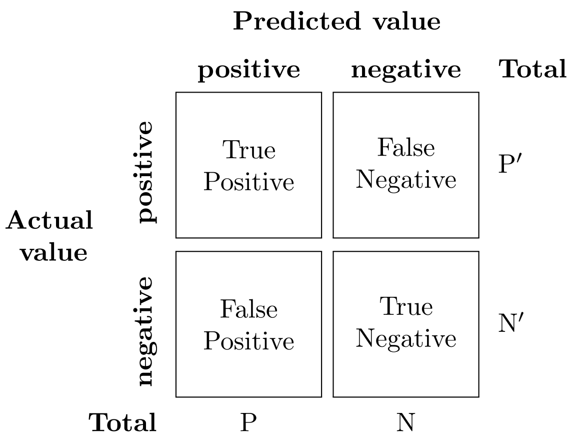


FIGURE 8.10: Confusion matrix for a binary classifier.

The performance of a binary classifier can be measured by the proportion of test records it classifies correctly. But consider the case of an HIV detection test, where such a measure is not sufficient. Suppose 10 in 1000 people in a community have HIV. A test can declare every person HIV negative and still achieve 99% accuracy. But clearly, such a test has failed its objective!

In general, the problem with measuring only accuracy is that in some cases, successful detection of one class is more important than the detection of the other class. A **confusion matrix** is a useful representation of the information needed for studying the performance of a binary classifier, and its general form appears in Figure 8.10.

Assuming the classes being predicted are "positive" and "negative," the matrix will contain four counts, one for each of the four categories shown in the figure. Let TP stand for the number of True Positive cases, that is, positive cases classified as positive, the top left entry in the table, and define FP, TN, and FN analogously.

The **accuracy** of a classifier is defined as

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall and **precision** are the preferred metrics for unbalanced datasets (where the number of cases in one class outweighs the numbers in the other class, like the community of 1000 with 10 cases of HIV). Precision measures the fraction of positive predictions that are correct.

$$\text{Precision} = \frac{TP}{TP + FP}$$

		Predicted value		
		positive	**negative**	**Total**
Actual value	**positive**	1	9	10
	negative	0	990	990
	Total	1	999	

FIGURE 8.11: Confusion matrix for an HIV classifier.

It answers the question that someone with a positive diagnosis cares about: How likely is it that my diagnosis is actually true? Recall measures the fraction of true positives that are detected.

$$\text{Recall} = \frac{TP}{TP + FN}$$

It answers the question that a community health official might ask: What proportion of the HIV cases in the community are we detecting?

It is possible to do well on some metrics at the cost of others; for example, an HIV test with the confusion matrix in Figure 8.11 has high precision and accuracy scores (1 and 0.991, respectively), but a poor recall score (0.1). Building a classification problem which maximizes both precision and recall is the key challenge of classification problems. A popular metric, the F_1 score, integrates the two metrics by taking the harmonic mean of precision and recall,

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

A high F_1 score ensures a reasonably high value of both precision and recall.

In a number of binary classification techniques (including logistic regression and the naïve Bayes classifier, which we will study soon) the classifier returns a (continuous) real value, representing the probability of the observation belonging to the positive class. In such cases, the classification boundary between classes must be determined by a threshold value. As the threshold value changes, the classification of test instances changes.

The receiver operating characteristic curve, or ROC curve, shown in Figure 8.12, illustrates the diagnostic ability of a binary classifier as the threshold of

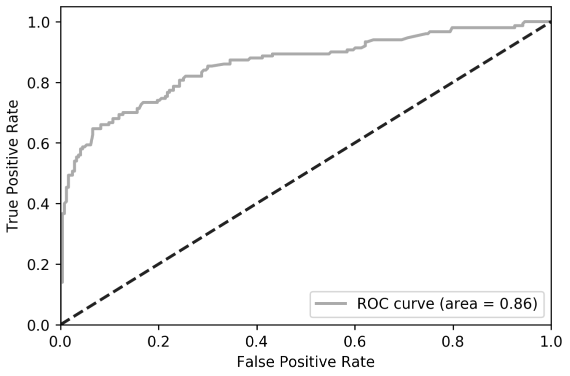


FIGURE 8.12: An example ROC curve from an analysis of the Titanic dataset.

discrimination varies. It plots the true positive rate (TPR) against the false positive rate (FPR).

$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{FPR} = \frac{FP}{TN + FP}$$

Notice that TPR is just another name for recall, and FPR is analogous but for false positives. Each threshold value results in new TPR and FPR values and hence a point on the ROC curve; from any given classifier, we can compute such a curve.

The best possible prediction will yield a point in the upper-left corner of the ROC curve implying a TPR value of 1 and FPR value of 0. If all cases are classified as belonging to the negative class this corresponds to the point $(0, 0)$ and if all cases are classified as belonging to the positive class this corresponds to the point $(1, 1)$.

We can quantify some attributes of a classifier by measurements of its ROC curve. A good classification model will lead to an ROC curve located as close as possible to the upper left corner of the diagram, while a classification model close to random guessing will have an ROC close to the line TPR = FPR.[8] The area under the ROC curve (commonly referred to as area under the curve, AUC) is a common metric used for evaluating the performance of a classifier. A perfect classifier has an AUC of 1, while a random guess classifier has an AUC of 0.5. In fact, AUC can be thought of as the probability that a randomly drawn member of the “negative” class will produce a score lower than the score of a randomly drawn member of the “positive” class [207].

[8] If the curve were entirely below the TPR = FPR line, the modeler should interchange its positive and negative predictions, producing a new classifier that gives better than random predictions.

While we will not go over the construction of the ROC curve in detail, it suffices to say that the test records are first sorted according to their output values and a threshold value corresponding to each of these output values is used to compute the TPR and FPR values on the ROC curve corresponding to the chosen threshold value.

8.7 Naïve Bayes classifier

The naïve Bayes classifier can be used in the same contexts as logistic regression can—for classification. Like logistic regression, it models the probability of each target class based on the features. But unlike logistic regression, it does not use a discriminative function (like logit) but rather assumes the data points within each class are generated from a specific probability distribution (for example, the Bernoulli distribution). For this reason, the naïve Bayes classifier is considered a **generative classifier**, and the logistic regression is considered a **discriminative classifier**.

The learning models we have studied so far have modeled $p(y \mid \mathbf{x}; \theta)$, the conditional probability of the response variable given the predictors. Recall the example of predicting whether a credit card transaction is fraudulent; in such a case, y is a binary response (fraudulent or not), and $\mathbf{x}$ is a vector of the available data about the transaction.

Now we adopt a different approach, studying features that characterize fraudulent transactions and features characterizing legitimate transactions. In this approach we will model $p(\mathbf{x} \mid y; \theta)$, the **class-conditioned** probability. We can use such models to predict whether a new transaction is fraudulent.

8.7.1 Using Bayes' rule

Ignoring θ for a moment for simplicity, **Bayes' rule**,

$$p(y \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y)p(y)}{p(\mathbf{x})}, \tag{8.14}$$

helps us relate $p(\mathbf{x} \mid y)$ to $p(y \mid \mathbf{x})$. Here $p(y)$ represents the **class prior**, i.e., the probability that a randomly chosen outcome $\mathbf{x}$ comes from class y, while $p(y \mid \mathbf{x})$ is the **posterior** probability[9] that an outcome with features $\mathbf{x}$ belongs to class y. Note that to predict the class based on the features, it is

[9]We speak only of probabilities in this section, not probability densities. Section 8.7.1.1 contains a method for handling continuous variables.

sufficient to predict the class y that has the highest $p(\mathbf{x} \mid y)p(y)$ value, since the denominator $p(\mathbf{x})$ is common to each class. In short, if $p(\mathbf{x}) \neq 0$,

$$\arg\max_y p(y \mid \mathbf{x}) = \arg\max_y \frac{p(\mathbf{x} \mid y)p(y)}{p(\mathbf{x})} = \arg\max_y p(\mathbf{x} \mid y)p(y). \tag{8.15}$$

Our goals are thus to estimate $p(\mathbf{x} \mid y)$ and $p(y)$. The **naïve Bayes classifier** estimates the class-conditioned probability $p(\mathbf{x} \mid y)$ in Bayes' rule (8.14) by assuming that the attributes of $\mathbf{x}$ are *conditionally independent* given the class label y, that is, if $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$, then

$$p(\mathbf{x} \mid y) = p(\mathbf{x}_1 \mid y)p(\mathbf{x}_2 \mid y)\cdots p(\mathbf{x}_k \mid y).$$

Let us understand this by way of an example.

Insurance claims are often filed after a traffic accident. An insurance provider would like to determine if the insured party is at fault or not, for claims purposes; let us call this target variable AF, with $AF = 0$ meaning "not at fault" and $AF = 1$ meaning "at fault." An incident report consists of text describing the incident, such as "The insured car jumped the red light and hit a vehicle turning," and so on. We can represent each report as a feature vector with length equal to the number of words in the English language. If a word appears in the report then we assign a 1 to the coordinate corresponding to this word, and otherwise we assign a 0.[10]

The independence assumption stated above says that $p(\text{red} = 1 \mid AF = 1)$ is independent of $p(\text{jumped} = 1 \mid AF = 1)$ that is

$$p(\text{red} = 1 \mid AF = 1) = p(\text{red} = 1 \mid AF = 1, \text{jumped} = 1).$$

This is a strong assumption! For example, the phrase "jumped a red light" is not uncommon, and so the words "jumped" and "red" occur together somewhat often. Thus the associated conditional probabilities are almost certainly not independent. This is why the word "naïve" appears in the name of the technique; we are intentionally ignoring these correlations, for a simpler model. And doing so works fairly well in practice.

Without such a strong assumption, we would also have enormous data storage concerns. Suppose $\mathbf{x}$ represents a vector (in our example a document) where each co-ordinate $\mathbf{x}_i$ is binary and y is also binary. Then the formula for $p(\mathbf{x} \mid y)p(y)$ would depend on over 2^m parameters, since $\mathbf{x}$ has m coordinates. But under conditional independence, we need only about $2m$ parameters! Thus the independence assumption can be viewed as a method for handling the curse of dimensionality (introduced in Section 7.2) in this context.

[10]In practice, a lot of pre-processing is done before such representation vectors are created (e.g., stop words removal, stemming, lemmatization, etc.) but we ignore these details for a simpler example. The reader may recall that this data storage format was called a "term-by-document matrix" and discussed in detail in Section 3.1.2.

TABLE 8.5: Summary of the 1308 fares in the Titanic dataset (left) and a discretization of that data into intervals (right).

	Fare
Mean	33.30
Standard deviation	51.76
Minimum	0.00
25th percentile	7.90
50th percentile	14.45
75th percentile	31.26
Maximum	512.33

Fare interval	Assigned ordinal value
$(-\infty, 7.90)$	1
$[7.90, 14.45)$	2
$[14.45, 31.26)$	3
$[31.26, \infty)$	4

8.7.1.1 Estimating the probabilities

For categorical variables, the conditional probability $p(X = \mathbf{x}_i \mid Y = y)$ can be estimated by the fraction of training instances of class y that have attribute $\mathbf{x}_i$. In the example above, $p(\text{red} = 1 \mid AF = 1)$ can be estimated as the fraction of claim reports in which the insured party was at fault that contain the word "red." However, if the feature is continuous, we can discretize the continuous variable and replace it with a discrete value corresponding to the interval in which the value lies, in effect obtaining an ordinal feature. Once we have the intervals, we can estimate $p(X = x \mid Y = y)$ by computing the fraction of training instances of class y that belong to the interval to which x belongs.

For example, if we consider the famous Titanic dataset, whose fare variable is summarized on the left of Table 8.5, we can divide its values into intervals using the quartiles in the summary statistics, with the result shown on the right of the same table. Thus a \$20 fare would be assigned a feature value of 3. Now we can treat this feature as we did the categorical variables. The estimation error in the conditional probability will depend on the size of the intervals.

Another option is to assume a form for the probability distribution for $p(\mathbf{x} \mid y)$ (for example, a Gaussian distribution), estimate the parameters of the distribution using the training data, and then obtain the probabilities from the realized distribution. As for the prior $p(y)$, we can compute the fraction of cases in the training set that belong to class y.

8.7.1.2 Laplace smoothing

We now know how to estimate $p(y \mid \mathbf{x})$, but one problem remains. Consider a word, such as "drone," that does not appear in any claims report in our training set (perhaps because drones were not in common use when our data was collected), but the word does appear in a report in the test set (because drones are more common now). Let $\mathbf{x}$ represent the feature vector associated to that report. Because the word is not mentioned in the training set, our method estimates

$$p(\text{drone} = 1 \mid AF = 1) = p(\text{drone} = 1 \mid AF = 0) = 0.$$

The following problem arises when we attempt to compute $p(AF = 1 \mid \mathbf{x})$.

$$\frac{\prod_i p(\mathbf{x}^{(i)} \mid AF = 1)p(AF = 1)}{\prod_i p(\mathbf{x}^{(i)} \mid AF = 1)p(AF = 1) + \prod_i p(\mathbf{x}^{(i)} \mid AF = 0)p(AF = 0)} = \frac{0}{0} \qquad (8.16)$$

This problem arose because we erroneously concluded that, because we haven't observed an event (the word "drone") in the finite observations in our training data, it happens with probability zero. Even unobserved events usually have some nonzero probability, however small. The technique called **Laplace smoothing** (or additive smoothing) can resolve this, as follows.

The key problem underlying equation (8.16) is that there are some indices j for which every $\mathbf{x}_j^{(i)} = 0$, thus making $p(\mathbf{x}_j^{(i)} = 1 \mid y = C) = 0$ for any class C. This is because the default way to estimate the probability $p(\mathbf{x}_j^{(i)} = 1 \mid y = C)$ is empirically: Write n_C for the total number of occurrences of word j in training documents of class C and write $|C|$ for the total number of occurrences of any word (from the set of words that we care about) in training documents of class C. Then the following natural empirical estimate for $p(\mathbf{x}_j^{(i)} = 1 \mid y = C)$ has problems when $|C| = 0$.

$$p(\mathbf{x}_j^{(i)} = 1 \mid y^{(i)} = C) = \frac{n_C}{|C|}.$$

We deviate from that problematic default by choosing some other default probability p that is independent of the data we've seen. In a text analysis problem like this one, it is common to choose $p = \frac{1}{k}$, the reciprocal of the number of words in the language. We then create a combination of these two probabilities using a parameter m as follows.

$$p(\mathbf{x}_j^{(i)} = 1 \mid y^{(i)} = C) = \frac{n_C + mp}{|C| + m} \qquad (8.17)$$

Consider how this formula behaves for different values of m. For $m = 0$ we return to the empirical estimate $\frac{n_C}{|C|}$. As $m \to \infty$, this value moves towards the other probability estimate, p. Thus m is a hyperparameter that the data analyst can choose to shift equation (8.17) closer to or farther from the empirical estimate. Although we use the letter m, it need not be an integer; the symbol α is sometimes used. It is common to choose $m = k$, which coupled with $p = \frac{1}{k}$, yields estimates of the form $\frac{n_C+1}{|C|+k}$.

8.7.2 Health care example

Consider the small set of five (fictional) hospital records shown in Table 8.6, classifying each patient as either having a heart condition or being diabetic.

TABLE 8.6: Fictional health care data used in an example in Section 8.7.2.

Dataset	Document ID	Words	Class
Train	1	diabetes, amputation, edema	diabetic
Train	2	insulin, diabetes, kidney	diabetic
Train	3	heart, breath, attack	heart condition
Train	4	amputation, heart	diabetic
Test	5	insulin, diabetes, breath	

TABLE 8.7: Probabilities computed for the example in Section 8.7.2.

j	$p(\mathbf{x}_j^{(i)} = 1 \mid y = \text{diabetic})$	$p(\mathbf{x}_j^{(i)} = 1 \mid y = \text{heart condition})$
diabetes	3/16	1/11
amputation	3/16	1/11
edema	2/16	1/11
insulin	2/16	1/11
kidney	2/16	1/11
heart	2/16	2/11
breath	1/16	2/11
attack	1/16	2/11
Total	16/16	11/11

A real example would include a much larger number of records and a much larger list of words, but we keep this list small so that we can discuss it with ease. We wish to train a model that can predict a patient's class based on the words that appear in the patient's records.

In order to apply the naïve Bayes classifier, we will need to first compute $p(\mathbf{x}_j^{(i)} = 1 \mid y^{(i)} = C)$ for each word j and each class C. We will use equation (8.17) with $m = k$ and $p = \frac{1}{k}$, as suggested above. Because we are focused on only eight different words in this small example, we have $m = 8$ and $p = 0.125$, and equation (8.17) simplifies to $\frac{n_C+1}{|C|+8}$.

Consider just one example, with $C =$ heart condition and say $j = 1$ is the index for the word "diabetes." In such a case, we have $n_C = 0$ and $|C| = 3$, yielding the probability estimate

$$p(\mathbf{x}_1^{(i)} = 1 \mid y = \text{heart condition}) = \frac{0+1}{3+8} = \frac{1}{11}.$$

Repeating such a computation for each j, C yields the results in Table 8.7.

With these probabilities in hand, we can apply the naïve Bayes classifier to the one item in our test dataset (the bottom row of Table 8.6). We apply equation (8.15), meaning that we should compute both $p(\mathbf{x} \mid y = \text{diabetic})$ and $p(\mathbf{x} \mid y = \text{heart condition})$ and find which class gives the larger probability.

Each probability will use the conditional independence assumption inherent in the model. If $C =$ diabetic, then

$$\begin{aligned} & p(y = C \mid \text{insulin} = 1, \text{diabetes} = 1, \text{breath} = 1) \\ \propto\ & p(\text{insulin} = 1 \mid y = C) \cdot p(\text{diabetes} = 1 \mid y = C) \\ & \cdot p(\text{breath} = 1 \mid y = C) \cdot p(y = C) \\ =\ & \frac{2}{16} \cdot \frac{3}{16} \cdot \frac{1}{16} \cdot \frac{3}{4} \approx 0.0011 \end{aligned}$$

A similar computation with $C =$ heart condition would yield approximately 0.0004. Choosing the y with the larger $p(\mathbf{x} \mid y)$ in this case means that our model would predict $y =$ diabetic.

In an applied context where the outcome is crucial, classification decision procedures like this one are not implicitly trusted. For instance, borderline cases may be referred to humans for expert classification. Or in a high-stakes environment such as health care, the automated system may just be a reference estimate that a doctor would use when considering a diagnosis.

The naïve Bayes classifier has many attractive features as a model; it is fast to apply to a test case, robust, and has a low storage requirement. Furthermore, as new training examples become available, it is computationally inexpensive to update the probabilities in Table 8.7 (or the analogous, larger table in a real application) with the new data. In cases where the independence assumptions hold (so that the word "naïve" may be dropped), it is in fact the **Bayes optimal classifier**.

We now take a moment to position naïve Bayes relative to the classification methods we introduce in the upcoming sections. Support vector machines (in Section 8.8) are a powerful technique, but for binary classification only, and thus naïve Bayes remains relevant when the number of classes is three or more. Decision trees (in Section 8.9) are more sophisticated than naïve Bayes, but can suffer from fragmentation (explained in that section), so in domains with many equally important features, naïve Bayes remains the more effective option. Furthermore, even when a practitioner plans to implement a more sophisticated system than naïve Bayes, it is still common to apply naïve Bayes first to use as a baseline for comparison when the more accurate classifier is later implemented.

8.8 Support vector machines

Support Vector Machines (SVMs) are binary classifiers that use separating hyperplanes (explained below) as decision boundaries between two classes. They differ from linear classifiers like logistic regression in that the separating hyperplane is chosen to maximize the *margin* between classes; linear classifiers are happy with simply separating the classes without any regard for margin of separation.

Section 8.8.1 introduces SVMs under the assumption of linearly separable classes. This assumption rarely holds of real world data, but we use it to develop the key ideas, then consider the nonlinearly-separable setting in Section 8.8.2. Finally, we consider nonlinear class boundaries in Section 8.8.3, in which we will introduce the widely-used kernel technique, a generalization of similarity measures based on inner products.

The term Support Vector Classifier (SVC) is often used to describe an SVM without the kernel technique, reserving the term SVM for when the kernel technique is required. We will not make this distinction, but we point it out because the classes in scikit-learn implementing SVMs use this terminology (`SVC`, `NuSVC`, and `LinearSVC`).

8.8.1 Linear SVMs in the case of linear separability

Suppose we have a two-class classification problem in which the two classes are linearly separable, that is, there is a hyperplane such that all points on one side of the hyperplane belong to one class and all points on the other side belong to the other class. In general, we would expect there to be infinitely many hyperplanes that can separate two such classes, each of which would therefore exhibit zero error on the training dataset. And yet not all such hyperplanes will perform equally well when exposed to unseen data.

We therefore seek a hyperplane that provides the largest separation between the classes, that is, the training points from each class are as far as possible from the hyperplane, as in Figure 8.13. Such a classification problem is of course predicated on the idea that data in the same class sit near to one another, and thus the margin shown in the figure gives the word "near" as much flexibility as possible before the chosen hyperplane would misclassify a test example.

Let $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, 2, \ldots, n\}$ be a set of training data and for convenience let the classes be $+1$ and -1. As before, each $\mathbf{x}^{(i)}$ is a vector of length p. The training boundary for a linear classifier can be expressed in the form

$$\mathbf{x} \cdot \theta + \theta_0 = 0,$$

where θ and θ_0 are parameters of the model. For any point $\mathbf{x}_+$ above the decision boundary, $\mathbf{x}_+ \cdot \theta + \theta_0 > 0$ and for any point $\mathbf{x}_-$ below the decision boundary, $\mathbf{x}_- \cdot \theta + \theta_0 < 0$.

Suppose the class label of points above the hyperplane $\mathbf{x} \cdot \theta + \theta_0 = 0$ is $+1$ and the label for those below is -1. Let X_+ be the set of $\mathbf{x}$ vectors in the $+1$ class that are closest to the hyperplane $\mathbf{x} \cdot \theta + \theta_0 = 0$; define X_- analogously. Then the $\mathbf{x} \in X_+$ lie on the hyperplane $\mathbf{x} \cdot \theta + \theta_0 = k_+$ for some constant k_+ and the $\mathbf{x} \in X_-$ lie on some $\mathbf{x} \cdot \theta + \theta_0 = k_-$, with $k_- < 0 < k_+$. The sets X_+ and X_- play a special role, as we will see in upcoming sections.

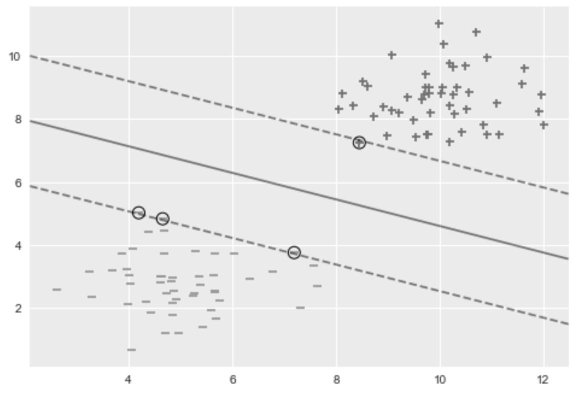


FIGURE 8.13: A two-class classification problem (positive and negative) with two features, corresponding to the x- and y-axes. The solid line is a hyperplane maximizing its distance from the points on either side, with the margins shown by the dashed lines. Support vectors are circled. SVMs can be used in any number of dimensions, but we choose two here for ease of visualization.

We can rescale θ and θ_0 so that the above **margin** hyperplanes satisfy the equations

$$H_+ : \mathbf{x} \cdot \theta + \theta_0 = 1$$

$$H_- : \mathbf{x} \cdot \theta + \theta_0 = -1.$$

The distance between H_+ and H_- is therefore given by $\frac{2}{\|\theta\|}$. We therefore have two complementary facts; if $y = +1$ then $\mathbf{x} \cdot \theta + \theta_0 \geq 1$ and if $y = -1$ then $\mathbf{x} \cdot \theta + \theta_0 \leq -1$. We can express these two facts together as $(\mathbf{x} \cdot \theta + \theta_0)y \geq 1$.

We can use this as a constraint in an optimization problem. We would like to find θ and θ_0 that maximize the distance $\frac{2}{\|\theta\|}$ between the margins while respecting the constraints

$$(\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} \geq 1 \qquad \forall i = 1, 2, \ldots, n. \tag{8.18}$$

Reframing this in terms of a cost, we can say we want to minimize the cost function $\frac{\|\theta\|^2}{2}$ subject to the same constraints.

Using Lagrange multipliers, the objective function $\frac{\|\theta\|^2}{2}$ and the constraints in (8.18) can be expressed as the task of minimizing

$$J(\theta) = \frac{\|\theta\|^2}{2} - \sum_{i=1}^{n} \lambda_i((\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} - 1). \tag{8.19}$$

We must assume that $\lambda_i \geq 0$, because if we do not do so, then any unmet constraint could make $J(\theta)$ unbounded below as follows. Suppose $(\mathbf{x}^{(i)} \cdot \theta +$

$\theta_0)y^{(i)} < 1$; then we can choose a large negative λ_i so that $-\lambda_i((\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} - 1)$ is as negative as we like. This lets us decrease $J(\theta)$ indefinitely outside the feasible region, and thus its minimum will be outside the feasible region. It would not be a faithful representation of the constraints in (8.18), so we require each $\lambda_i \geq 0$.

The resulting problem (8.19) is a convex optimization problem, so we can try to solve it analytically in the usual way.

$$\frac{\delta J}{\delta \theta} = \theta - \sum_{i=1}^{n} \lambda_i y^{(i)} \mathbf{x}^{(i)} = 0 \tag{8.20}$$

$$\frac{\delta J}{\delta \theta_0} = -\sum_{i=1}^{n} \lambda_i y^{(i)} = 0 \tag{8.21}$$

But the Lagrange multipliers are not yet known, so we do not have an explicit formula for θ and θ_0.

Chapter 6 discussed Lagrange duality in detail, and we can apply it here to arrive at the following constraints on the Lagrange multipliers, which Chapter 6 called the KKT conditions.

1. $\lambda_i \geq 0 \quad \forall i = 1, 2, \ldots, n$
2. $\lambda_i((\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} - 1) = 0$ for $i = 1, 2, \ldots, n$.

The second of these conditions implies that $\lambda_i = 0$ or $(\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} = 1$. Thus λ_i can be greater than 0 only for those training points $(\mathbf{x}^{(i)}, y^{(i)})$ for which $(\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} = 1$, that is, the point $(\mathbf{x}^{(i)}, y^{(i)})$ lies on the boundary. The $\mathbf{x}^{(i)}$'s for which $\lambda_i > 0$ are called the **support vectors**.

Using the constraints (8.20) and (8.21) on λ_i we can rewrite the (primary) optimization problem as

$$\frac{\|\theta\|^2}{2} - \sum_{i=1}^{n} \lambda_i((\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} - 1) \tag{8.22}$$

$$= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} + \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \lambda_i y^{(i)} \mathbf{x}^{(i)} \cdot \left(\sum_{j=1}^{n} \lambda_j y^{(j)} \mathbf{x}^{(j)} \right)$$

$$= \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \tag{8.23}$$

Equation (8.23) is the dual Lagrangian formulation of the optimization problem and is entirely in terms of the Lagrange multipliers and the training data. Instead of a primary problem which was framed as a minimization problem, this dual formulation is a maximization problem and can be solved using quadratic programming or gradient descent. Once the multipliers are found, we can compute an estimate $\hat{\theta}$ using (8.20). Since the λ_i are usually computed numerically, they are only approximations, hence $\hat{\theta}_0$ may not be exact, but can be estimated as the average of the θ_0 values obtained by solving the equation $y^{(i)}(\theta \cdot \mathbf{x}^{(i)} + \theta_0) = 1$ for each support vector $(\mathbf{x}^{(i)}, y^{(i)})$.

To predict the class of a test case $\mathbf{z}$, we can use our estimated $\hat{\theta}$ and $\hat{\theta}_0$ as follows.

$$y = \begin{cases} +1 & \text{if } \mathbf{z} \cdot \hat{\theta} + \hat{\theta}_0 > 0 \\ -1 & \text{if } \mathbf{z} \cdot \hat{\theta} + \hat{\theta}_0 < 0 \end{cases}$$

If we cared only about the training data, then considering when $\mathbf{z} \cdot \hat{\theta} + \hat{\theta}_0 \geq 1$ or $\mathbf{z} \cdot \hat{\theta} + \hat{\theta}_0 \leq -1$ would suffice. But this is precisely the point of SVMs; we have created an optimal margin around the training data, so that test data that may satisfy $\mathbf{z} \cdot \hat{\theta} + \hat{\theta}_0 \in (-1, 1)$ is also classified sensibly.

8.8.2 Linear SVMs without linear separability

In practice it is unlikely that one finds a hyperplane that perfectly separates two classes. Because real data is noisy, there will usually be some points that lie on the wrong side of any hyperplane we might try to use as a classifier. In such cases, we may want to find a hyperplane that provides the best separation between the classes while ensuring that points on the wrong side of the dividing hyperplane are not too far from it. Let $\xi_i \geq 0$ represent the **slack**, the amount by which the i^{th} training point is on the *wrong* side of the margin (or zero if it is correctly classified), as in Figure 8.14. We can reframe the problem from Section 8.8.1 as one of minimizing

$$\frac{\|\theta\|^2}{2} + C\sum_{i=1}^{n} \xi_i \tag{8.24}$$

such that for $i = 1, 2, \ldots, n$,

$$(\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} \geq 1 - \xi_i \qquad \text{and} \qquad \xi_i \geq 0.$$

The second term in the objective function penalizes a decision boundary with large slack variables and C is a hyperparameter that encapsulates the tradeoff between minimizing the training error and maximizing the margin. Figure 8.15 shows that lower values of C increase the margin and the training error but may generalize better to new data, as we will see at the end of this section.

The associated Lagrange primal problem can be framed as

$$L_p = \frac{\|\theta\|^2}{2} + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \lambda_i((\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} - (1 - \xi_i)) - \sum_{i=1}^{n} \mu_i \xi_i \tag{8.25}$$

where λ_i and μ_i are Lagrange multipliers. Like earlier, these lead to the following KKT conditions.

1. $\xi_i \geq 0$, $\lambda_i \geq 0$, and $\mu_i \geq 0$

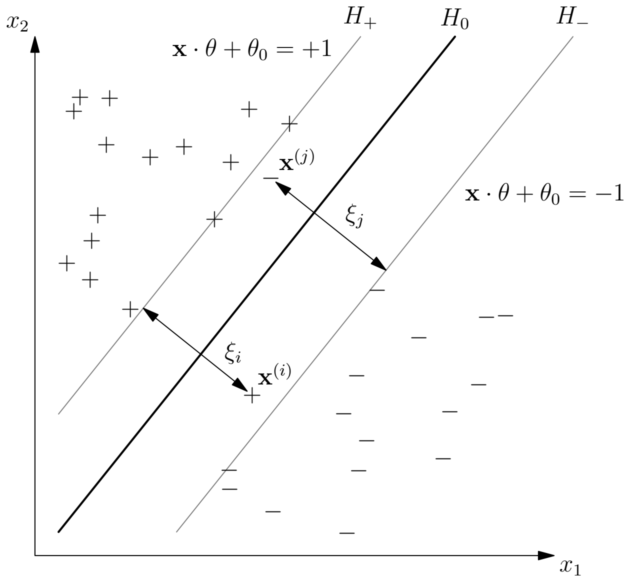

x_2
H_+
H_0
H_-
$\mathbf{x} \cdot \theta + \theta_0 = +1$
$\mathbf{x}^{(j)}$
ξ_j
$\mathbf{x} \cdot \theta + \theta_0 = -1$
ξ_i
$\mathbf{x}^{(i)}$
x_1

FIGURE 8.14: Illustration of slack variables, as introduced in Section 8.8.2.

2. $\lambda_i((\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} - (1 - \xi_i)) = 0$
3. $\mu_i \xi_i = 0$

The associated dual problem can be obtained by combining these conditions with equation (8.25), yielding

$$\sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y^{(i)} y^{(j)} \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}. \tag{8.26}$$

This is very similar to the separable case except that $0 \leq \lambda_i \leq C$, which follows from the constraint $\frac{\delta Lp}{\delta \xi_i} = 0$ and $\lambda_i \geq 0$ and $\mu_i \geq 0$.

The solution for $\hat{\theta}$ is again expressed in terms of the fitted Lagrange multipliers λ_i,

$$\hat{\theta} = \sum_{i=1}^{n} \lambda_i y^{(i)} \mathbf{x}^{(i)},$$

and one can compute $\hat{\theta}_0$ as before. We can determine the class of any new test vector $\mathbf{z}$ by checking the sign of $\mathbf{z} \cdot \hat{\theta} + \hat{\theta}_0$ and again only the support vectors matter.

Notice the computational efficiency implicit in the previous sentence. Applying the model to a new observation $\mathbf{z}$ requires computing the dot product and sum $\mathbf{z} \cdot \hat{\theta} + \hat{\theta}_0$ using vectors of length n, the size of the training data.

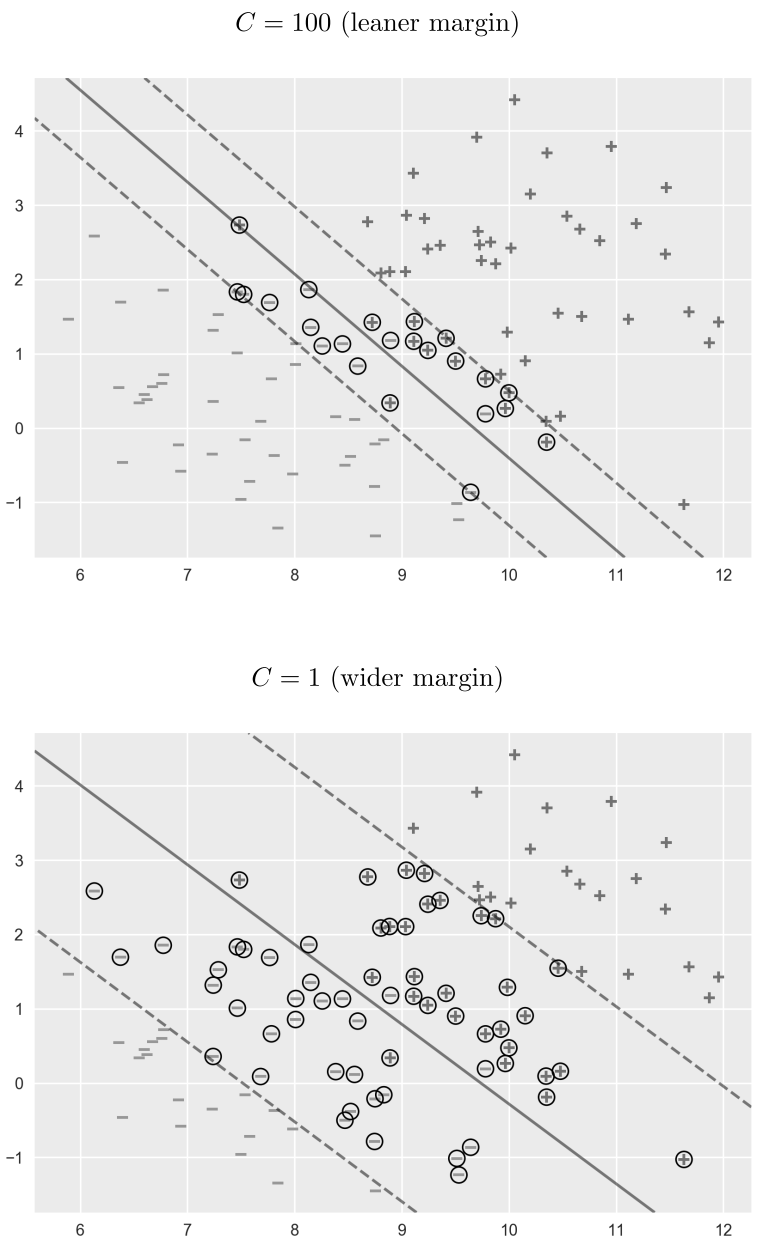


FIGURE 8.15: SVM with nonlinearly-separable data shown using two different C values, but the same set of data. The smaller C value leads to more $\xi_i > 0$, but also more data points are used in the computation of $\hat{\theta}$ and $\hat{\theta}_0$.

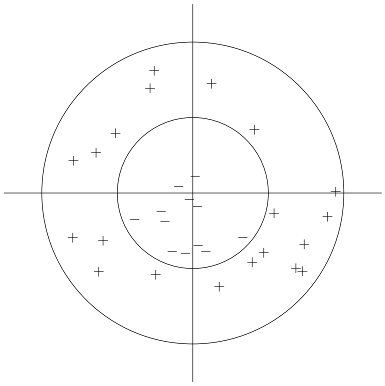


FIGURE 8.16: Classes separated by a nonlinear boundary.

Some applications include vast amounts of training data, and thus this may be an expensive loop to run. But since we know in advance that most of the λ_i are zero (since typically a small portion of a large dataset are support vectors) we can pre-compute just the list of indices i for which λ_i is nonzero, and significantly reduce the computational cost of applying our model to new data.

Considering the second of the KKT conditions given above, we see that if $(\mathbf{x}^{(i)} \cdot \theta + \theta_0)y^{(i)} - (1 - \xi_i) \neq 0$ then we must have $\lambda_i = 0$. This happens precisely when $\mathbf{x}^{(i)}$ is *not* one of the support vectors. Consequently the formula for $\hat{\theta}$ will include only those terms where $\mathbf{x}^{(i)}$ is a support vector. Since we already know that the method for computing $\hat{\theta}_0$ uses only support vectors, we see their crucial role in impacting the classification results of this model—indeed, no other data points come into play! Thus we see the importance of the hyperparameter C, which chooses how many of the data points will be used as support vectors, as shown in Figure 8.15.

8.8.3 Nonlinear SVMs

The support vectors we have studied so far do not allow for nonlinear decision boundaries. For example, consider two classes placed in concentric circles in $\mathbb{R}^2$, with

$$y = \begin{cases} -1 & \text{if } \mathbf{x}_1^2 + \mathbf{x}_2^2 < 1 \\ +1 & \text{if } \mathbf{x}_1^2 + \mathbf{x}_2^2 \geq 1, \end{cases}$$

as in Figure 8.16.

There is no linear boundary that separates the two classes. But all is not lost. If we are willing to view the same data in a higher dimensional space, we might succeed. If we view this data in $\mathbb{R}^3$ by embedding $(\mathbf{x}_1, \mathbf{x}_2)$ as $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_1^2 + \mathbf{x}_2^2)$, then points of class -1 are clearly below (in the third coordinate) the points of class +1. Hence the two classes can be separated by a linear boundary that corresponds to a circle in $\mathbb{R}^2$.

Just as we have studied how to extend linear regression to nonlinear regression using arbitrary basis functions $b_i(\mathbf{x})$, such as polynomials, in

$$y = \theta_q b_q(\mathbf{x}) + \theta_{q-1} b_{q-1}(\mathbf{x}) + \cdots + \theta_1 b_1(\mathbf{x}) + \epsilon,$$

we can take a similar approach with SVM classifiers and consider arbitrary transformations $\phi : \mathbb{R}^p \to \mathbb{R}^q$ given by

$$\phi(\mathbf{x}) = (b_1(\mathbf{x}), b_2(\mathbf{x}), \ldots, b_q(\mathbf{x})).$$

For suitable ϕ, this approach produces a linear boundary in the transformed space for a nonlinear boundary in the original space.

There are two potential problems with this approach. First, we do not know which transformation ϕ will produce a linear boundary in the transformed space. Second, even if we know an appropriate map ϕ, it may be computationally expensive to solve the constrained optimization problem in a high-dimensional feature space.

Let us handle the second problem first. Suppose we have an appropriate map ϕ, so that the problem can be represented as a minimization of $\frac{\|\theta\|^2}{2}$ such that

$$(\phi(\mathbf{x}^{(i)}) \cdot \theta + \theta_0) y^{(i)} \geq 1, \qquad \forall i = 1, 2, \ldots, n.$$

This is the optimization problem we saw in the linear SVM case, except that $\mathbf{x}$ has been replaced by $\phi(\mathbf{x})$. The associated dual problem can we written as

$$\sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y^{(i)} y^{(j)} \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)}). \tag{8.27}$$

Here $\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$ measures the similarity of $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ in the transformed space. If the dimension of the transformed space to which $\mathbf{x}$ is mapped is large, computing $\phi(\mathbf{x})$ and $\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$ can be computationally intensive. But there is some good news here. We may be able to compute the inner product above in the original space!

Let us consider an example. Let $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, $c > 0$ and $\phi : \mathbb{R}^2 \to \mathbb{R}^6$ be given by

$$\phi(\mathbf{x}) = (\mathbf{x}_1^2, \mathbf{x}_2^2, \sqrt{2}\mathbf{x}_1\mathbf{x}_2, \sqrt{2c}\mathbf{x}_1, \sqrt{2c}\mathbf{x}_2, c).$$

This general transformation has many properties we won't investigate here, but one worth noting is that a circle maps to a linear boundary in the transformed space, because $\mathbf{x}_1^2 + \mathbf{x}_2^2 = c$ can be written as $\phi(\mathbf{x}) \cdot \theta + \theta_0$ when $\theta = (1, 1, 0, 0, 0)$ and $\theta_0 = -c$. If we compute $\phi(\mathbf{u}) \cdot \phi(\mathbf{v})$, we see that it can be expressed in terms of $\mathbf{u}$ and $\mathbf{v}$ directly.

$$\begin{aligned}
&\phi(\mathbf{u}) \cdot \phi(\mathbf{v}) \\
&= (\mathbf{u}_1^2, \mathbf{u}_2^2, \sqrt{2}\mathbf{u}_1\mathbf{u}_2, \sqrt{2c}\mathbf{u}_1, \sqrt{2c}\mathbf{u}_2, c) \cdot (\mathbf{v}_1^2, \mathbf{v}_2^2, \sqrt{2}\mathbf{v}_1\mathbf{v}_2, \sqrt{2c}\mathbf{v}_1, \sqrt{2c}\mathbf{v}_2, c) \\
&= \mathbf{u}_1^2\mathbf{v}_1^2 + \mathbf{u}_2^2\mathbf{v}_2^2 + 2\mathbf{u}_1\mathbf{u}_2\mathbf{v}_1\mathbf{v}_2 + 2c\mathbf{u}_1\mathbf{v}_1 + 2c\mathbf{u}_2\mathbf{v}_2 + c^2 \\
&= (\mathbf{u} \cdot \mathbf{v} + c)^2
\end{aligned}$$

In general, we define $K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v})$ to be the **kernel function** induced by ϕ. Just as in equation (8.27), the dual problem does not depend on the exact form of the mapping function ϕ but rather on the induced kernel function.

This is of particular value when the computation of the inner product on the transformed space can be done on the much smaller original space. Which brings us to the question of which ϕ one can choose so that $\phi(\mathbf{u}) \cdot \phi(\mathbf{v})$ behaves nicely. The following theorem is quite useful.

Theorem 7 (Mercer) *A kernel function K can be expressed as $K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v})$ if and only if for any function $g(x)$ with $\int g^2(x)dx$ finite,*

$$\int K(\mathbf{x}, \mathbf{y})g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y} \geq 0.$$

It can be used to show the value of the following three popular kernel functions.

1. the polynomial kernel, $(\mathbf{u} \cdot \mathbf{v} + c)^q$ for $q \in \mathbb{N}$
2. the sigmoid kernel, $\tanh(a(\mathbf{u} \cdot \mathbf{v}) + b)$ for $a, b \geq 0$
3. the Gaussian kernel, $e^{-\|\mathbf{u}-\mathbf{v}\|^2/2\sigma^2}$ for $\sigma \neq 0$

We leave it to the reader to verify that these indeed are kernel functions.

Another characterization of kernel functions besides Theorem 7 is as follows. Given any m points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}$ and a kernel function K, we can consider an $m \times m$ kernel matrix M such that $M_{i,j} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$. The function K is a kernel function if and only if the associated kernel matrix is symmetric positive semi-definite for any m points, $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}$ [431, chapter 3].

Figure 8.17 shows two classes, one comprised of two independent normal random variables $\mathbf{x}_1, \mathbf{x}_2$ and the other satisfying $\mathbf{x}_1^2+\mathbf{x}_2^2 > 4$. Note the presence of some lighter points outside the circle $\mathbf{x}_1^2+\mathbf{x}_2^2 = 4$. A nonlinear support vector machine with Gaussian kernel was used to separate the two classes, and its decision boundary is the loop shown in the figure.

The Gaussian kernel is also called the **Gaussian radial basis function** (RBF). This kernel takes smaller and smaller values as two points $\mathbf{x}$ and $\mathbf{y}$ are further apart in the original space. Surprisingly, the associated transformation

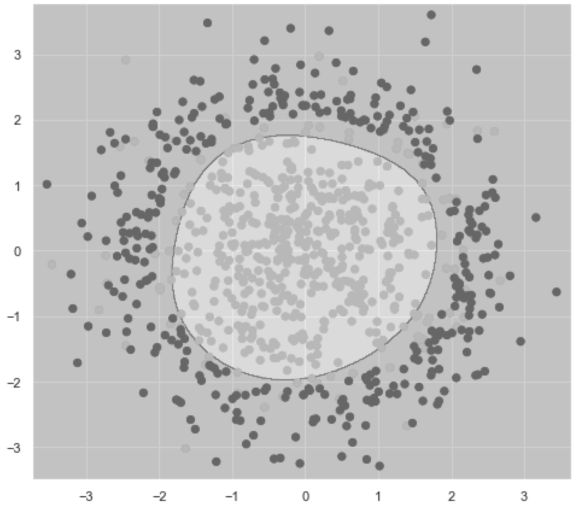


FIGURE 8.17: A nonlinear SVM with Gaussian kernel.

space of ϕ for the RBF kernel is infinite dimensional, even though computing the kernel is easy.

Many machine learning methods besides SVMs benefit from this **kernel trick** where a computation that requires $\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$ can instead be done in the input space. For example, this method can be used in logistic regression [520], linear regression, and k-means clustering. This trick can be leveraged to focus calculations locally or indirectly increase the feature space.

8.9 Decision trees

The final important machine learning model we cover here is decision trees, a conceptually simple, non-parametric technique supporting both classification and regression. It stratifies the feature space into simple regions (usually rectangular) and then fits a constant model on each simple region. Typical constant models are majority class (mode) for classification problems and mean for regression problems.

This has much more geometric flexibility than other methods, which comes with both advantages and disadvantages. For example, consider the binary data shown in each image in Figure 8.18. While there may be kernel functions that could make the decision boundary a single plane in a higher-dimensional space, it would be nice to have a technique that is capable of learning the

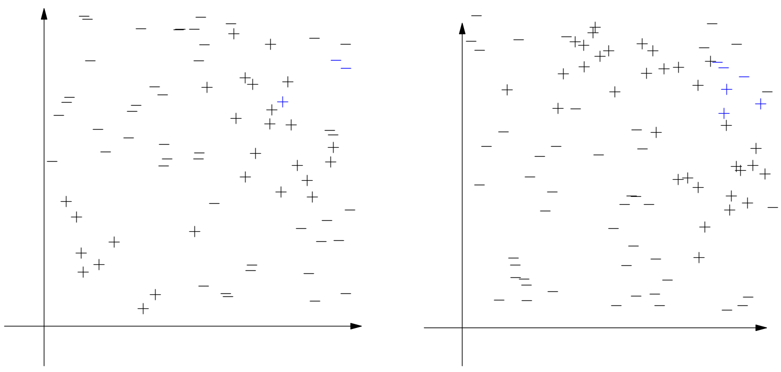


FIGURE 8.18: Two examples of binary classification datasets with nontrivial geometric structure.

shape of each class from the data itself, which decision trees can do. The corresponding disadvantage is concern about fitting the model too well to the data, because it is capable of tailoring itself very well to the shape of the data it is given.

The key idea behind decision trees is to ask a sequence of carefully chosen questions about the features. Each response leads to a follow-up question until the model has narrowed its focus to a small enough subset of the data that it has a reasonable guess about the correct class (in the case of a classification problem) or an appropriate value (in the case of a regression problem). These questions can be organized in a hierarchical structure of nodes and directed edges, a **tree**, with the typical graph-theoretic meaning. Unlike trees in nature, decision trees are represented with the **root** on top, so that the questions one asks proceed down the page. All nodes except the root have exactly one incoming edge.

Non-leaf nodes contain a decision rule, embodying the idea of asking a question of the data and proceeding down the appropriate branch based on the response received. Leaf nodes, instead, are assigned a class label or value, and are used for prediction. To use a decision tree to classify an observation, one begins at the root and follows branches downward based on the relationship between the decision rules at each node and the observation in question. When one reaches a leaf, it is used to classify the observation.

Consider Table 8.8, containing responses to people's desire to play tennis given various weather conditions. A decision tree created from this data appears in Figure 8.19, with the root at the top and three leaves at the bottom. Each node contains three values: The top is the best prediction of whether people wish to play tennis given by that node (yes or no); the middle is the percentage of the training data that passes through the node; and the bottom is the proportion of that training data that is in the yes category.

TABLE 8.8: A table of data about whether people wish to play tennis on various days, based on the weather for the day.

Outlook	**Temperature**	**Humidity**	**Windy**	**Play tennis**
sunny	hot	high	no	no
sunny	hot	high	yes	no
overcast	hot	high	no	yes
rainy	mild	high	no	yes
rainy	cool	normal	no	yes
rainy	cool	normal	yes	no
overcast	cool	normal	yes	yes
sunny	mild	high	no	no
sunny	cool	normal	no	yes
rainy	mild	normal	no	yes
sunny	mild	normal	yes	yes
overcast	mild	high	yes	yes
overcast	hot	normal	no	yes
rainy	mild	high	yes	no

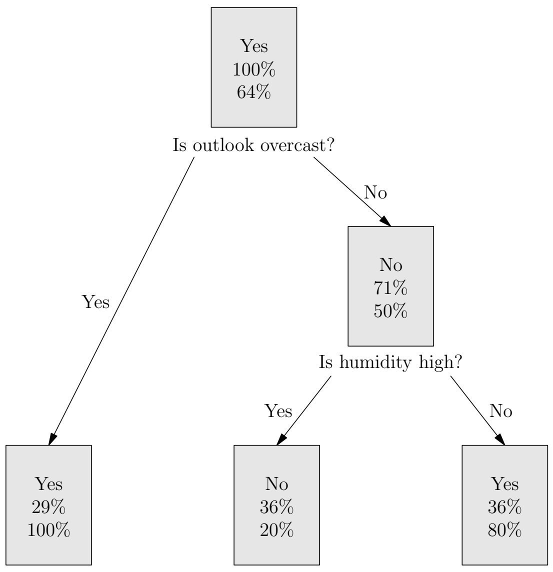


FIGURE 8.19: A decision tree for playing tennis, computed from the data in Table 8.8. See Section 8.9 for an explanation of the format.

TABLE 8.9: Possible binary splits on the outlook feature from Table 8.8.

Left branch	**Right branch**
{sunny, overcast}	{rainy}
{sunny, rainy}	{overcast}
{rainy, overcast}	{sunny}

For instance, at the root node, we see that the best guess in general for whether people wish to play tennis is yes, which was computed from all 100% of the training data, 64% of which had the target variable as yes. But if the weather were sunny with high humidity, we could follow the appropriate path downward from the root, ending at the leftmost leaf, and predict that people are not likely to wish to play tennis in that situation. Our prediction would be based on the 36% of the training data that has those attributes, because in only 20% of those observations was the answer a yes.

8.9.1 Classification trees

Although we have mentioned that trees can be used for both classification and regression problems, let us focus first on classification. The model we discuss below forms the basis of most leading decision tree models today. Though our model below does not restrict the number of splits at a node, most libraries implementing decision trees limit trees to binary splits. This is due to the large number of partitions ($2^{q-1} - 1$) that are possible for an unordered variable with q values, even in the case of binary splits. For instance, consider from Table 8.8 the predictor "outlook," which has three levels. We must consider the $2^2 - 1$ possible ways to separate its values, which are in Table 8.9.

Building a decision tree with minimum training error can be computationally very expensive. In fact, finding such a decision tree is an NP-complete problem [241]. Instead of searching for the best tree, we will use a greedy approach to building a tree that is locally optimal and forms the basis of leading decision tree models like classification and regression tree (CART, used in scikit-learn), ID3, C4.5, and others.

Initialize the tree with a single node, the root. Let N_t denote the training records associated to node t, and place all training records in N_{root}. Let the list of all classes be $c_1, \ldots, c_k$. If all records in N_t belong to the same class, we say node t is a **pure** node and we declare node t a leaf and label it with that one class. Otherwise, we use some splitting criterion I (to be discussed below) to split N_t into smaller subsets that are used to form new **child** nodes. We can apply this process recursively to grow each branch of the tree until each ends in a leaf.

We will refine this procedure further, but first let us discuss some measures commonly used as splitting criteria. In each,

$$p(c_i \mid t) = \frac{|\{j \mid y^{(j)} = c_i\}|}{|N_t|}$$

is the probability that an observation belongs to class c_i given that it is in N_t. We will apply each to the root node in the tennis example, which has 14 instances with 9 yes responses and 5 no responses.

1. **Classification error**: The classification error at node t is defined to be

   $$1 - \max_i p(c_i \mid t). \tag{8.28}$$

   This splitting criterion measures the error that would be made if each member of the node were assigned the *most common* class in N_t.

   The classification error at the root node is $1 - \frac{9}{14} \approx 0.357$.

2. **Gini index**: The Gini index at node t is defined as the total variance at node t across all classes,

   $$\sum_i p(c_i \mid t)(1 - p(c_i \mid t)) = 1 - \sum_i p(c_i \mid t)^2. \tag{8.29}$$

   The Gini index at the root is $1 - \left(\left(\frac{5}{14}\right)^2 + \left(\frac{9}{14}\right)^2\right) \approx 0.459$.

3. **Entropy**: The entropy at node t is given by

   $$-\sum_i p(c_i \mid t) \log_2(p(c_i \mid t)), \tag{8.30}$$

   where $0 \log_2 0$ is treated as 0.

   The entropy at the root is $-\left(\frac{5}{14}\log_2(\frac{5}{14}) + \frac{9}{14}\log_2(\frac{9}{14})\right) \approx 0.940$.

In Figure 8.20, we can see the three splitting criteria together for a binary classification problem where the horizontal axis represents the fraction of positive instances at a node and the vertical axis represents the splitting criterion value. Each splitting criterion takes values close to zero if all $p(c_i \mid t)$ are near zero or one. They take small values when the node is pure and higher values otherwise; they are measures of impurity at a node.

Since our goal is to achieve purer nodes, we can split a node into child nodes based on any impurity criterion, call it I in general. For instance, if a parent node with observations N_{parent} is split into two child nodes with observations N_{left} and N_{right}, the quality of the split can be defined as

$$\Delta = I(\text{parent}) - \left(\frac{|N_{\text{left}}|}{|N_{\text{parent}}|} I(\text{left}) + \frac{|N_{\text{right}}|}{|N_{\text{parent}}|} I(\text{right})\right).$$

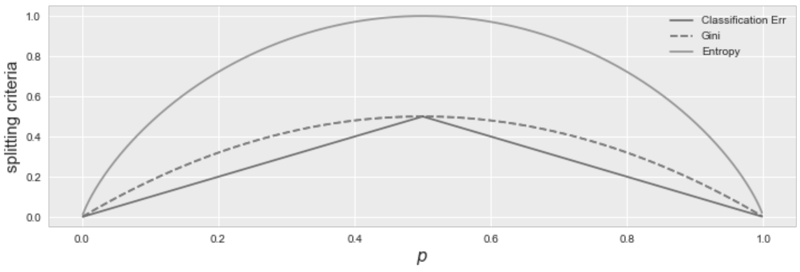


FIGURE 8.20: Splitting criteria in a binary classification setting, with entropy on top, Gini index dashed, and classification error on bottom.

TABLE 8.10: Gains in purity by splitting on outlook at the root with the Gini index as the splitting criterion.

	Sunny		**Overcast**		**Rainy**	
Play tennis	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**
yes	2	7	4	5	3	6
no	3	2	0	5	2	3
I(child)	0.480	0.346	0.000	0.500	0.480	0.444
Δ	0.065		0.102		0.002	

We use Δ to measure the gain in purity by splitting on attribute i, and we develop strategies for splitting a node in a way that maximizes Δ.

In our tennis example, let us compute the gain in purity for various ways that we split the root node using the outlook variable as measured by the Gini index. Outlook has three categories, so we must reduce it into a binary decision; first consider the split overcast vs. not overcast. We can compute the gain in purity at this split as

$$\Delta \approx 0.459 - \left(\frac{4}{14} \cdot 0.0 + \frac{10}{14} \cdot 0.5\right) \approx 0.102.$$

We could repeat this calculation for other ways to split outlook, and for other features as well. Table 8.10 shows a portion of those calculations, focusing on just the categories in the outlook variable for this example. The overcast vs. not overcast binary split provides the greatest gain in purity. As Figure 8.19 shows, we get a pure node with preference to play tennis if it is overcast.

While the three splitting criteria are quite similar numerically, classification error is less sensitive to changes in class probabilities, so entropy and Gini index are more commonly used. But these measures tend to favor attributes with a large number of distinct values, which can result in overfitting, leading to unreliable predictions on unseen data.

To resolve this problem, one can use the splitting criterion that maximizes the gain ratio, defined as Δ divided by the split information, which is the entropy of the class probabilities of the splits. For example, if a node with 20 observations splits into two nodes with 5 and 15 observations then the split information is

$$-(0.25 \log_2(0.25) + 0.75 \log_2(0.75)).$$

It is possible that a split may yield child nodes where each new node has the same predicted value. Though this may seem strange at first, recall that a split is performed if there is an increase in purity. For example, if one of the resulting nodes is pure with class y and the other node has 60% members in class y, though both the nodes have the same label, we can be more confident of the class of members in the first node.

On performing a split it may happen that one resulting node is empty. This can happen if none of the training data has the combination of predictor values associated with the split. In this case, we simply declare the parent node a leaf and let its predicted class be the most common class among its members.

Since real data can throw up surprises, it is possible the dataset has observations that agree on all feature variables yet have different class labels. For instance, two customers may have exactly the same attributes, but one gets approved for a credit card and other is rejected. If two observations in a node have exactly the same combination of feature values, yet they belong to different classes, we do not perform a split, but declare the node a leaf, and again, let its predicted class be the most common class among its members.

So far, we have discussed only decision trees whose attributes are categorical. We can naturally extend this to ordinal and continuous attributes as well. Ordinal attributes are a special type of categorical data that come with an additional restriction: when splitting the values into a left set L and a right set R, every element of L must be less than every element of R. Continuous attributes can be discretized as in Table 8.5 and then treated as ordinal attributes.

We will put our discussion of classification trees on pause for a moment while we consider using decision trees for regression, but there are additional improvements to the application of decision trees in general (for both classification and regression) that we return to in Section 8.9.3, and in some ways in Section 8.10 as well.

8.9.2 Regression decision trees

When the response variable is continuous, we make some minor modifications to the technique of the previous section. As in the classification case, a decision tree will partition the feature space into disjoint regions $R_1, R_2, \ldots, R_q$. Usually these will be high-dimensional rectangles, because they will be formed by conditions of the form $\mathbf{x}_j \leq t$, if $\mathbf{x}_j$ is continuous. For every observation that falls into one of these regions, we will make the same prediction. Unlike the majority vote in the case of classification trees, here we will assign the mean of all response values for training observations in the region. When forming the regions, we would like to minimize the RSS given by

$$RSS = \sum_{j=1}^{q} \sum_{\mathbf{x}^{(i)} \in R_j} (y^{(i)} - \bar{y}_{R_j})^2,$$

where $\bar{y}_{R_j}$ is the mean of the response variable across all training observations in region R_j. Since it is computationally infeasible to consider every possible partition of the feature space, just as it was in the classification problem, we can take an inductive greedy approach.

Again, start with a root containing all observations. At every node our goal is to find the best binary split based on the splitting variable $\mathbf{x}_j$ and splitting point s that minimizes the RSS across the resulting regions. Formally, let us define

$$R_{\text{left}}(j,s) = \{\mathbf{x}^{(i)} \mid \mathbf{x}_j^{(i)} \leq s\} \text{ and } R_{\text{right}}(j,s) = \{\mathbf{x}^{(i)} \mid \mathbf{x}_j^{(i)} > s\}.$$

Then our goal is to find j and s that minimize

$$\sum_{\mathbf{x} \in R_{\text{left}}(j,s)} (y^{(i)} - \bar{y}_{R_{\text{left}}(j,s)})^2 + \sum_{\mathbf{x} \in R_{\text{right}}(j,s)} (y^{(i)} - \bar{y}_{R_{\text{right}}(j,s)})^2. \qquad (8.31)$$

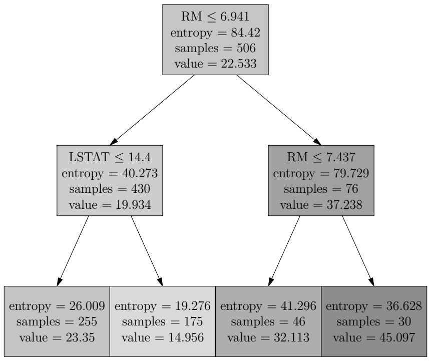
For example, let us consider the Boston housing data and build a regression decision tree for MEDV, using only the variables LSTAT and RM mentioned earlier. At the root there are 506 observations with a mean MEDV of 22.53. At this point the best split is with $j = \text{RM}$ and $s = 6.941$. (One can find the optimal s value by considering all midpoints between two adjacent values of the variable, in this case RM.) Repeating the split on each of the child nodes creates a tree with seven nodes, having depth two, as shown in Figure 8.21. It shows a model that, for example, assigns all houses with LSTAT < 14.4 and RM < 6.941 a MEDV value of 23.35.

We now return to considering modifications to the tree-building process, which can benefit both classification and regression trees.

8.9.3 Pruning

The previous two sections described algorithms for splitting nodes, but when is the tree-building process finished? We could split every node until all nodes are pure or until all members of every node share the same attribute values. While these approaches are sufficient to stop the tree growth, they would typically make the tree very deep, which typically indicates overfitting. One way to address this issue is to **prune** the decision tree using one of the following strategies.

1. **Prepruning** prunes the tree before it is fully generated, by setting a stringent criterion on the minimum improvement in Δ (the gain in



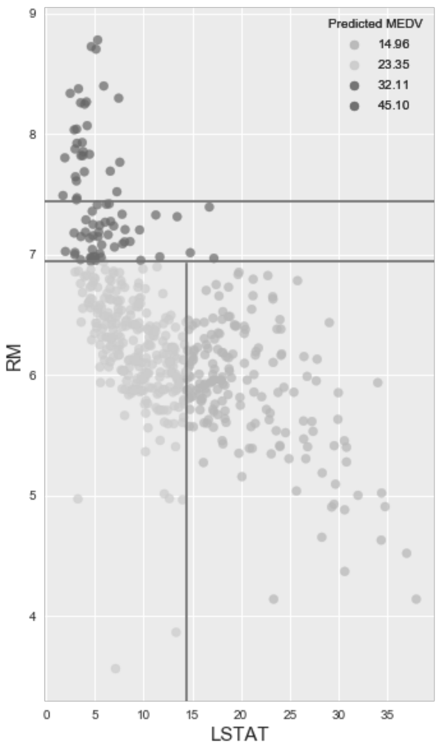


FIGURE 8.21: Regression decision tree (left) with corresponding regions (right).

purity), that must be met for a node to be split. The choice of Δ is a balance between underfitting (if Δ is too high) and overfitting (if it is too low).

2. **Postpruning** grows a maximal tree and then prunes it back, either by replacing a subtree with a node whose class or value is the mode or mean of the constituent observations respectively in the subtree or by replacing a subtree with its most commonly used branch. Postpruning usually leads to better outcomes than prepruning, but is more cost intensive since we have to build out the full tree first.

3. **Cost-complexity pruning** grows a large tree and then finds a subtree T of it that minimizes the cost-complexity criterion

$$\sum_{t=1}^{|T|} |N_t| I(t) + \alpha |T|.$$

   The sum is over terminal nodes only, of which there are $|T|$, a measure of the complexity of the tree. I is the impurity criterion and α is a hyperparameter, which can be determined using cross-validation. This formulation has resemblance to lasso, in that if α is large, we get a small tree, and if it is small, we get a large subtree within the original tree. In particular, when α is zero, we get back the original, large tree.

The popularity of decision trees comes in part because they are inexpensive, non-parametric models that can handle both classification and regression. Furthermore, they are easy to construct and interpret, including explaining their meaning to nontechnical stakeholders. Decision trees implicitly account for interactions between features, support both categorical and quantitative variables, and can even handle missing data. For instance, for categorical data, we can add a category "missing" for each of the predictors. Once a tree of depth d has been constructed, it has $O(d)$ worst case complexity to generate a prediction. Readers interested in experimenting with decision trees can refer to the scikit-learn documentation for the `DecisionTreeClassifier` class, which contains links to examples of how to use it [385].

But decision trees have some weaknesses as well. They do not perform well on certain types of problems related to boolean logic, where building out the tree is analogous to writing out a complete truth table. Because the algorithms given above aspire only to local optimality, if a suboptimal split happens early, it negatively affects the rest of the construction. In domains with many equally important features, it is more effective to use SVMs than decision trees, since decision trees can suffer from a problem called **fragmentation**. This term describes the fact that each leaf of a tree uses only a small subset of the training data, and thus each prediction is based on limited support from the data. Decision trees can also be highly unstable because they are highly dependent on the training data. So a different train-test split can give a very different decision tree for the same dataset.

These disadvantages motivate some of the improvements in the following section, which applies to decision trees and other models as well.

8.10 Ensemble methods

On the British and American game show *Who Wants to be a Millionaire?*, each contestant can use once the privilege of polling the audience for the answer to a multiple choice question on which the contestant may be stumped. The audience usually includes people who want to be on the show, so although they did not qualify for the show that day, it's a safe bet that their guesses are better than random. Polling the audience usually reveals the right answer. Something similar happens in ensemble learning, which combines several models to achieve better performance than any one of the individual models.

One of the disadvantages just listed for decision trees is their high variance (propensity for overfitting). A very successful solution to this problem is to combine multiple trees (i.e., form a "forest"). This is an example of an **ensemble** method. In general, classification accuracy can be improved by aggregating the predictions from multiple classifiers.

When using an ensemble method on a classification problem, we first train a set of **base classifiers** using the training data. In our game show analogy, these are the audience members. We then form a model that predicts by aggregating "votes" from each of the base classifiers.

Under certain conditions, ensemble methods can be much better than the individual classifiers. Let us consider a simple example. Suppose we have 21 **independent** classifiers *each* with an error rate of ϵ and we predict the class based on the *majority* vote. The error of the ensemble method will be given by

$$\epsilon_{\text{ensemble}} = \sum_{i=11}^{21} \binom{21}{i} \epsilon^i (1-\epsilon)^{21-i}.$$

For instance, if $\epsilon = 0.4$, $\epsilon_{\text{ensemble}} = 0.174$ which is much lower than any individual classifier. In Figure 8.22 we see that the ensemble does better than the base classifiers if and only if $\epsilon < 0.5$. The above formulation also requires that the base classifiers not be correlated; while this independence condition may not always be achievable, improvements in error rates can be realized if the correlations are small.

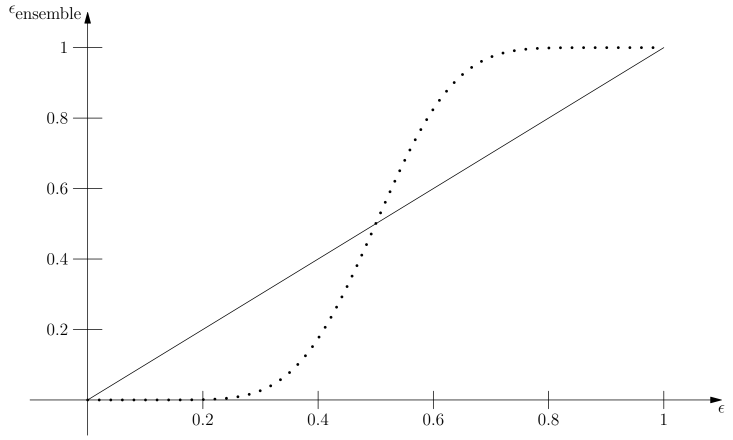


FIGURE 8.22: Ensemble classifier error in an ensemble of 21 independent base classifiers each with error rate ϵ.

8.10.1 Bagging

Decision trees are prone to high variance. We know from statistics that if $X_1, X_2, \ldots, X_n$ are independent random variables with variance σ^2 then their mean $\bar{X}$ has variance only $\frac{\sigma^2}{n}$. So with sufficiently many independent predictors, we can reduce the variance by averaging the predictions. How can we produce these approximately independent and identically distributed decision trees?

One idea would be to train multiple base trees on separate training sets. But we may not have a very large training set, so partitioning the data may thin out the training dataset so much that the trees are not representative of the actual response function. Another approach would be **bootstrapping**, as in Section 8.2.2.

Bagging (**B**ootstrap **Agg**regat**ing**) is a technique of fitting multiple large trees to bootstrapped versions of the training data and then predicting based on the majority vote from these trees. Bagging improves test errors by reducing the variance of the base classifiers. Since only about 63% of the training data is visible to each base classifier, it is also less prone to overfitting due to noise in training data. However, if the primary cause of error in the original classifier were underfitting rather than overfitting, then bagging may in fact lead to worse results, since the base classifiers are now seeing only about 63% of the training data and not the full training dataset.

8.10.2 Random forests

Random forests are a generalization of bagging. In bagging we bootstrapped the training set to build a number of decision trees. While there is randomness in which observations make it to the bootstrapped set, all p features are considered in building the trees. If there is a strong predictor of the response variable in the feature set, it is likely that most trees will split on this predictor. Hence the predictions from these trees, while being identically distributed, will likely be highly correlated. But averaging highly correlated quantities does not reduce the variance as much as it would for uncorrelated quantities.

So to de-correlate the trees and hence reduce the variance, random forests choose a further randomization; each time a tree is split, only m out of p features are considered. If $m = p$ features are used, a random forest is equivalent to bagging. In practice, usually $m \approx \sqrt{p}$ randomly chosen features are used. The smaller the value of m, the more uncorrelated the trees will be. When m is small relative to p, the trees do not have access to a majority of the features. This forces enough trees to rely on other moderately strong predictors and as a consequence, the correlation is reduced. The random selection of variables controls overfitting and for most datasets the results do not seem to be too sensitive to m.

While this gives us an excellent way to reduce the variance while keeping the bias low (if the trees are fully developed), one downside is that we have lost the ease of interpretability that we had when using a single decision tree. We can still rank variables by their importance and gain some insight, but the importance rankings can be much more variable than the classification results themselves.

8.10.3 Boosting

Boosting is one is of the most powerful ensemble methods. It is a general purpose strategy that can be applied to various statistical methods, but we will limit this discussion to boosting decision trees for binary classification. In bagging we built multiple *independent* trees using bootstrapped datasets and finally predicted using majority vote. In boosting, we will instead build a sequence of trees, each new tree informed by the previous tree. While we will use the same model to build the trees, we will adaptively modify the distribution of the training data so that newer trees focus on those observations that were previously classified incorrectly. This increases the weight of misclassified observations so that the tree corrects the bias on those observations. By iteratively using this approach and using a *weighted* combination of these trees, we can create a new, strong classifier.

One of the most well-known implementations of boosting is called **AdaBoost**. Let $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, 2, \ldots, n\}$ denote the training set. For ease of notation, let $y^{(i)} \in \{-1, 1\}$.

Let $w_j^{(i)}$ denote the weight assigned to example $(\mathbf{x}^{(i)}, y^{(i)})$ during the j^{th} round of boosting. We require that the weights in any round sum to 1, that is, $\sum_{i=1}^{n} w_j^{(i)} = 1$. In the first round, we assign each observation in the training set equal weight, $\frac{1}{n}$. Once we have fit a tree C_j to the training data, we determine the weighted classification error on the training data after round j by

$$\epsilon_j = \frac{\sum_{i=1}^n w_j^{(i)} \mathbf{1}_{y^{(i)} \neq C_j(\mathbf{x}^{(i)})}}{\sum_{i=1}^n w_j^{(i)}},$$

where $\mathbf{1}$ is the indicator function introduced earlier. The weights of the training data are updated by

$$w_{j+1}^{(i)} = \frac{w_j^{(i)}}{Z_j} \cdot \begin{cases} e^{-\alpha_j} & \text{if } C_j(\mathbf{x}^{(i)}) = y^{(i)} \\ e^{\alpha_j} & \text{if } C_j(\mathbf{x}^{(i)}) \neq y^{(i)}, \end{cases}$$

where

$$\alpha_j = \frac{1}{2} \ln \left(\frac{1 - \epsilon_j}{\epsilon_j} \right).$$

The normalizing factor Z_j is chosen so that the newly updated weights sum to 1, i.e., $\sum_{i=1}^n w_{j+1}^{(i)} = 1$. We note that if the error rate ϵ_j is close to 1, then α_j is a large negative number and if it is close to 0 it is a large positive number. So the weighting scheme increases the weights of incorrectly labeled training points and decreases the weights of correctly labeled training points. The final classifier after k iterations is given by

$$C = \sum_{j=1}^k \alpha_j C_j.$$

The user must determine the appropriate number of iterations.

Unlike the earlier ensemble techniques using committee voting, where every classifier was given equal weight, in this implementation classifier C_j is given an importance of α_j. If the error rate ϵ_j ever exceeds random guessing, i.e. greater than 50%, the weights $w_j^{(i)}$ are readjusted to $\frac{1}{n}$. This weighted voting process penalizes classifiers with poor accuracies. We leave it to the reader to check that at each round, the weighting scheme assigns the same weight to the correctly classified and misclassified instances. In Figure 8.23, we plot the same data as in Figure 8.17, this time using AdaBoost to separate the classes.

AdaBoost enjoys some good error bounds as well. It has been shown that the training error of the AdaBoost implementation is bounded,

$$\epsilon_{\text{AdaBoost}} \leq \prod_{j=1}^k \sqrt{\epsilon_j (1 - \epsilon_j)},$$

where k is number of iterations. If $\gamma_j = 0.5 - \epsilon_j$, then γ_j measures how much better classifier C_j is to random guessing. Using Taylor series expansion, we can check

$$\epsilon_{\text{AdaBoost}} \leq \prod_{j=1}^k \sqrt{1 - 4\gamma_j^2} \leq e^{-2\sum_j \gamma_j^2}.$$

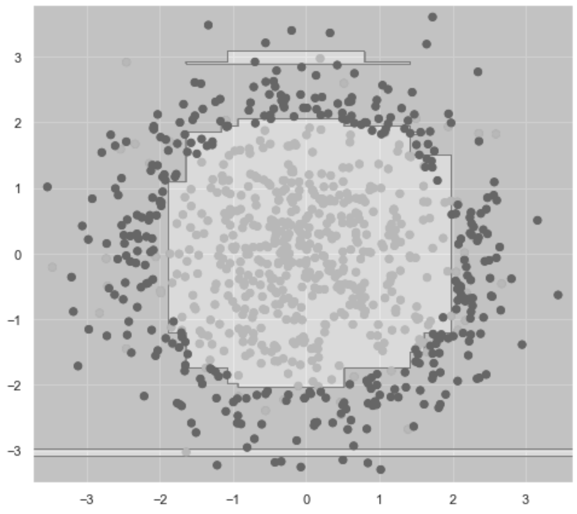


FIGURE 8.23: Results of applying AdaBoost to classify the data shown in Figure 8.17.

So if each C_j is a **weak classifier**, that is, a classifier that is only slightly better than random guessing,[11] and for all j, $\gamma_j > \gamma$, for some $\gamma > 0$, the training error of the AdaBoost classifier decreases exponentially.

8.11 Next steps

Due to the vast scope of the field, we have only addressed some of the most popular techniques in machine learning, and those to only a limited extent. Other popular techniques the reader may wish to learn include linear and quadratic discriminant analysis, k-nearest neighbors (KNN), regression splines, reinforcement learning, and more on kernel methods.

Neural networks are considered a part of machine learning, but because they are both quite complex and quite powerful, this text dedicates an entire chapter to it, immediately following this one. And Chapter 5 already addressed unsupervised learning.

While learning about the wide range of models and analytic techniques, it's important to work on projects. As we've done in each chapter of this text,

[11]For example, a tree of depth 1, sometimes called a **decision stump**, is often a weak classifier.

we highlight the degree to which projects provide invaluable experience that can come only with the specifics of a real dataset and the opportunity to learn the programming skills required to work with it. We suggest the reader try the following activities to get their hands dirty with machine learning.

1. Install both Python and scikit-learn using the instructions online for both (e.g., [385]), then verify that you can reproduce the work in Listing 8.1 and obtain the same output. Must we do anything to ensure that scikit-learn chooses the same test-train split each time we run the code, so that our computation is reproducible?

2. Investigate the documentation for the `LinearRegression` class in scikit-learn to see how to add regularization to your linear model.

3. Using the same dataset, choose a classification problem instead. The easiest example is to let the binary variable CHAS be the target, but this is a strange variable to predict; instead, consider discretizing a variable like CRIM, PTRATIO, or RM. Replace the linear regression step in the code with a logistic regression step instead. How might we report, for the resulting logistic regression model, each of the classifier performance metrics introduced in this chapter? If you apply SVMs instead, do they perform better or worse than logistic regression?

4. Learn how to extend the code from Listing 8.1 to import a dataset from a spreadsheet or CSV file, perhaps that you obtained from the Internet or a database. Use a decision tree or tree-based ensemble method to predict your target variable.

5. Visit Kaggle [242], a website full of machine learning competitions, some for fun and education, others for monetary reward. View their tutorial on how to enter a competition, which is based on the Titanic dataset introduced in this chapter. Once you've completed the tutorial, find another non-tutorial competition and enter!

Chapter 9

Deep Learning

Samuel S. Watson

Brown University

9.1 Introduction

9.1.1 Overview

Many machine learning models learn a relationship between variates in a dataset by introducing a parametric class of functions and identifying the values of the parameters that minimize a specified loss value. For example, logistic regression models the relationship between predictor and response with the function $\mathbf{x} \mapsto \sigma(L(\mathbf{x}))$, where σ is the logistic function and L is a linear function. Support vector machines (Section 8.8) use $\mathbf{x} \mapsto \sigma(L(\mathbf{x}))$, where σ is the sign function.

The core idea of deep learning is to enhance simple parametric models by *composing* several of them. Such compositions are called **neural networks**, in reference to their historical role in efforts to model biological systems. We will call each simple model being composed a **layer**. Composing models leads to enhanced model expressivity, because earlier layers may learn during the training process to output values that are useful for later layers (as in Figure 9.1). This capability has proved to be remarkably useful in solving a variety of real-world problems. However, it took researchers several decades to get the ideas and technology in place for this potential to be realized.

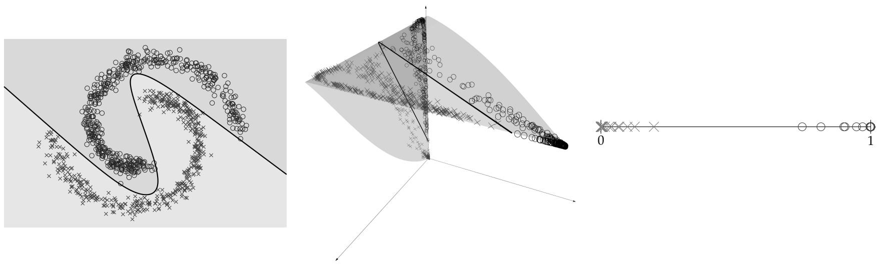


FIGURE 9.1: The response variable for the set of points shown in the first figure is 0 (for points marked with an X) or 1 (for points marked with a circle). A composition of a map from $\mathbb{R}^2$ to $\mathbb{R}^3$ with a map from $\mathbb{R}^3$ to $\mathbb{R}^1$ can map the originally entangled data to a linearly separable configuration in $\mathbb{R}^3$ (second figure) and from there the two classes of data points can be mapped to the intervals $[0, \frac{1}{2})$ and $(\frac{1}{2}, 1]$ (third figure). The preimages of $[0, \frac{1}{2})$ and $(\frac{1}{2}, 1]$ are shown in different shades of gray in each of the first two figures.
The two maps used in this example were of the form $(x_1, x_2) \mapsto (\sigma(ax_1+bx_2+c), \sigma(dx_1+ex_2+f), \sigma(gx_1+hx_2+i))$, where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, and $(x_1, x_2, x_3) \mapsto jx_1+kx_2+lx_3$. The parameters a through l were chosen jointly so that the composition predicts the response variable as well as possible.

9.1.2 History of neural networks

Although interest in deep learning has risen sharply in the last decade or so, its roots go back to the middle of the 20th century. In 1943, Warren McCulloch and Walter Pitts proposed the first mathematical model of the neuron [338]. In this model, each neuron receives signals of various strengths from other neurons and "fires" if and only if its aggregate stimulus is sufficiently large. In other words, the activation of a neuron with input vector $\mathbf{x} \in \mathbb{R}^n$ is defined to be $\mathbf{1}_{\{\mathbf{w}\cdot\mathbf{x}+b>0\}}$ (using the notation from Section 8.5.1), where $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ are parameters of the model called *weight* and *bias*.

Fascinated by the ideas in the McCulloch and Pitts paper, Marvin Minsky set out to build a neural network machine that could learn to escape a maze. In 1951, he worked with Dean Edmonds to build the world's first *neurocomputer* out of valves, motors, gears, and wires [365].

In 1958, Frank Rosenblatt invented the *perceptron algorithm* for finding values for $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ in the activation formula $\mathbf{1}_{\{\mathbf{w}\cdot\mathbf{x}+b>0\}}$ that yield good results on a given dataset [408]. A dedicated computer was built to implement this algorithm, and Rosenblatt hyped the possibilities, saying the machine "may eventually be able to learn, make decisions, and translate languages" [111].

The perceptron proved difficult to train, and a 1969 book by Minsky and Seymour Papert [349] demonstrated some of its theoretical shortcomings. They showed that a perceptron cannot represent the `xor` function $(x, y) \mapsto \mathrm{mod}(x + y, 2)$ on $\{0, 1\}^2$. They also pointed out that Rosenblatt's algorithm does not work on *compositions* of perceptrons, which do not share the perceptron's lack of expressivity. This book was seen by many as confirmation that the optimism about neural networks was ill-founded. Its reception initiated the first *AI winter*, in which many researchers either left the field or found it difficult to get their research published [493].

In his 1974 PhD thesis [497], Paul Werbos introduced the idea of training neural networks by **backpropagation**, which means using the chain rule to apply gradient descent to tune the model parameters (as in Section 9.2.1). However, this idea did not become widely known until it was rediscovered by David Parker and Yann LeCun in 1985 and then rediscovered yet again and popularized in 1986 by David Rumelhart, Geoffrey Hinton, and Ronald Williams in *Learning representations by error propagation* [415]. This paper succeeded in popularizing the backpropagation idea and sparked an era of renewed interest in neural networks.

In 1989, George Cybenko proved that the neural network functions are dense in $C([0, 1]^d)$ [113], and LeCun et al. showed that backpropagation algorithms can leverage this expressivity in practical settings by helping the United States, Postal Service with recognizing handwritten zip code digits [307]. Over the next several years, novel network architectures were used to tackle various natural language and image-related tasks.

By the late 1990s, these endeavors began to lose favor as limitations of existing backpropagation algorithms were encountered. Deeper networks were required for increasingly challenging problems, and these networks were found not to train well. At fault were the repeated multiplications prescribed by the chain rule, which tend to attentuate or amplify the gradient signal exponentially in the number of layers. This **vanishing/exploding gradient** problem makes earlier layers in deep networks particularly difficult to train [423].

In a 1995 paper [308], LeCun et al. compared several models for handwritten digit recognition and showed favorable outcomes for the much simpler support vector machine approach. Random forests (Section 8.10.2), which are very effective and remain quite popular, were introduced the same year [223]. These and other machine learning models became more fashionable than neural nets in the the late 1990s and early 2000s.

Interest in neural networks was renewed in 2006 when Hinton, Simon Osindero, and Yee-Whye Teh published *A fast learning algorithm for deep belief nets* and rebranded the neural networks enterprise as "deep learning" [220]. These researchers developed an effective solution to the vanishing/exploding gradient problem by initializing each layers' weights using an unsupervised learning method (rather than initializing them randomly). In other words, they ameliorated the problem of getting stuck during gradient descent by simply choosing a better starting point in the space of model parameters.

Also in 2006, Fei Fei Li and a team of colleagues began working to assemble a collection of millions of 224×224 color photos. In 2010, this dataset was used to hold a contest called the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), in which researchers compete to develop models that correctly classify images (as well as other image processing tasks) [35]. The winner of the 2012 ILSVRC, called *AlexNet* after its designer Alex Krizhevsky, achieved a breakthrough improvement in classification error rate from about 26% to about 17%. He and his colleagues Ilya Sutskever and Geoffrey Hinton used a larger version of the neural network introduced for handwritten digit recognition two decades prior [280]. Classification error of the participant models continued to shrink substantially over the next few years. In 2015, the ILSVRC winner achieved a better-than-human image classification error rate of 3.57%.

Neural networks have also been applied successfully to natural language tasks in recent years. In 2019, the research organization OpenAI announced a neural network called GPT-2, which can generate responses to essay prompts [399]. The results are very impressive. (Note that this model was trained on a large corpus of data from the Internet, not on *The Lord of the Rings* in particular.)

> *Prompt (human written).* Legolas and Gimli advanced on the orcs, raising their weapons with a harrowing war cry.
>
> *Response (machine-generated, first paragraph).* The orcs' response was a deafening onslaught of claws, claws, and claws; even Elrond

was forced to retreat. "You are in good hands, dwarf," said Gimli, who had been among the first to charge at the orcs; it took only two words before their opponents were reduced to a blood-soaked quagmire, and the dwarf took his first kill of the night. The battle lasted for hours until two of the largest Orcs attempted to overwhelm Aragorn. When they finally stopped, they lay defeated and lifeless for miles and miles.

In addition to its role in image and language processing, deep learning has led to monumental improvements in computer board game play. In 2015, Google Brain's *AlphaGo* engine became the first computer to defeat a professional Go player [435]. In 2019, the Top Chess Engine Championship was won by a neural-network-based engine called *Leela Chess Zero* [96], which is an open-source project based on work published by the Google Brain team.

The pace of achievements in deep learning over the past decade has been unparalleled in other areas of machine learning and has cemented its status as an important resource in the data scientist's toolkit.

9.2 Multilayer perceptrons

In this section, we will develop the ideas necessary to implement a plain-vanilla neural network called the *multilayer perceptron*.

Let p and q be positive integers. We will construct a function from $\mathbb{R}^p$ to $\mathbb{R}^q$ as a composition of simple, parametrized functions called layers. Perhaps the simplest idea would be to compose linear functions, with arbitrary dimensions for the intermediate Euclidean spaces. But compositions of linear functions are linear, so any two consecutive linear layers may be collapsed into one with no change in expressivity, and the set of such compositions is no larger than the set of linear functions from $\mathbb{R}^p$ to $\mathbb{R}^q$.

Thus we will build our composition by inserting a nonlinear function between every pair of linear layers. We will do this in a particularly simple way: we fix a function $K : \mathbb{R} \to \mathbb{R}$, which we call the **activation function**. We will abuse notation and write K also for the function which applies K componentwise to any Euclidean vector.

Recall that an *affine* function is a map of the form $\mathbf{x} \mapsto A\mathbf{x} + \mathbf{b}$ for some matrix A and vector $\mathbf{b}$.

Definition 9.1 *Consider a function $K : \mathbb{R} \to \mathbb{R}$. A **multilayer perceptron** with activation K is a function that may be expressed as an alternating composition of affine functions and componentwise applications of K.*

For concreteness, let's look at a simple example. If $A_1(\mathbf{x}) = \begin{bmatrix} 3 & -2 \\ 1 & 4 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $A_2(\mathbf{x}) = \begin{bmatrix} -4 & 0 \\ 3 & 1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} -2 \\ 2 \end{bmatrix}$, and K is the function that applies $x \mapsto \max(0, x)$ componentwise, then $A_2 \circ K \circ A_1$ is a multilayer perceptron with activation K. If $\mathbf{x} = \begin{bmatrix} -2 \\ -4 \end{bmatrix}$, then we have $A_1(\mathbf{x}) = \begin{bmatrix} 3 \\ -17 \end{bmatrix}$. Applying K to each component yields $\begin{bmatrix} 3 \\ 0 \end{bmatrix}$. Finally, applying A_2 yields $\begin{bmatrix} -4 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} -14 \\ 11 \end{bmatrix}$.

The earliest proposed activation function was the indicator $K(x) = \mathbf{1}_{\{x>0\}}$. However, neural network functions with indicator activation are locally constant, which makes it impossible to incrementally improve them by adjusting the parameters in a direction of local improvement. Sigmoid functions addressed this problem and were the most widely used activation function for many years. More recently, the ReLU (pronounced *RAY-loo*) activation function $x \mapsto \max(x, 0)$ has gained favor because it has been found to perform well in practice and is computationally efficient.

9.2.1 Backpropagation

Consider an L-layer multilayer perceptron $\mathcal{N}$ with activation function K and affine maps $A_j(\mathbf{x}) = W_j\mathbf{x} + \mathbf{b}_j$, for $j = 1, \ldots, L$. Given a set of training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^p$ and $\mathbf{y}_i \in \mathbb{R}^q$ for each $i \in \{1, 2, \ldots, n\}$, we define the **loss function**

$$\mathcal{L}(\mathcal{N}, \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n) = \frac{1}{n} \sum_{i=1}^n L(\mathbf{y}_i, \mathcal{N}(\mathbf{x}_i)),$$

where L is a function that penalizes the descrepancy between an observed value $\mathbf{y}_i$ and the value $\mathcal{N}(\mathbf{x}_i)$ predicted by the model for the corresponding input $\mathbf{x}_i$. One common choice is $L(\mathbf{y}_1, \mathbf{y}_2) = |\mathbf{y}_1 - \mathbf{y}_2|^2$, in which case the loss function measures *mean squared error.*

One reasonable way to search for parameter values that minimize the loss function is to compute the gradient of the loss function with respect to the vector of parameters and adjust the parameters in the direction opposite to that gradient. In other words, we consider $\mathcal{N}$ as a function of $W_1, \mathbf{b}_1, W_2, \mathbf{b}_2, \ldots, W_L, \mathbf{b}_L$ and $\mathbf{x}$, and differentiate

$$\frac{1}{n} \sum_{i=1}^n L(\mathbf{y}_i, \mathcal{N}(\mathbf{x}_i; W_1, \mathbf{b}_1, W_2, \mathbf{b}_2, \ldots, W_L, \mathbf{b}_L))$$

with respect to W_j and $\mathbf{b}_j$ for each $j \in \{1, 2, \ldots, L\}$.

We will use use matrix calculus to keep this process organized. Let's begin by taking derivatives of various types of layers in Figure 9.2.

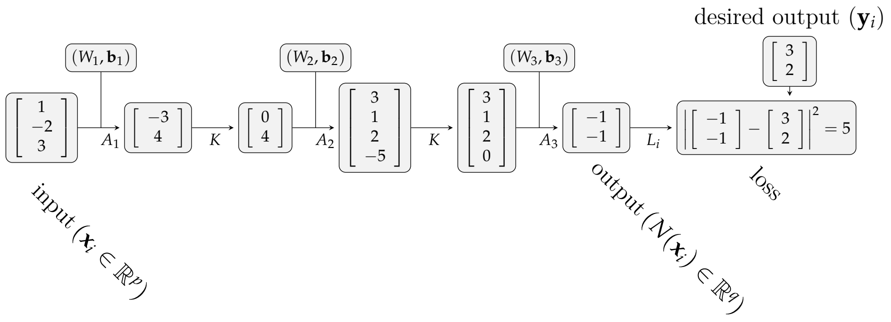

desired output ($\mathbf{y}_i$)
$(W_1, \mathbf{b}_1)$
$(W_2, \mathbf{b}_2)$
$(W_3, \mathbf{b}_3)$
$\begin{bmatrix} 3 \\ 2 \end{bmatrix}$
$\begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$
A_1
$\begin{bmatrix} -3 \\ 4 \end{bmatrix}$
K
$\begin{bmatrix} 0 \\ 4 \end{bmatrix}$
A_2
$\begin{bmatrix} 3 \\ 1 \\ 2 \\ -5 \end{bmatrix}$
K
$\begin{bmatrix} 3 \\ 1 \\ 2 \\ 0 \end{bmatrix}$
A_3
$\begin{bmatrix} -1 \\ -1 \end{bmatrix}$
L_i
$\left\| \begin{bmatrix} -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 3 \\ 2 \end{bmatrix} \right\|^2 = 5$
input ($\mathbf{x}_i \in \mathbb{R}^p$)
output ($\mathcal{N}(\mathbf{x}_i) \in \mathbb{R}^q$)
loss

FIGURE 9.2: Consider a training observation with input $[1, -2, 3]$ and output $[3, 2]$. The multilayer perceptron maps $[1, -2, 3]$ to $[-1, -1]$ through an alternating composition of affine maps $A_j = W_j\mathbf{x} + \mathbf{b}_j$ for $j \in \{1, 2, 3\}$ and ReLU activation functions K. The error associated with this observation is the squared distance from $[-1, -1]$ to $[3, 2]$.

First, the derivative of an affine map $A(\mathbf{u}) = W\mathbf{u} + \mathbf{b}$ with respect to $\mathbf{u}$ is $\frac{\partial (W\mathbf{u})}{\partial \mathbf{u}} + \frac{\partial \mathbf{b}}{\partial \mathbf{u}} = W + 0 = W$.

Second, the derivative of $K(\mathbf{x})$ with respect to $\mathbf{x}$ has $(i, j)^{\text{th}}$ entry $\frac{\partial (K(\mathbf{x})_i)}{\partial x_j}$, which is equal to 0 if $i \neq j$ and $K'(x_i)$ if $i = j$. In other words, the derivative of K with respect to $\mathbf{x}$ is $\operatorname{diag} K'(\mathbf{x})$.

Lastly, the derivative of $L(\mathbf{y}_i, \mathbf{y}) = |\mathbf{y} - \mathbf{y}_i|^2$ is

$$\frac{\partial}{\partial \mathbf{y}} \left[(\mathbf{y} - \mathbf{y}_i)^T (\mathbf{y} - \mathbf{y}_i) \right] = 2(\mathbf{y} - \mathbf{y}_i)^T,$$

by the product rule.

Note that the derivative of $W \mapsto W\mathbf{v}$ is a third-order tensor,[1] since W is a second-order tensor and $W\mathbf{v}$ is a first-order tensor. The following exercise provides the key tool for handling this derivative by showing how it dots with an arbitrary vector $\mathbf{u}$ to give a matrix.

Exercise 9.1 *Given $f : \mathbb{R}^{m \times n} \to \mathbb{R}$, we define*

$$\frac{\partial}{\partial W} f(W) = \begin{bmatrix} \frac{\partial f}{\partial w_{1,1}} & \cdots & \frac{\partial f}{\partial w_{1,n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial w_{m,1}} & \cdots & \frac{\partial f}{\partial w_{m,n}} \end{bmatrix},$$

where $w_{i,j}$ is the entry in the i^{th} row and j^{th} column of W. Suppose that $\mathbf{u}$ is a $1 \times m$ row vector and $\mathbf{v}$ is an $n \times 1$ column vector. Show that

$$\frac{\partial}{\partial W} (\mathbf{u} W \mathbf{v}) = \mathbf{u}^T \mathbf{v}^T.$$

[1] Tensors were first mentioned in Section 3.1. Here, second- and third-order refer to the number of dimensions of the object.

Exercise 9.1 implies that $\frac{\partial}{\partial W}(W\mathbf{v})$ is characterized by the property that $\mathbf{u}\frac{\partial}{\partial W}(W\mathbf{v}) = \mathbf{u}^T\mathbf{v}^T$ for any row vector $\mathbf{u}$.

With these derivatives in hand, we can describe a simple **gradient descent** algorithm for training a multilayer perceptron:

1. **Initialization**. We initialize the weight and bias values in some manner. For example, we could start out with zero weights, or we could sample from a specified probability distribution. We will discuss initialization further in Section 9.3.1.

2. **Forward propagation**. We calculate the loss $L(\mathbf{y}_i, \mathcal{N}(\mathbf{x}_i))$ for each observation $(\mathbf{x}_i, \mathbf{y}_i)$ in the training set (storing the vectors computed at each layer).

3. **Backpropagation**. For each observation, we calculate the derivatives of each layer and multiply them progressively according to the chain rule to find and store the derivative of the final layer, the composition of the last two layers, the composition of the last three layers, and so on. We use these stored derivative values and Exercise 9.1 to compute the derivative of the loss with respect to each weight matrix, and similarly for the bias vectors. Then we compute, for each weight matrix and bias vector, the mean of these derivatives across all observations.

4. **Gradient descent**. We choose a small value $\epsilon > 0$ called the **learning rate** and change the values of each weight matrix by $-\epsilon$ times the mean derivative of the loss with respect to that matrix. In other words, we adjust the weights in the direction which induces the most rapid instantaneous decrease in the loss value. Likewise, we adjust each bias vector by $-\epsilon$ times the mean derivative of the loss with respect to that vector.

We repeat these steps until a specified stopping condition is met. For example, we might stop when the computed derivatives get sufficiently small (suggesting that a local minimum has been reached), or we might keep track of the loss computed with respect to a withheld subset (called **validation data**) of the original data. We would stop when the updates are no longer significantly decreasing the validation loss.

In practice, the algorithm described above is computationally infeasible because the training set is often quite large (tens of thousands of observations, say), and computing all of the necessary derivatives for each weight update makes the whole process impractically slow. Therefore, it is customary to instead do **stochastic gradient descent**, wherein derivatives are calculated only with respect to a randomly sampled subset of the original training set, and weight updates are performed based on an average over this **mini-batch**. Commonly used values for the size of the mini-batch range from 30 to 200 or so.

Exercise 9.2 *Visit the website for this text [84] and work through the computational notebook there to experiment with a multilayer perceptron implemented from scratch.*

9.2.2 Neurons

Multilayer perceptrons are often depicted with each vector component drawn in its own node, as shown in Figure 9.3. The nodes in such a diagram are called **neurons**, and it's customary to depict the activation operations as internal to each neuron. Thus the neurons have input and output values, drawn as pill halves in Figure 9.3. The arrows correspond to entries of the W matrices, while the components of the bias vectors are not shown. Note that the final layer uses identity activation.

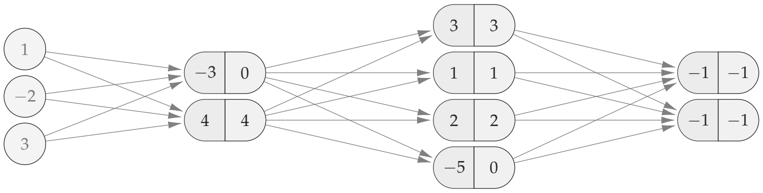


FIGURE 9.3: A component-style multilayer perceptron diagram.

9.2.3 Neural networks for classification

So far we have considered multilayer perceptrons that output *vectors*. Many important neural network applications call for the network to return a *classification* instead. For example, we might want to build a neural network that takes as input a vector in $[0, 1]^{784}$ representing the pixel values for a 28×28 grayscale image depicting a handwritten digit and outputs a value in $\{0, 1, 2, \ldots, 9\}$ that corresponds to the digit that appears in the image. Let's denote by C the number of classes, in that case 10.

The customary approach to adapting neural networks to classifiation problems is to encode each sample classification $\mathbf{y}_i$ as a vector in $\mathbb{R}^C$ with a 1 in the position corresponding to its class and with zeros elsewhere. We set up the network to output vectors in $\mathbb{R}^C$ and then apply the **softmax** function

$$\mathbf{u} \mapsto \left[\frac{\mathrm{e}^{u_j}}{\sum_{k=1}^{C} \mathrm{e}^{u_k}} \right]_{j=1}^{C}$$

from $\mathbb{R}^C$ to $[0, 1]^C$. Applying the softmax ensures that the entries of the prediction vector are positive and sum to 1. This allows us to interpret components of $\mathbf{y}$ as asserted probabilities for each possible classification. A confident prediction assigns a number close to 1 in one position and small numbers elsewhere.

Finally, we perform stochastic gradient descent using the loss functions $L(\mathbf{y}_i, \mathbf{y}) = -\log(\mathbf{y} \cdot \mathbf{y}_i)$. This function returns zero if the output vector asserts a probability of 1 to the correct class, and it applies a large penalty if the $\mathbf{y}$ assigns a small probability to the correct class.

Training a network for classification requires differentiating the loss-softmax composition:

Example 1 *Find the derivative of* $L(\mathbf{y}_i, \text{softmax}(\mathbf{u}))$ *with respect to* $\mathbf{u}$*.*

Solution. Suppose that j is the correct class for the i^{th} observation, that is, $\mathbf{y}_{ij} = 1$. Then

$$L(\mathbf{y}_i, \text{softmax}(\mathbf{u})) = -\log\left(\frac{\mathrm{e}^{u_j}}{\sum_{k=1}^{n} \mathrm{e}^{u_k}}\right) = -u_j + \log \sum_{k=1}^{n} \mathrm{e}^{u_k}.$$

Differentiating with respect to u_j yields

$$-1 + \frac{\mathrm{e}^{u_j}}{\sum_{k=1}^{n} \mathrm{e}^{u_k}},$$

and differentiating with respect to u_ℓ for $\ell \neq j$ yields

$$\frac{\mathrm{e}^{u_\ell}}{\sum_{k=1}^{n} \mathrm{e}^{u_k}}.$$

Therefore, the derivative with respect to $\mathbf{u}$ is $-\mathbf{y}_i^T + \text{softmax}(\mathbf{u})^T$. $\square$

Exercise 9.3 *Think of two advantages of outputting probability vectors rather than discrete class values (solution on page 440).*

9.3 Training techniques

Training neural networks to perform well on real-world problems is seldom straightforward. The loss is a non-convex function of the model parameters, so there is no guarantee that a gradient descent algorithm will converge to a global minimum. Furthermore, even if the optimization algorithm does achieve very low training loss, the resulting model may be overfit (as in Figure 9.4). In this section, we discuss several techniques for addressing these challenges.

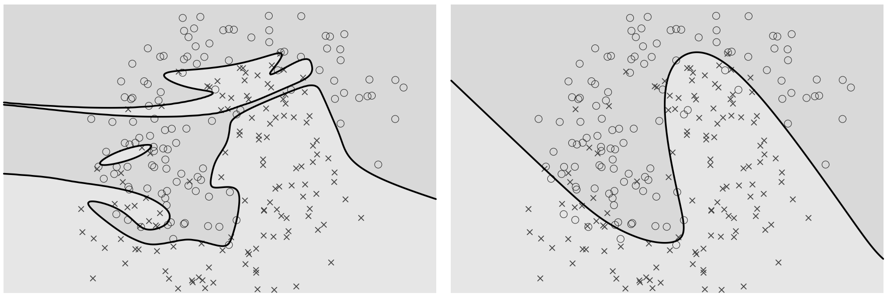


FIGURE 9.4: A neural network can overfit the training data, as shown in the first figure. The model in the second figure has a higher loss on the training set, but it performs better on test data.

9.3.1 Initialization

Effective network training requires starting the weights and biases in a part of the parameter space where gradient descent can work well. If many of the activation functions map their inputs to zero, for example, many weight and bias values will have zero marginal effect on the loss and may therefore never change. Similarly, learning is hindered if the parameters start out orders of magnitude larger or smaller than where they should end up. The *Xavier-He initialization* scheme specifies that the initial value of the weights in each layer should be drawn independently from a Gaussian distribution with mean zero and variance

$$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}},$$

where n_{inputs} and n_{outputs} are the dimensions of the domain and codomain for that layer. This formula is obtained by compromising an interest in achieving constant variance of the values output by each layer (which suggests a standard deviation of $1/\sqrt{n_{inputs}}$) and constant variance of the backpropagated gradients (which suggests $1/\sqrt{n_{outputs}}$) [190].

Xavier-He initialization is much simpler than the approach of Hinton, Osindero, and Teh mentioned in Section 9.1.2 [220], but it does tend to work well in practice and has become a popular default initilization scheme.

Initializing bias values is more straightforward; the popular deep learning framework Keras (see Section 9.7.2) uses zero vectors. In [194], it is recommended to initialize each bias vector entry to a fixed, small positive number like 0.1.

9.3.2 Optimization algorithms

There are many numerical optimization algorithms which aim to improve on stochastic gradient descent. For example, **momentum gradient descent**

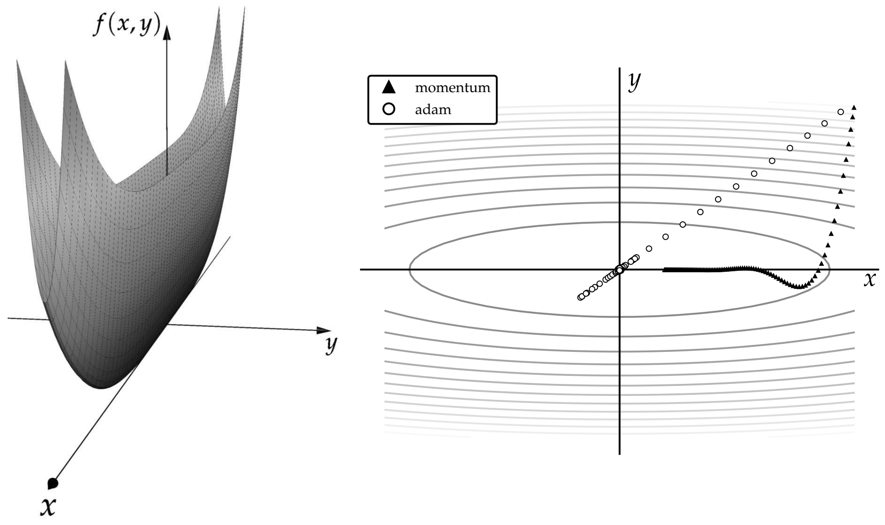


FIGURE 9.5: A function f exhibiting pathological curvature (left), and a comparison of momentum gradient descent and Adam for that function (right), with the same starting point and learning rate.

takes steps based on the backpropagated derivative computed from the current mini-batch *as well as* the derivatives computed in previous mini-batches. Specifically, we choose a value $\alpha \in [0, 1]$ called the **momentum** and take a weighted average of the gradients computed at each iteration: the current mini-batch gets a weight of 1, the mini-batch immediately preceding the current one gets a weight of α, the one before that gets a weight of α^2, and so on. Momentum gradient descent produces more stable, less responsive derivative estimates than plain stochastic gradient descent. We can adjust α to navigate the tradeoff between stability and responsiveness.

Adam optimization combines momentum with another idea: adjusting each component of each gradient descent step based on the average squared gradient seen in that component over the previous gradient descent iterations. In other words, the step size for a given parameter gets a boost if the loss function tends, on average across iterations, to be insensitive to small changes in that parameter. This allows the iterates to proceed more directly toward the minimum when in a region where the function exhibits *pathological curvature* (where the function's second derivative in one direction is much larger than the second derivative in another direction, as in Figure 9.5).

Both momentum gradient descent and Adam are common optimizers to use when training a neural network, and they are more popular than plain stochastic gradient descent.

9.3.3 Dropout

Dropout is a regularization technique which entails making a random subset of the neurons in a given layer inactive for a mini-batch (this is equivalent to assigning those neurons an activation function of zero for that mini-batch). Each neuron in a dropout layer is inactivated with a specified probability α, independently of other units and independently of the same neuron's status in other mini-batches. Dropout reduces overfitting by preventing the neural network from relying too heavily on specific pathways or individual neurons. It has been a popular technique since its introduction in 2014 [441], because it is simple and often gives a modest improvement in model results.

9.3.4 Batch normalization

Feature scaling is a common feature engineering technique useful for avoiding difficulties associated with large differences in scale between features. **Batch normalization** grants neural networks the same convenience on a neuron-by-neuron basis: it modifies the output from each neuron by standardizing its output values across the current mini-batch. Furthermore, rather than standardizing to zero mean and unit variance, we introduce two new trainable parameters per neuron—commonly denoted β and γ—which specify the mean and standard deviation of the activation values which are output for the next layer. Mathematically, the batch normalization update takes the following form (with $\mathbf{x}$ representing the vector of activation values at a specific neuron across the current mini-batch):

$$\begin{aligned}\hat{\mu} &= \operatorname{mean}(\mathbf{x})\\ \hat{\sigma}^2 &= \operatorname{std}(\mathbf{x})\\ \mathbf{x}_{\text{out}} &= \beta + \frac{\gamma(\mathbf{x}-\hat{\mu})}{\sqrt{\hat{\sigma}^2+\epsilon}},\end{aligned}$$

where mean and std compute the sample mean and sample standard deviation, respectively (and where the addition of β and the subtraction $\mathbf{x}-\hat{\mu}$ are componentwise).

For example, if we have 6 neurons in a given layer and a mini-batch size of 100, then we will have 6 such vectors $\mathbf{x}$ in $\mathbb{R}^{100}$ for that layer. Also, ϵ is a *smoothing term* used to avoid division by zero in the event that $\hat{\sigma}$ is equal to zero. A typical choice is $\epsilon = 10^{-5}$.

9.3.5 Weight regularization

Another way to address the overfitting problem is to add a term to the loss function that penalizes the model for using large weights. For ℓ^2 regularization, we add a constant multiple of the sum of the squares of the parameters in the weight matrices, and for ℓ^1 regularization we add a constant multiple of the sum of their absolute values. Since the derivative of $\mathbf{w} \mapsto \lambda|\mathbf{w}|^2$ is $2\lambda\mathbf{w}^T$, ℓ^2 regularization has the effect of changing the parameter update rule from $\mathbf{w} \leftarrow \mathbf{w} - \epsilon\frac{\partial L}{\partial \mathbf{w}}$ to $\mathbf{w} \leftarrow \mathbf{w}(1 - 2\epsilon\lambda) - \epsilon\frac{\partial L}{\partial \mathbf{w}}$, where ϵ is the learning rate and L is the loss without the weight regularization term. In other words, ℓ^2 regularization scales each parameter value towards zero by a factor of $1 - 2\epsilon\lambda$ on each iteration.

9.3.6 Early stopping

Training a neural network for too long can lead to overfitting. *Early stopping* entails holding out a portion of the training data—called **validation** data[2]—and halting the training process once the loss or accuracy on the validation set stops improving. For example, we might stop when we reach a prespecified number of mini-batches after the last time a record minimum loss was reached (and then revert the network parameters to their values at the time of the last record minimum).

9.4 Convolutional neural networks

Multilayer perceptrons represent a one-size-fits-all approach to model fitting. Although they do have knobs for the user to adjust (depth, architecture, activation function, optimization algorithm, etc.), they do not support the incorporation of *domain-specific knowledge* into the structure of the model. There are two especially important areas where domain-tailored model architectures have proved to be very useful: *image processing* and *natural language processing.*

In this section, we will discuss **convolutional neural networks**, which are designed for handling image data. Common problems tackled with convolutional neural networks include image recognition (e.g, "what type of animal is represented in this image?") and object segmentation (e.g., "where are the faces in this picture?").

Like multilayer perceptrons, convolutional neural networks (or *convnets*) are alternating compositions of linear and nonlinear maps. The idea is that the output of each layer should store features of the image, starting with the raw image pixels, and increase in utility for the task at hand. (See Figure 9.6.)

[2]Note that since validation data is not used to compute gradients as the model is training, it is reasonable to use it to check whether the model is overfitting. However, validation data is distinct from *test* data since we *are* using the validation data as a part of the overall training process.

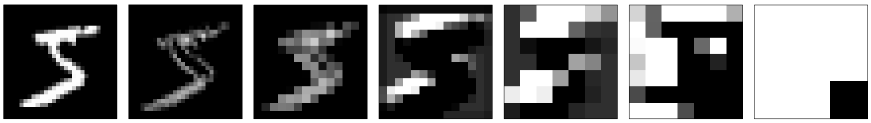


FIGURE 9.6: An image representing a handwritten digit followed by partial outputs of the layers of a convolution neural net trained to identify digit images. Each layer output consists of a *stack* of two-dimensional arrays, and this figure shows only one of them per layer.

Convnets preserve the two-dimensional structure of the image data, however, and set up connections which reflect spatial relationships between pixels.

It turns out to be useful to represent layer inputs and outputs in a convnet as *three*-dimensional arrays, for two reasons: First, images may have different *channels* (like separate red, green, and blue values for each pixel, if the image is in color) that are naturally represented with a shallow third axis. Second, equipping layers to map a stack of images to a stack of images allows them to track several image features simultaneously and recombine them in different ways in each successive layer. This additional expressivity turns out to be critical for good performance on real-world problems.

9.4.1 Convnet layers

Convolutional neural networks are built using two main types of layers: *convolutional* and *pooling* layers. Images are customarily assumed to be square, with the idea that rectangular images encountered in practice may be resampled to square dimensions.

Convolutional layer. Use Figure 9.7 to follow along with the following description of a convolutional layer computation. Suppose that the layer's input is a $k \times k \times \ell$ array, and fix an integer $s \leq k$. The output of the layer will be an array whose first two dimensions correspond to the grid of contiguous $s \times s$ square subgrids of a $k \times k$ square grid and whose depth t is chosen as part of the network architecture. Each channel in the output array is associated with an $s \times s \times \ell$ array called a *filter* together with a scalar b called the *bias*. The filter entries and biases are parameters to be learned during training.

We define the dot product of two arrays of the same shape to be the sum of the products of corresponding entries. Each entry in a given output channel is obtained by dotting that channel's filter with the corresponding $s \times s \times \ell$ subarray of the input and adding that channel's bias to the result.

Pooling layer. Given an input tensor of dimension $k \times k \times \ell$ and an integer $p \geq 0$, a pooling layer applies a fixed scalar-valued function to each $p \times p$ subgrid in a partition of each channel in the previous layer. Thus the output of the pooling layer is $(k/p) \times (k/p) \times \ell$, if p divides k. The most common choice for p is 2, and the most common choice for the nonlinear

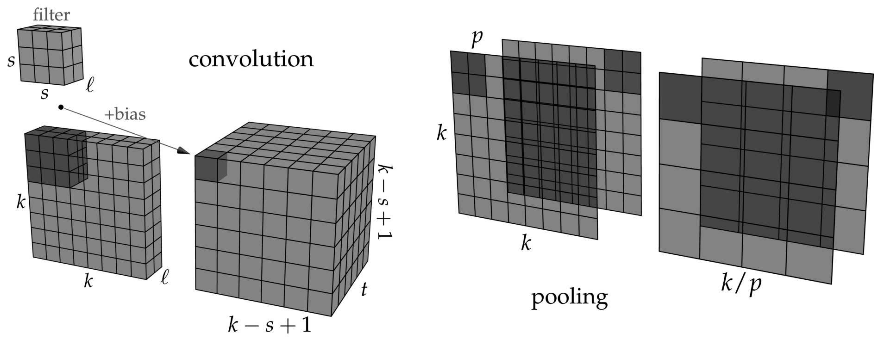


FIGURE 9.7: (Left) An illustration of the convolution operation: each channel in the output tensor is associated with a trainable filter (the $s \times s \times \ell$ tensor) and a trainable scalar bias value. The values in the output tensor are determined by the input tensor, filter, and bias by dotting the filter with each continguous $s \times s \times \ell$ input subtensor and adding the bias. (Right) An illustration of the pooling operation: input and output tensors have the same number of channels, and each output channel is obtained from the corresponding input channel by applying a specific function to the entries in each square in a partition of the channel.

function applied to each subgrid is $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \mapsto \max(a, b, c, d)$. A pooling layer which uses the maximum function is called a **max pool** layer. Note that pooling layers do not have any model parameters.

Exercise 9.4 *Explain how a convolutional neural network without pooling layers may be viewed as a multilayer perceptron with a set of equations imposing constraints on the model parameters (solution on page 440).*

9.4.2 Convolutional architectures for ImageNet

Given an image recognition task, an ordinary convolutional neural network like the one shown in Figure 9.8 typically performs significantly better than a multilayer perceptron with a similar number of neurons. However, achieving state-of-the-art performance has required substantial innovation beyond convolutional and pooling layers. We will use the history of the ImageNet contest (ILSVRC) as a focal point for surveying how cutting-edge performance has been achieved over the period of renewed interest in convnets from 2012 to the present.

AlexNet. The first deep learning classifier to win the ILSVRC was **AlexNet** in 2012 [280]. See Figure 9.9 for the representation of the network architecture from the original paper.

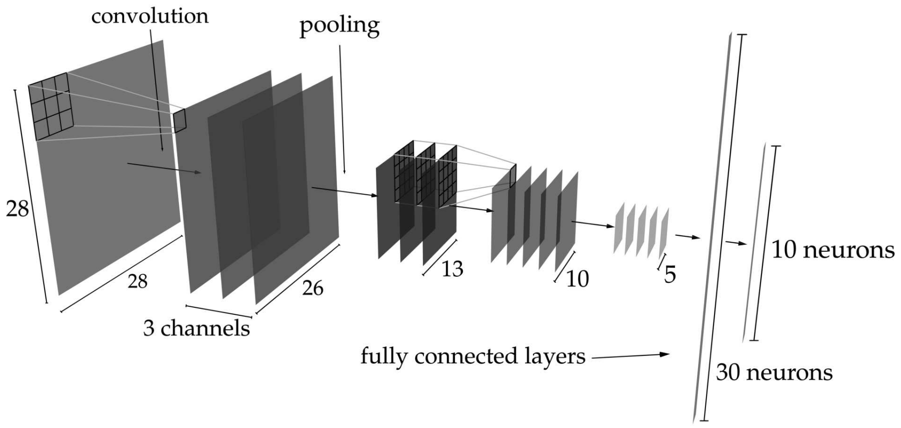


FIGURE 9.8: A convolutional neural network architecture. We begin with a convolutional layer followed by a pooling layer, then another convolutional layer and another pooling layer. Finally, the data go through two fully connected layers to yield an output vector in $\mathbb{R}^{10}$.

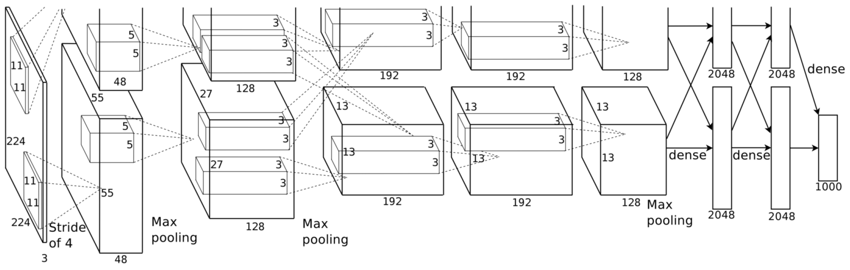


FIGURE 9.9: The AlexNet architecture. The *dense* layers are multilayer perceptron layers. From [280], used with permission.

Note the **stride** value of 4 shown in the first layer. This means that the 11×11 filter traverses the grid in increments of 4 pixels rather than 1 pixel. This results in each spatial dimension in the next layer's channels being approximately one-fourth as large as they would be with a stride of 1 pixel. This stride value reflects an effort to keep the model's overall number of parameters manageable.

The convolutional layers are shown on two parallel tracks. This represents an accommodation of a practical constraint: a single processing unit did not have enough memory to store the necessary data, so the layers were split across two.

Several training techniques were used, including ℓ^2 regularization with a parameter of $\lambda = 5 \times 10^{-4}$ and dropout with $\alpha = 1/2$. The mini-batch size was 128, and the momentum used in the gradient descent algorithm was 0.8. The convolutional and dense layers use ReLU activations, and the last layer uses a softmax activation. AlexNet also used a local *response normalization* scheme that fixes an ordering on the channels and then applies a standardization (de-meaning and dividing by the standard deviation) across several nearby channels in the same spatial position. This practice has fallen out of favor in recent years.

The winner of the ILSVRC in 2013 was a modification of AlexNet called *ZFNet*.

GoogLeNet. The winner of the 2014 contest was *GoogLeNet* (architecture illustrated in Figure 9.10).

This network introduces a number of innovations. Perhaps its most salient novelty is the use of the **inception module** as a building block. See Figure 9.11 for an illustration.

The idea of the inception module is to allow the model to effectively choose for itself the best spatial dimension for the spatial dimension of the filter used in each layer. We do this by inserting layers of different filter dimensions and concatenating the results channel-wise (which requires some padding around the edges so that the convolution operations don't actually shrink the array dimensions). We also insert some 1×1 convolutions and a max pooling layer, because otherwise the concatenation step causes the number of channels (and therefore also the number of parameters) to get out of hand.

A second new feature of GoogLeNet is the insertion of two additional classification stacks, each consisting of an average pool layer followed by one or two dense layers and a softmax activation. They attach directly to the outputs from intermediate layers, and their outputs are used as a part of the training loss function. The purpose of these stacks is to ensure that the early layers receive sufficiently large gradient nudges to properly train. Although this would seem to impose an unnatural requirement on early layers (namely, that they should produce a representation of the images which supports a classification stack which will not be used at prediction time), this feature was nevertheless found to be worthwhile.

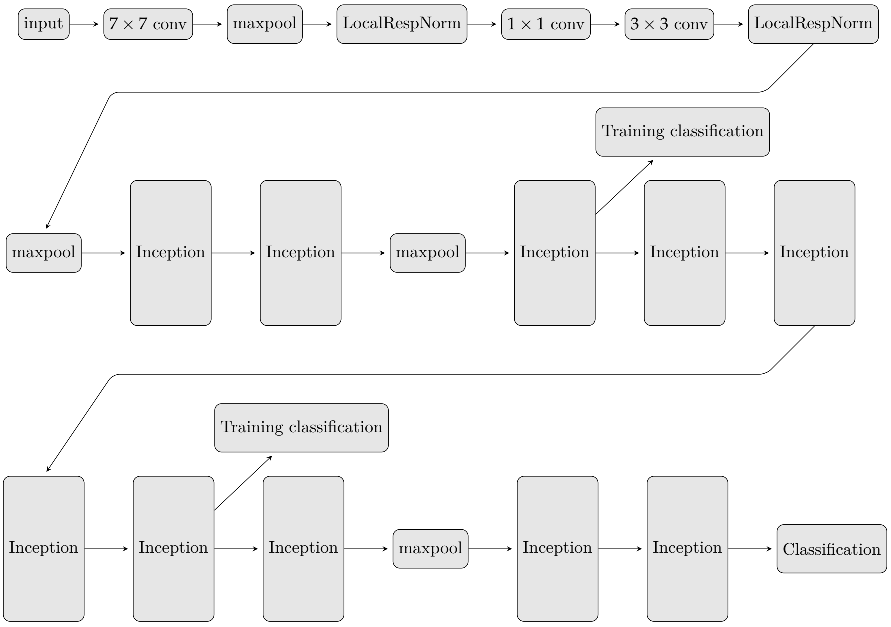


FIGURE 9.10: GoogLeNet, also known as InceptionNet.

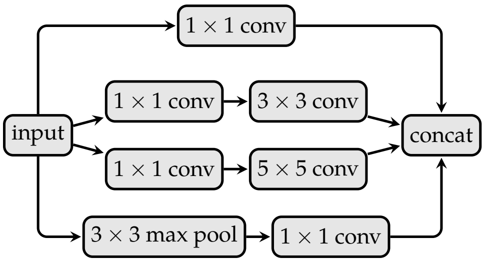


FIGURE 9.11: An inception module.

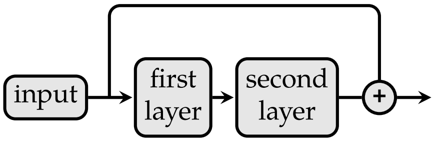


FIGURE 9.12: A ResNet unit.

Lastly, GoogLeNet decreased the overall number of parameters (from about 62 million to about 11 million) and achieved better balance than AlexNet in the distribution of the parameter count across layers. AlexNet has most of its parameters in the last couple of layers, since large dense layers tend to be much more parameter-intensive than convolutional layers. The dense layers in GoogLeNet were made smaller than the ones in AlexNet by judicious use of pooling layers and 1×1 convolutional layers.

ResNet. The winner of the 2015 contest was **ResNet**. It accelerated the trend of increased network depth by using 152 layers! Despite the number of layers, ResNet's overall number of parameters was similar to AlexNet, at 60 million.

In principle, a deeper network should perform better than a shallower one, because it can choose to approximate the identity function for any layers it doesn't need, thereby replicating the behavior of a shallower network. However, in practice this doesn't work because training difficulty increases in the number of layers.

These observations lead to the key insight of ResNet: we can make it easier for the network to learn the identity function by interpreting the layer outputs as specifying the *difference* between input and output rather than specifying the output directly. In other words, we sum a layer's input to its output and forward the result to the next layer. This is the idea that gives ResNet its name, since a *residual* is a difference between two quantities.

We can illustrate this idea in a computational graph diagram by drawing a connection that goes around a computational unit (like a neural network layer, or a stack of two neural network layers). See Figure 9.12.

Further details about how ResNet is trained include the following.

1. Batch normalization is used after every convolutional layer.
2. Weights are initialized using the Xavier-He heuristic.
3. The learning rate decreases over time: it starts at 0.1 and decays by a factor of 10 each time a performance plateau is detected.
4. Mini-batch size is 256.
5. *No* dropout is used.

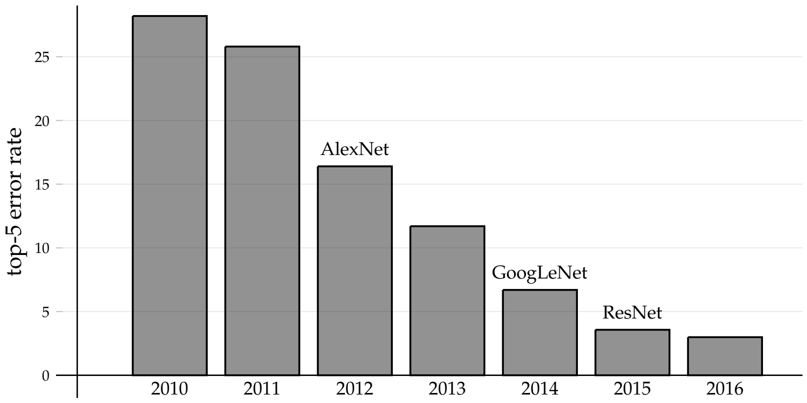


FIGURE 9.13: ILSVRC results: the *top-five error rate* is the percentage of images in the test set whose correct classification does not appear among the five classes asserted by the model as most likely.

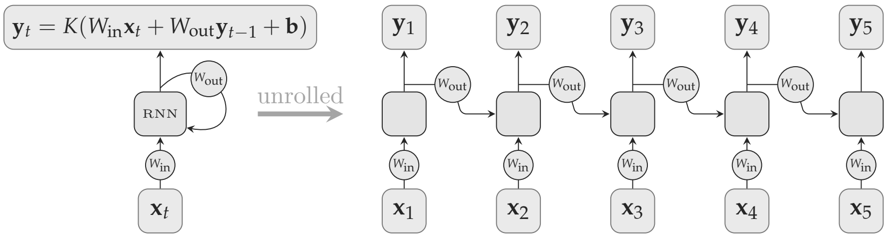
In 2016, Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi introduced *Inception-ResNet*, which uses inception modules with residual connections. This model was used as a part of the 2016 winning submission from the *Trimps-Soushen* team. See Figure 9.13 for a comparison of the performance of all of the ILSVRC winners. The last couple of winners match human-level performance on the metric shown.

9.5 Recurrent neural networks

MLPs and convnets are *feedforward* neural networks: their computational graphs are acyclic. Such networks are not ideal for sequential data (like text or audio) because they don't have a natural mechanism for modeling the sequential structure of the data. *Recurrent* neural networks remedy this problem by introducing loops between nodes.

The simplest recurrent neural network (Figure 9.14) consists of a single recurrent **cell**. The input data are fed into the cell sequentially, and the cell also outputs a sequence. For example, suppose that $\mathbf{x}_t$ is a vector representation of the t^{th} word in a sentence to be translated, and that $\mathbf{y}_t$ represents the t^{th} word in the translation. The output at each time step depends on the input at that time step and the output at the previous time step. Mathematically, we have

$$\mathbf{y}_t = K(W_{\text{in}}\mathbf{x}_t + W_{\text{out}}\mathbf{y}_{t-1} + \mathbf{b}),$$



$\mathbf{y}_t = K(W_{\text{in}}\mathbf{x}_t + W_{\text{out}}\mathbf{y}_{t-1} + \mathbf{b})$
RNN
W_{out}
W_{in}
$\mathbf{x}_t$
unrolled
$\mathbf{y}_1$ $\mathbf{y}_2$ $\mathbf{y}_3$ $\mathbf{y}_4$ $\mathbf{y}_5$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ $\mathbf{x}_5$

FIGURE 9.14: A simple recurrent neural network. We can *unroll* a recurrent neural network by distinguishing time steps spatially in the diagram.

where K is the cell's activation function, W_{in} and W_{out} are matrices, $\mathbf{b}$ is a vector, $\mathbf{y}_t$ is the model's output at time t, and $\mathbf{x}_t$ is the input data at time t.

Note that a recurrent neural network can handle a variety of input lengths, since we can keep putting the next $\mathbf{x}_t$ in and getting a new $\mathbf{y}_t$ value out until we reach the end of the sequence of $\mathbf{x}_t$'s.

9.5.1 LSTM cells

Simple recurrent cells are seldom used in practice because they don't effectively learn relationships involving long-term memory. For example, if the main character in a story is revealed to be a former tennis prodigy, that information might very well be crucial for correctly interpreting a sentence many paragraphs later. The **long short-term memory** (LSTM) cell was invented in 1997 by Sepp Hochreiter and Jürgenas Schmidhuber as a way of providing the recurrent neural network with a dedicated mechanism for retaining and modifying long-term state [224]. There are many variants of the LSTM architecture, a typical example of which is shown in Figure 9.15.

Each cell outputs two values to the next cell: the long-term state C_t (the top line in Figure 9.15) and the current timestep's output value $\mathbf{y}_t$. The cell takes C_{t-1} and y_{t-1} as input, as well as the current input vector $\mathbf{x}_t$. Three dense layers are used to modify the information in the long-term state, the first by componentwise multiplication and the second and third by vector addition. A fourth dense layer and the long-term state combine multiplicatively to produce the current output value $\mathbf{y}_t$.

Recurrent networks based on LSTM cells have been popular since their introduction and continue to be used in household natural language systems like Amazon's Alexa [204].

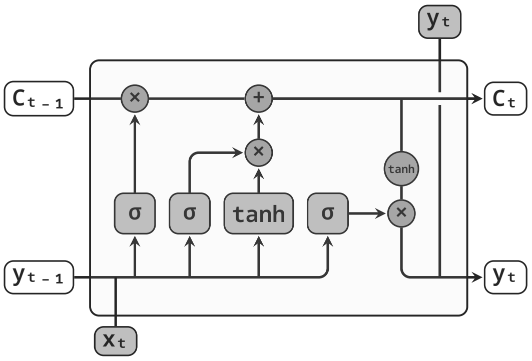


FIGURE 9.15: An LSTM cell. The top line carries the long-term information, and the bottom line produces each time step's output value y_t. The vectors entering on the left side of the cell are the corresponding outputs from the previous cell. Unannotated merging of lines represents vector concatenation, merging annotated with a circle represents combining the two incoming vectors with the operation shown, a circle represents function application, line splitting corresponds to vector copying, and the boxes marked σ or `tanh` represent dense neural network layers with trainable weights and an activation function given by the label.

9.6 Transformers

9.6.1 Overview

In recent years, state-of-the-art natural language results have been obtained using a type of feedforward neural network called a **transformer**, introduced in a paper called *Attention is all you need* [485]. For example, the network GPT-2 discussed in Section 9.1.2 is built using this idea. For concreteness, let's discuss the translation problem: given a sentence in one language, we are looking to output a translation of that sentence in another language.

We will discuss the high-level structure of a transformer before describing the function computed by each component. The two stacks in Figure 9.16 are called the **encoder** and **decoder**, respectively. The input to the encoder is a sequence of words encoded as a matrix (where the i^{th} row contains a vector representation of the i^{th} word in the input sentence, for each i), and the output of the encoder is a digested form of the input sentence whose purpose is to play a role in the decoder's computation. Conceptually, the encoder output corresponds to the abstract state stored in a translator's mind when they have comprehended a passage in the source language and are about to begin writing a translation.

The input ingredients for the decoder stack are the output from the encoder stack and a matrix of vector representations of the (initially empty) list of output symbols that have been generated thus far. The decoder's *last* layer is a dense layer with softmax activation, and it produces a probability distribution over the vocabulary of the target language. A word, which will serve as the next word in the translation, is chosen from that distribution. The decoder stack is executed again to generate another word, and so on, until eventually it outputs a special `end` symbol.

9.6.2 Attention layers

The key component of a transformer is **attention**. The idea of the attention mechanism is that particular words are operative at different times during a translation task. For example, when translating `¿Cómo estás?` from Spanish to English, we're looking mostly at the word `Cómo` when we output `how`, then mostly at `estás` when we output `are`, and again at `estás` when we output `you`. Attention is an architectural component which mimics this ability to focus on different parts of the input sequence over time. Although self-attention layers appear before attention layers in the transformer architecture, we will discuss attention layers first because that will make it easier to draw analogies.

To define the computation performed by an attention layer, we fix a positive integer h and associate to each attention layer $3h+1$ trainable matrices, denoted (W_i^Q, W_i^K, W_i^V), where i ranges from 1 to h, and W^O. We say that the layer has h **heads**. The motivating concept for the heads is that they are focusing on different aspects of the context, analogous to the way that human readers keep track of setting, weather, mood, etc.[3]

We define the function

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V,$$

where d is the the number of columns of the matrix Q and where softmax acts row-wise. The rows of the three arguments of this function are referred to as *queries*, *keys*, and *values*. Recalling from linear algebra that that the rows of a matrix product AB are linear combinations of the rows of B with weights given by corresponding rows of A, we can see that the Attention function recombines the values based on the alignment of queries and keys (as computed by the dot products implicit in the matrix multiplication QK^T). The application of the softmax function ensures that the weights used to recombine the values sum to 1.

For each $i \in \{1, 2, \ldots, h\}$, we define

$$\text{head}_i = \text{Attention}(XW_i^Q, ZW_i^K, ZW_i^V),$$

[3]All analogies between neural network behavior and the human mind should be taken with a large grain of salt. The analogies in this section are particularly strained because we think of matrix rows as corresponding to words, even though that interpretation is only valid for the original input.

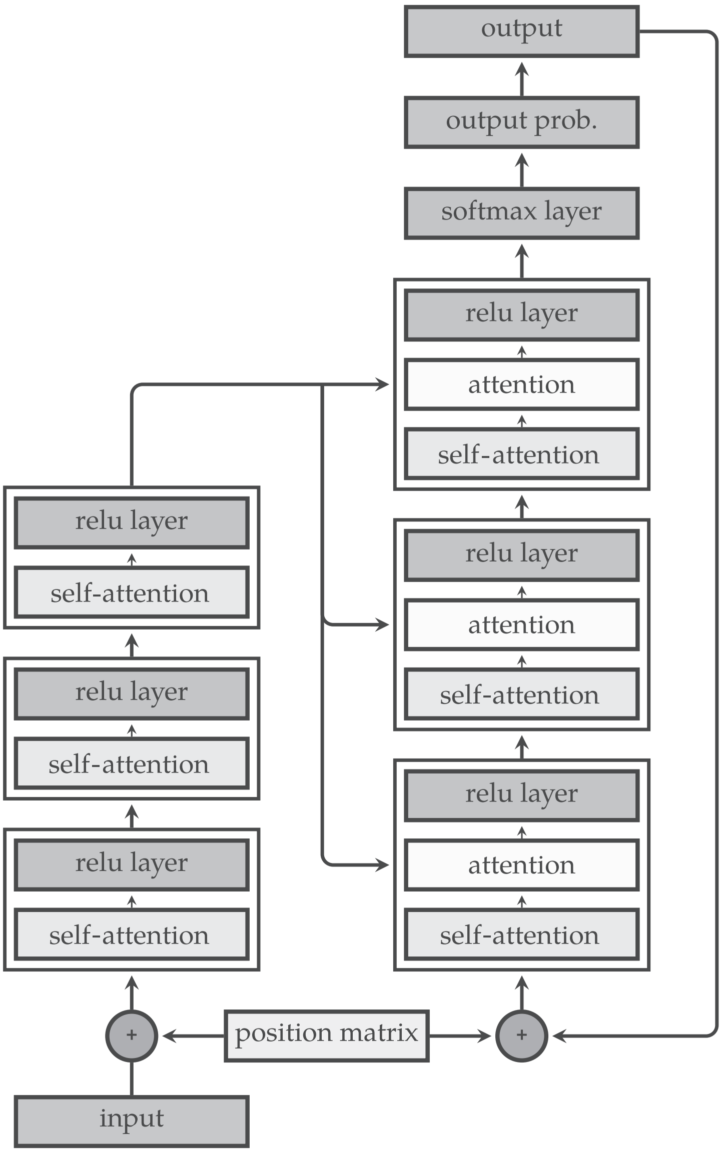


FIGURE 9.16: Architecture of a transformer.

where X is the input matrix for the layer and Z is the matrix output from the encoder stack. In other words, queries are computed in a learnably linear way from the input matrix, keys and values are computed in a learnably linear way from the encoder output, and each head returns the attention for that query-key-value triple.

Finally, the output of the attention layer is defined to be

$$\text{concat}(\text{head}_1, \ldots, \text{head}_n)W^O,$$

where concat is the function that concatenates vectors. In other words, the outputs from each head are combined in a learnably linear way to produce the overall output for the layer.

9.6.3 Self-attention layers

The self-attention layers in the encoder stack are similar to attention layers, the only change being the formula for the head outputs. For a self-attention layer, we have

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V).$$

In other words, the queries, keys, and values for a self-attention layer are all computed from its input matrix. The human-reader analogue for self-attention is taking note of specific words as you interpret others. For example, correctly interpreting the sentence `the dog caught the frisbee before it hit the ground` requires recognizing that the word `it` refers to the frisbee rather than the dog. An effectively trained self-attention layer would be able to place most of the softmax weight for the row corresponding to `it` in the position corresponding to the `frisbee` row of V.

9.6.4 Word order

Note that the transformer, as we have described it so far, does not handle rows differently based on their row index. Since word order is meaningful in natural languages, we will graft in the ability to distinguish words by their position in the sentence. We do this by fixing a predetermined *position* matrix and adding this matrix to the input before it reaches the first self-attention layer of each stack, as in Figure 9.16. The rows of the position matrix are distinct, so each row vector being fed into the first self-attention layer incorporates information about the word itself and about its position in the sentence.

9.6.5 Using transformers

We conclude this section by suggesting two tools for applying transformers to real-world problems without having to code them from scratch.

1. When OpenAI announced GPT-2, they released code for a small version of the model, which may be applied by users to their own datasets [398].
2. Google open-sourced a state-of-the-art transformer model called BERT, which can be trained efficiently in the cloud on the user's data [127].

GPT-2 can be fine-tuned[4] on user-collected plain text files, and its fundmental functionality is text continuation. In other words, the user supplies a segment of natural language text as input, and the output is an attempt at continuing the segment. This functionality can be leveraged to complete a variety of other tasks, however. For example, GPT-2 can be made to summarize a passage by concluding the input text with *TL;DR* (which is an Internet abbreviation used to introduce summaries; it stands for "too long; didn't read").

9.7 Deep learning frameworks

9.7.1 Hardware acceleration

Since larger neural networks tend to perform better on complex problems, state-of-the-art deep learning pushes the limits of modern computational resources. In particular, advanced models are trained on clusters of computers in a way that leverages *parallelization*. In this section we will discuss several ways to parallelize neural network computations.

GPU acceleration. The **CPU** (central processing unit) is the primary component in a computer that executes program instructions. In the 1970s, arcade game manufacturers recognized that the general-purpose design of the CPU made it suboptimally suited to the matrix operations used heavily in graphics rendering, so specialized chips called **GPU**s (graphics processing units) were introduced. GPUs generally have a slower clock speed than CPUs, but they can parallelize computations more efficiently. Matrix operations are ideally suited to parallelization-oriented optimization, since different entries in a matrix sum or product may be computed independently. Because neural network calculations are heavily reliant on matrix operations, using GPUs is a natural way to accelerate them. Users often experience speedups of 1–2 orders of magnitude from activating GPU acceleration, so GPUs are often considered essential for serious neural network training.

Hyperparameter parallelization. Hyperparameters are parameters which are not adjusted during model training; these include choices like the number of layers, the number of neurons in each layer, filter size in a convnet, etc. *Hyperparameter search* entails trying training several models with

[4]Fine-tuning refers to using new data to apply further training iterations to a model which has already been trained on a different, typically much larger dataset.

different choices of hyperparameters, for purposes of finding a good set of hyperparameters. This is an example of an *embarrassingly parallel* problem: the different training runs are not interrelated, so we can just deploy them to different machines and collect the results.

Model parallelization. We can split each layer into multiple pieces; for example, in a convolutional neural net we could put half the channels in a given layer on one GPU and half on another. This problem is not embarrassingly parallel, since the values from both sets of channels need to be communicated to the next layer. These data transfer costs trade off against the parallelization benefits. AlexNet manages this tradeoff by incurring the data transfer costs for only some of the layers. (See Figure 9.9.)

Data parallelization. We can train our model on separate mini-batches simultaneously. This entails a change to the training algorithm, since typically each mini-batch needs to be computed with weights that were updated based on the previous mini-batch. The idea is to store the weights in a centralized server and have each parallel implementation of the model read and write to the server whenever a mini-batch is finished [124].

9.7.2 History of deep learning frameworks

The application of neural networks to real-world problems entails a great deal of computational overhead. The user must

1. obtain data,
2. make various choices about the training process (such as model architecture, activations and loss functions, parameter initialization, optimization algorithm, etc.),
3. specify an algorithm implementing that set of choices, and
4. dispatch the computation to multiple processors for a parallelization speedup.

The first two steps require input from the user, but the third and fourth steps can be automated. This automation improves the user's experience by saving them time and reducing the number of opportunities to introduce bugs. A **deep learning framework** is a software package that efficiently implements neural network computations and provides a convenient interface for specification of data and algorithm options. The rise of deep learning frameworks has been a boon to reproducibility and researcher productivity.

In 2002, Ronan Collobert, Samy Bengio, and Johnny Mariethoz released **Torch**, a machine learning project that included support for multilayer perceptrons [105]. This framework was released as a C++ library[5] and was

[5]C++ is popular among software engineers and is often used in performance-sensitive applications.

intended primarily for users with the savvy to link and call C++ libraries from their own code. In 2008, they added a graphical user interface and an interface in the scripting language *Lua*, making their tools accessible to a broader audience [106]. Torch remained popular until the mid 2010's, when it was overshadowed by several frameworks backed by large companies. Active development on Torch halted in 2018.

The second major deep learning framework, a Python[6] library called **Theano**, was introduced in 2007 by the Montreal Institute for Learning Algorithms at the University of Montreal [462]. Theano served as a nexus of innovation in deep learning computation until 2017 when its developers saw fit to pass the baton to its well-resourced competitors [33].

The major entrants into the deep learning framework scene in the mid-2010s were Facebook, Google, and Microsoft. In 2016, Facebook introduced **PyTorch**, the Python-based heir to the design and internal structure of Torch. Facebook also released **Caffe2** in 2017, which was based on the Berkeley deep learning framework *Caffe*. PyTorch is typically used for convenient experimentation, and Caffe2 for production. In 2018, Facebook announced that the two systems would be merged, so that users may transition more seamlessly from research to deployment.

Google's **TensorFlow** was introduced in 2015, and **Microsoft Cognitive Toolkit** (originally called *CNTK*) was released in 2016. In 2015, François Chollet wrote the Python library **Keras**, which provides a unified interface to Theano, TensorFlow, and Microsoft Cognitive Toolkit. Computations are specified by the user in Keras and dispatched to the chosen *backend* for the heavy lifting. Keras has since been incorporated into both Tensorflow and Microsoft Cognitive Toolkit as a built-in way to interact with those libraries.

In 2017 Facebook and Microsoft announced the Open Neural Network Exchange (ONNX) to facilitate switching between frameworks. For example, a neural network model can be trained in one system, saved as a file in ONNX format, and then loaded into a different system to be used as a part of a visualization.

One of the main reasons that frameworks are indispensable to the deep learning researcher is that they allow computations to leverage GPU resources with the flip of a switch. Performing scientific computation on GPUs required considerable expertise and effort until around 2007 when the manufacturer NVIDIA[7] introduced an interface called **CUDA** for performing calculations on GPUs without having to heavily re-write code written for a CPU. Every major deep learning platform uses CUDA to support GPU computing.

[6]Python is a programming language whose design prioritizes convenience and flexibility. Python is often supplemented with packages like Theano, which use C or C++ code under the hood to achieve good performance for specific applications.

[7]The GPU market is dominated by two manufacturers: NVIDIA (roughly 70% market share) and AMD (about 30%).

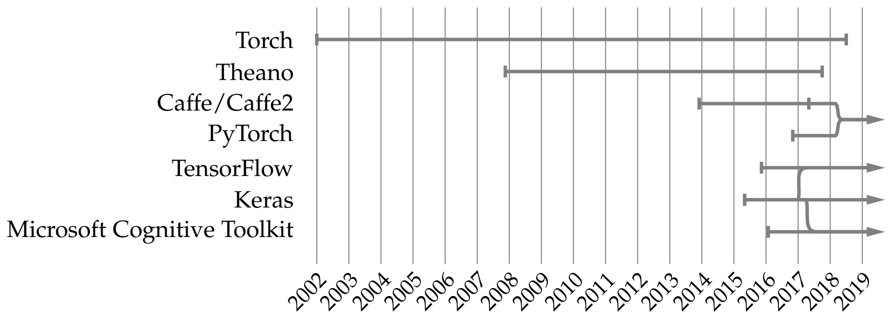


FIGURE 9.17: A timeline of popular deep learning frameworks. The lines merging from Keras to TensorFlow and Microsoft Cognitive Toolkit indicate the integration of a Keras interface.

9.7.3 TensorFlow with Keras

The primary two high-level deep learning interfaces are PyTorch and Keras, as in Figure 9.17. In this section, we will perform some basic neural network training with Keras. You can follow along in a live computing environment in your browser on the website of resources associated with this text [84]. Code in this section is written in Python, but Keras is accessible from other languages as well.

We begin by pulling image data into our session. In Keras, the built-in datasets are stored as submodules of `tensorflow.keras.datasets`, and the data are loaded using the `load_data` function in that module. This function returns two pairs: the training data (input and output) and the test data (input and output). Here we load a dataset of images of handwritten digits, similar to the left image in Figure 9.6.

```
from tensorflow.keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

We can run `X_train.shape` to check that `X_train` is a $60{,}000 \times 28 \times 28$ tensor, and similarly we find that `y_train` is a length-60,000 vector of labels (each of which is a digit, 0-9). The test data are similar, except that there are 10,000 images and labels.

The pixel intensities in the tensor of images are integers from 0 to 255, so we divide the tensors by 255 to get values ranging from 0 to 1.

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

Next we construct a Python object that represents a multilayer perceptron. The simplest way to do this is with a `Sequential` model, specifying the desired `Dense` layers in order. We start out with a `Flatten` layer, since dense layers expect vectors, while each image is a 28×28 matrix. Keras observes the convention that the sample axis is the first axis, so the flatten operation acts only on the other two axes.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
model = Sequential([Flatten(input_shape=(28,28)),
                    Dense(units=100,activation='relu'),
                    Dense(units=10,activation='softmax')])
```

Next we prepare the model for training using the `compile` method.

```
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

The `sparse_categorical_crossentropy` loss is appropriate for classification problems where the training labels for the n classes are $\{0, 1, \ldots, n-1\}$. The `optimizer` argument selects stochastic gradient descent as our optimization algorithm, and the `metrics` argument engages accuracy tracking during training.

Now we can train the model[8]:

```
model.fit(X_train, y_train, epochs=5)
```

The `epochs` argument specifies the number of passes through the dataset. For example, if the mini-batch size is 32 (the default value in Keras), then approximately 60,000/32 mini-batches are used per epoch.

Finally, we test our model using the test data.

```
results = model.evaluate(X_test, y_test)
dict(zip(model.metrics_names, results))
```

The last line zips the names of the metrics returned by the `evaluate` method with their values. We get something like

```
{'loss': 0.224, 'acc': 0.935},
```

indicating an accuracy of about 93.5%. Although this is far from the state of the art on MNIST (which is a relatively easy dataset to get good results on), this is a reasonable starting point and serves to illustrate that in fewer than 15 lines of code, you can create a neural network for image recognition! It also shows the key steps of a Keras workflow: build, compile, fit, and evaluate.

[8]This line of code took about 30 seconds to run on my notebook computer, while the previous steps each took a second or less.

9.8 Open questions

Gaining insight into the inner workings of a neural network is highly desirable for reasons of transparency, fairness, and scientific interest. However, methods for understanding neural networks are mostly descriptive. A robust mathematical theory of neural network training could offer substantial practical value.

A theory of neural network training on a classification might, for example, envisage each point $\mathbf{x} \in \mathbb{R}^n$ for a given class in a real-world dataset as a perturbation of a random point selected from a probability distribution on a relatively low-dimensional submanifold of $\mathbb{R}^n$. This idea is called the **manifold hypothesis**; see [156] for mathematical results on checking whether it holds for a given dataset.

If the manifolds corresponding to different classes are separable by a hyperplane, then solving the classification problem amounts to identifying a separating hyperplane. However, if they are entangled, then hopefully they transform under successive layers of the network to manifolds that are more easily separable (as in Figure 9.1). This idea has been investigated empirically [60], but progress remains elusive.

9.9 Exercises and solutions

Exercises for this chapter have been included at relevant points in the text itself, rather than collected at the end. Readers can find them on pages 415, 416, 418, and 424.

For readers who want a software develompent challenge, consider replacing Exercise 9.2 on page 416 with building your own multilayer perceptron from scratch in your favorite language, then comparing your work to what you find in [84]. For readers who want hands-on experience with applying the chapter's methods to data, start by repeating the work in Section 9.7.3 yourself, then try extending it to a new dataset.

Solution to Exercise 9.3. A model which outputs discrete classes can't be trained by gradient descent because there is typically no meaningful metric on the space of classes. Furthermore, a probabilistic output has the advantage of being able to faithfully represent uncertainty when the model is unable to confidently choose one class over others.

Solution to Exercise 9.4. If we flatten each three-dimensional array into a vector, then the computation performed by the convolutional layer is equivalent to an MLP layer with some weights constrained to be zero (the ones connecting any pair of pixels which are not close enough spatially to be related by the convolution operations) and others constrained to be equal (the ones which involve the same entry in the same filter in the convolution operation).

Chapter 10

Topological Data Analysis

Henry Adams, Johnathan Bush, and Joshua Mirth

Colorado State University

10.1 Introduction

Given a finite sampling of data points from a shape, what knowledge can we learn about the shape? For example, suppose that our dataset is sampled from a shape consisting of 17 "blobs," or components. In Chapter 5, we learned how to use clustering techniques to recover the fact that our dataset might have been sampled from a shape with 17 underlying components. But what if your points are sampled from a circle—is it possible to recover this circular shape from only a finite noisy sampling (Figure 10.1)? What if your dataset is sampled from a surface, such as a sphere, or torus, or two-holed torus (Figure 10.2)? Though it is rare for a dataset to lie on such a surface, later in this chapter we will see a natural example arising from conformations of the cyclo-octane molecule C_8H_{16}. What properties about a shape can you recover given only a finite sample of noisy data points from that shape?

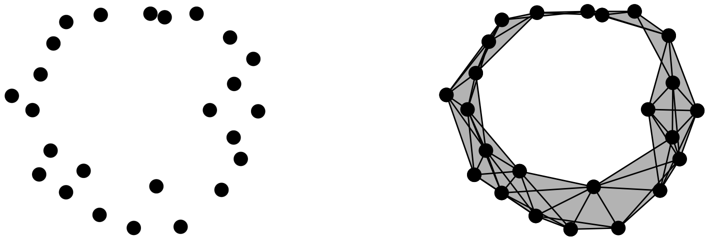


FIGURE 10.1: (Left) Points sampled from a circle with noise. (Right) A reconstruction of the circle.

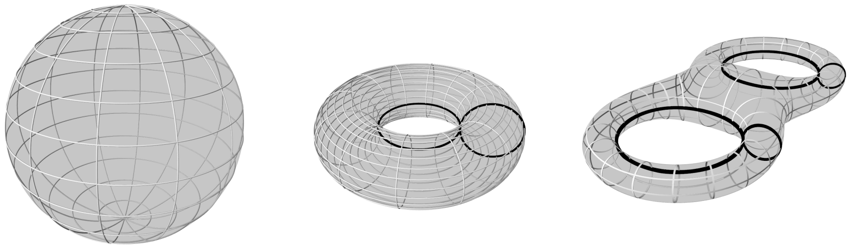


FIGURE 10.2: (Left) A hollow sphere has a single two-dimensional hole. (Middle) A hollow torus has a single two-dimensional hole and two one-dimensional holes (shown in black). (Right) A hollow double torus has a single two-dimensional hole and four one-dimensional holes (shown in black). Any other one-dimensional hole can be written as a "linear combination" of these four.

In this chapter, we explain how it is possible to recover *topological* properties about the shape from which one's dataset is sampled. Topology is a way to quantify the number of *holes*, or *topological features* (we will use these two names interchangeably), of each dimension in a space. Roughly speaking, a zero-dimensional feature is a connected component, a one-dimensional feature is a loop, and a two-dimensional feature is a void. For example, one can use topology to make precise the intuitive notions that a hollow sphere (the surface of a ball) has a single two-dimensional hole, whereas a hollow torus has a single two-dimensional hole and two one-dimensional holes (Figure 10.2).

We begin by describing example applications of topology to data analysis in Section 10.2, followed by a more formal introduction to topology in Section 10.3. In Section 10.4 we introduce *simplicial complexes*, which are the data structure we use to store topological spaces on a computer. We introduce *homology* in Section 10.5, which is a way to count the number of holes of each dimension in a space, i.e., to make the counts in Figure 10.2 precise. Section 10.6 describes *persistent homology*, which was developed in the late 1990s and early 2000s as a way to compute topological properties when one is given only a dataset, i.e., a finite sampling from some unknown underlying space. We end with additional sections on sublevelset persistence (Section 10.7), a survey of software packages for topological data analysis (Section 10.8), and a brief pointer to related papers and expository articles (Section 10.9). It is important for any algorithm in data analysis to be robust to noise, as all data measurements are inherently noisy to some degree, and we describe how persistent homology is robust to noise in Appendix 10.10. Some accompanying code for persistent homology, with an accompanying tutorial, is available on the website for this book [84].

10.2 Example applications

We give a survey of only a few of the beautiful example applications of topology to data analysis, including datasets arising from image processing, molecule configurations, agent-based modeling, and dynamical systems.

10.2.1 Image processing

If you take a large collection of black and white photographs of indoor and outdoor scenes, what are the most common 3×3 pixel patches, i.e., the most common tiny squares of adjacent pixels? Select out only the high-contrast patches, i.e. those that are far from being monochromatic, and normalize each patch to have the same "contrast norm," and to have the same average grayscale color. It turns out that the most common patches are linear gradients at all angles, forming a circle. The next most common patches are quadratic gradients at all angles, forming two additional circles (Figure 10.3). Topological data analysis can be used to find the number of one-dimensional holes in this dataset; for further details see [81] and the references [233, 483] within. As we describe in Section 10.6, one can use these low-dimensional models in order create image compression algorithms.

10.2.2 Molecule configurations

As another example of a dataset with an interesting shape, consider conformations of the cyclo-octane molecule. The cyclo-octane molecule C_8H_{16} consists of a ring of eight carbons atoms, each bonded to a pair of hydrogen

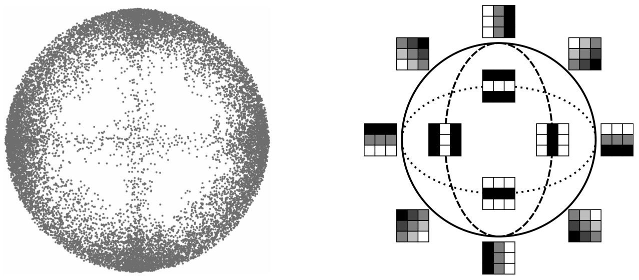


FIGURE 10.3: (Left) PCA projection of the 3×3 image patches described in the text. (Right) Three-circle model.

atoms in Figure 10.4 (left). A conformation of this molecule is a chemically and physically possible realization in 3D space $\mathbb{R}^3$, modulo translations and rotations. The locations of the carbon atoms in a conformation determine the locations of the hydrogen atoms via energy minimization, and hence each molecule conformation can be mapped to a point in $\mathbb{R}^{24} = \mathbb{R}^{8\cdot 3}$, as there are eight carbon atoms in the molecule, and each carbon location is represented by three coordinates x, y, z. This map realizes the conformation space of cyclo-octane as a subset of $\mathbb{R}^{24}$. The papers [71, 336, 337] show that the conformation space of cyclo-octane is the union of a sphere with a Klein bottle, glued together along two circles of singularities, as shown in Figure 10.4 (right).

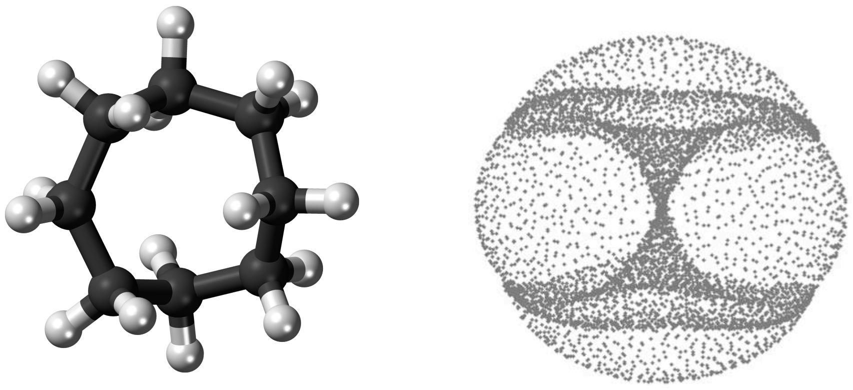


FIGURE 10.4: (Left) The cyclo-octane molecule consists of a ring of 8 carbon atoms (black), each bonded to a pair of hydrogen atoms (white). (Right) A PCA projection of a dataset of different conformations of the cyclo-octane molecule; this shape is a sphere glued to a Klein bottle (the "hourglass") along two circles of singularity. The right image is from [337], used with permission.

Indeed, using persistent homology in Section 10.6 and following [522], we compute the persistent homology of the dataset of cyclo-octane molecules. The computations show a single connected component, a single one-dimensional hole, and a part of two-dimensional holes, matching the homology groups of the union of a sphere with a Klein bottle, glued together along two circles of singularities.

10.2.3 Agent-based modeling

Of course, most datasets do not contain beautiful shapes such as spheres, tori, or Klein bottles. Nevertheless, persistent homology is still a useful way to summarize both the local geometric and global topological features that may be present in a noisy dataset, such as a flock of birds (Figure 10.5).

FIGURE 10.5: A flock of birds (artist's rendering) contains a lot of both local geometric and global topological data.

10.2.4 Dynamical systems

The prior examples are all built on point cloud data—i.e., a finite set of points. However, persistent homology can also be applied to functions through a process known as *sublevelset persistence.* Indeed, consider Figure 10.6 (left), which shows a Rayleigh-Bénard convection system obtained by mixing two metals [276]. Figure 10.6 (right) shows various sublevelsets of this function, as the thresholding parameter increases. Note that we obtain an increasing sequence of spaces, and hence can apply persistent homology to describe attributes of the Rayleigh-Bénard function. Though the focus of this chapter is persistent homology for point clouds, we return to sublevelset persistence in Section 10.7.

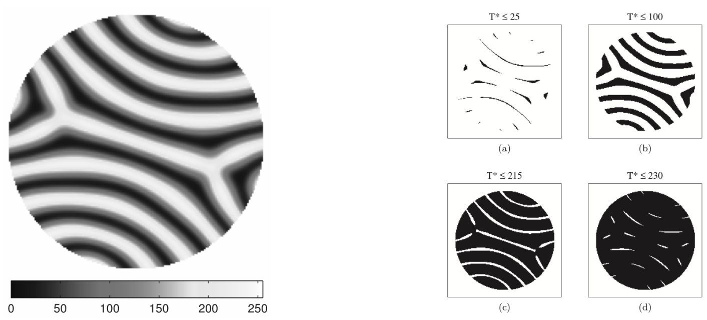


FIGURE 10.6: (Left) A surface created from Rayleigh–Bénard convection. (Right) Four increasing sublevelsets from that surface. Images from [276], used with permission.

10.3 Topology

We begin by giving a high-level introduction to topology. The mathematician reader may be somewhat familiar with the general idea of the subject, but a brief review may be helpful. Topology was developed by Poincaré and others in the 1900s. Topology it is now one of the major branches of mathematics, and it interacts in important and surprising ways with many of the other main branches of mathematics, such as analysis, algebra, geometry, combinatorics, logic, number theory, etc.

The reader may recall that in Chapter 7 we learned how to recover *geometric* properties of a space from only a finite sample. What is the difference between geometric and topological properties? Geometric properties include, for example, distances and curvatures—things that can be measured with a ruler. By contrast, topological properties are invariant upon stretching and bending (but not ripping or gluing) a shape. For example, to a geometer, a circle is not the same thing as an ellipse, since a circle has constant curvature, whereas an ellipse has regions of larger and smaller curvature. By contrast, to a topologist, a circle is considered to be the same shape as an ellipse, since it is possible to continuously deform the circle until one obtains the ellipse, or vice versa (Figure 10.7). This topological perspective emphasizes what the circle and the ellipse both share in common: a single hole.

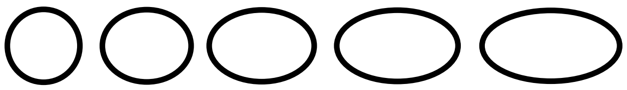


FIGURE 10.7: The transformation of a circle into an ellipse.

As another example, it is frequently said that a topologist cannot distinguish between her coffee cup and her doughnut. Indeed, consider Figure 10.8, in which you can see a coffee cup gradually deform into a doughnut as one side shrinks away the cup portion, while the handle expands into a larger ring shape, until we we are left with an edible doughnut!

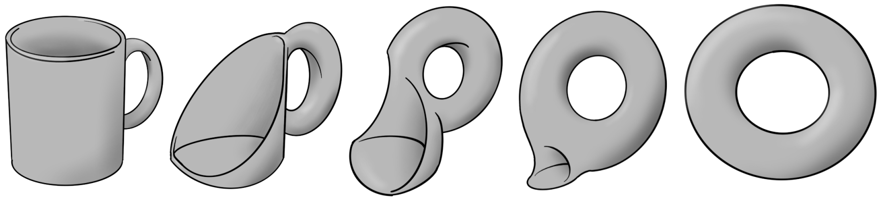


FIGURE 10.8: The transformation of a doughnut into a coffee cup.

In mathematics, ignoring structure is sometimes detrimental, but it is also sometimes advantageous. For example, parity arguments in mathematics ignore an integer's exact value in order to emphasize whether that integer is odd or even, which sometimes is all that matters. In topology, we ignore some geometric structure (such as distances or curvature) in order to emphasize other properties, such as the number or type of *holes* in a space. Roughly speaking, a topological property is one that is invariant upon stretching or bending (but not tearing or gluing) a space.

The field of algebraic topology studies ways to assign algebraic invariants to topological spaces. The word "invariant" implies that two topological spaces which can be deformed to each other will be assigned the same algebraic data, i.e., these algebraic assignments are invariant (or unchanged) upon stretching or bending a space. In other words, these algebraic invariants depend only on the topological properties of the underlying space. In Section 10.5, we will learn more about one such algebraic invariant, called *homology*.

10.4 Simplicial complexes

A simplicial complex is a collection of points, line segments, triangles, tetrahedra, and "higher-dimensional tetrahedra," called *simplices*, together with data specifying how these spaces are attached. In particular, we require simplices to be attached along faces, defined below, ensuring that every point of the complex is contained in a unique maximal simplex. A simplicial complex definines a triangulation of a topological space, whereby the space has been decomposed into simpler parts (simplices) admitting a feasible computational analysis (Figure 10.9). Simplicial complexes are useful in data analysis because a simplicial complex built on top of a finite dataset (a finite number of vertices) admits a combinatorial description that one can store and manipulate on a computer.

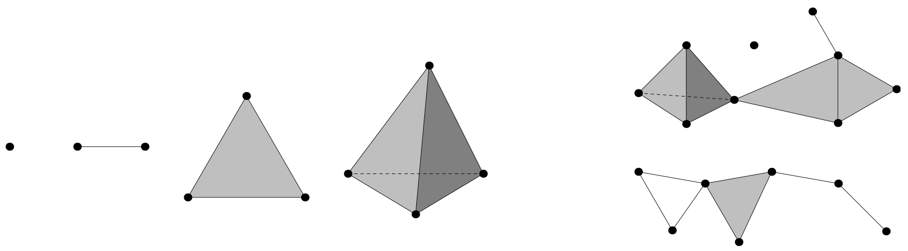


FIGURE 10.9: (Left) Simplices of dimension zero, one, two, and three. (Right) Simplices of various dimensions glued together to form a simplicial complex.

Formally, a finite *abstract simplicial complex* K with vertex set $V = \{v_0, v_1, \ldots, v_n\}$ is a collection of subsets[1] of V, such that K is closed under taking subsets, and every singleton of V belongs to K. By "closed under taking subsets" we mean that if $\sigma \subseteq V$ is in K and $\tau \subseteq \sigma$, then τ is also necessarily in K. By convention, we often ignore the element $\varnothing \in K$ because it is irrelevant to the topology of K. Each element of K is called a *face* of K, and a subcollection of elements of K consisting of all subsets of a single face defined by $d + 1$ vertices is called a *d-simplex*.

To an abstract simplicial complex K we may define an associated topological space, denoted $|K|$, called the *geometric realization* of K. First, given a d-simplex σ, define $|\sigma|$ to be the convex hull of $d + 1$ linearly independent vectors in $\mathbb{R}^{d+1}$. Then, roughly speaking, the geometric realization $|K|$ of a simplicial complex K is a topological space obtained by taking the disjoint union of each $|\sigma|$, for $\sigma \in K$, and then identifying these topological simplices along common faces as determined by the combinatorial structure of K. In this way, an abstract simplicial complex provides a purely combinatorial description of a decomposition of a topological space into geometric simplices, specifying both which simplices are present and the ways in which they are attached along faces (Figure 10.10).

$$\Big\{\{0,1,2\},\{0,1\},\{0,2\},\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{3,5\},\{0\},\{1\},\{2\},\{3\},\{4\},\{5\},\emptyset\Big\}$$

K

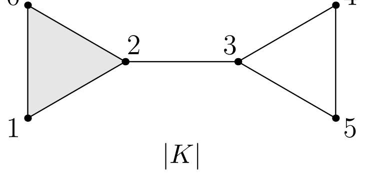


FIGURE 10.10: An abstract simplicial complex K (left) and its geometric realization $|K|$ (right).

[1] i.e., a subset of the powerset $\mathcal{P}(V)$

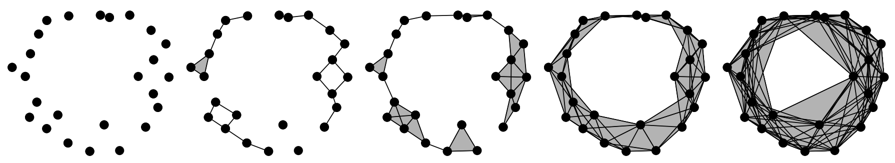


FIGURE 10.11: A dataset and its Vietoris–Rips complexes at five different choices of scale.

In topological data analysis, we assume that our dataset X arises as a finite sample of points from an unknown space. Two common methods of defining a simplicial complex from a metric space X are the *Vietoris–Rips* and *Čech simplicial complexes.*[2] By definition, the Vietoris–Rips simplicial complex with vertex set X and scale parameter $r > 0$, denoted $\mathrm{VR}(X; r)$, contains a simplex $\{v_0, \ldots, v_d\}$ whenever its diameter is less than r, i.e., $\mathrm{diam}(\{v_0, \ldots, v_d\}) < r$. Similarly, the Čech simplicial complex with vertex set X and scale parameter $r > 0$, denoted $\check{\mathrm{C}}(X; r)$, contains a simplex $\{v_0, \ldots, v_d\}$ whenever the collection of open balls of radius $r/2$ centered at each v_i have a nonempty intersection.

Hence, both the Vietoris–Rips and Čech simplicial complexes associate to X a topological space depending on a notion of "closeness" of points within X. Further, given $0 \leq r \leq r'$, it follows that $\mathrm{VR}(X; r) \subseteq \mathrm{VR}(X; r')$ and $\check{\mathrm{C}}(X; r) \subseteq \check{\mathrm{C}}(X; r')$, that is, both complexes define a *filtration* of topological spaces parametrized by r. This observation is closely related to *persistent homology* in Section 10.6; the word *persistent* implies that we are interested in which topological features or holes persist over a range of scale parameters r, as shown in Figure 10.11.

10.5 Homology

Homology provides a method to associate a sequence of algebraic structures, e.g., vector spaces or modules, to a topological space. Importantly, the structures computed by homology are algebraic invariants, meaning they depend only on topological properties of the space, and remain constant under certain allowable transformations, e.g., stretching or bending. Roughly speaking, the homology invariants measure the number of connected components, one-dimensional holes, two-dimensional voids, etc., of the space. The computation of homology is motivated by the observation that two spaces may be distinguished by comparing the set of such invariants associated to each space. As an example, a circle is not the same shape (up to stretching and bending)

[2]Čech is pronounced approximately like "check."

as a figure-eight, because a circle contains a single hole while a figure-eight contains two holes (Figure 10.12).

FIGURE 10.12: (Left) A circle and a simplicial complex with the same shape. (Right) A figure-eight and a simplicial complex with the same shape.

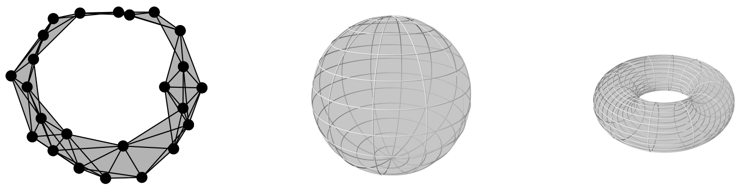


FIGURE 10.13: In all three shapes X above, $H_0(X) = F$ indicates the presence of a single connected component. (Left) $H_1(\text{circle}) = F$ corresponds to the one-dimensional hole. (Middle) $H_2(\text{sphere}) = F$ corresponds to the two-dimensional hole. (Right) The torus has two one-dimensional holes and a single two-dimensional hole, giving $H_1(\text{torus}) = F^2$ and $H_2(\text{torus}) = F$.

More precisely, homology associates to a space X a sequence of abelian groups $H_0(X), H_1(X), H_2(X), \ldots$, such that $H_d(X)$ gives a measure of the number of "d-dimensional holes" in X (Figure 10.13). Because the homological groups associated to a space X remain constant under continuous deformations of the space (that is, the groups are topological invariants), it is sufficient to compute the homology of a simplicial complex which has the same shape as X (Figure 10.12).

10.5.1 Simplicial homology

There are a number of equivalent methods to compute the homology of a space. We consider one particular method of computation, simplicial homology, which is motivated by the observation that the homology of a finite simplicial complex is computable through techniques of linear algebra. For simplicity, throughout the remainder of this section, we fix the coefficients to live in the field $F = \mathbb{F}_2 = \{0, 1\}$ of two elements, in which $1 + 1 = 0$. As a consequence, each homology group $H_n(X)$ is a vector space[3] over F. Furthermore, since addition is the same as subtraction in the field of two elements, we do not need to keep track of negative signs in any of the following formulas.

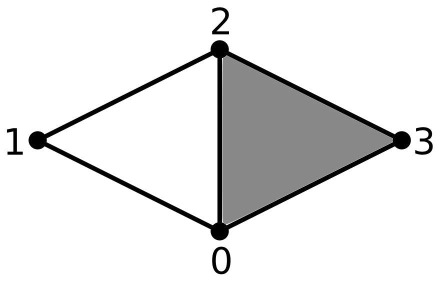


FIGURE 10.14: We compute the homology of simplicial complex J and find a single connected component, a single one-dimensional hole, and no higher-dimensional holes.

As an example, consider the simplicial complex J with vertices $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, edges $\{0,1\}$, $\{0,2\}$, $\{0,3\}$, $\{1,2\}$, $\{2,3\}$, the triangle $\{0,2,3\}$, and no higher-dimensional faces (Figure 10.14). This space has one connected component, a single one-dimensional hole, and no two-dimensional voids or higher-dimensional holes. Therefore, we expect to compute $H_0(J) = F$, $H_1(J) = F$, and $H_i(J) = 0$ for $i > 1$. More generally, for a space X with n distinct k-dimensional holes, we expect to compute $H_n(X) = F^k$. Roughly speaking, in order to detect these topological features in J, we should understand the relationship between simplices creating holes, and those filling holes. More precisely, observe that the central hole in this example is determined by the *existence* of all three edges $\{0,1\}$, $\{1,2\}$, $\{0,2\}$ in J together with the *absence* of the two-dimensional face $\{0,1,2\}$ in J.

10.5.2 Homology definitions

In light of this guiding example, let K denote a finite simplicial complex defined on a vertex set V. If $\sigma = \{v_0, v_1, \ldots, v_d\}$ is a d-simplex in K, then let $\{v_0, \ldots, \widehat{v_i}, \ldots, v_d\} = \{v_0, \ldots, v_{i-1}, v_{i+1}, \ldots, v_d\}$ denote the simplex obtained by removing the i^{th} vertex of σ; we call such a simplex the *i^{th} boundary face of σ.*

Define a *simplicial d-chain* to be a finite formal[4] linear sum $\sum_{i=1}^{n} \sigma_i$, where each σ_i is a d-simplex of K. Let C_d denote the set of all simplicial d-chains. Note that C_d has the structure of a vector space over our field F

[3]By making different choices of coefficients, one can instead endow homology groups with the structure of an abelian group or a module.

[4]By *formal* we mean that if σ and σ' are two d-simplices, then we are allowed to write $\sigma + \sigma'$ as a d-chain, even though we will not define how to "add" the two simplices σ and σ' together.

under addition. For example, if $\sigma, \sigma', \sigma'', \sigma'''$ are d-simplices in K, and if we add the d-chains $\sigma + \sigma' + \sigma''$ and $\sigma + \sigma'''$ together, then we obtain the d-chain $\sigma' + \sigma'' + \sigma'''$ as output, where the two σ summands have cancelled since we are using $F = \mathbb{F}_2$ coefficients in which $1 + 1 = 0$.

Define the d^{th} boundary operator ∂_d by

$$\partial_d(\sigma) = \sum_{i=0}^{d} \{v_0, \ldots, \widehat{v_i}, \ldots, v_d\},$$

and extend linearly on simplices to obtain a map $\partial_d \colon C_d \to C_{d-1}$, called the d^{th} *boundary map.* By definition, observe that each boundary map is determined by its effect on the generators of C_d. For example, if $\sigma = \{v_0, v_1, v_2\}$, then $\partial_2(\sigma) = \{v_1, v_2\} + \{v_0, v_2\} + \{v_0, v_1\}$. If $\sigma = \{v_0, v_1, v_2\}$ and $\sigma' = \{v_0, v_1, v_3\}$, then

$$\begin{aligned}
&\partial_2(\sigma + \sigma') \\
&= \partial_2(\sigma) + \partial_2(\sigma') \\
&= (\{v_1, v_2\} + \{v_0, v_2\} + \{v_0, v_1\}) + (\{v_1, v_3\} + \{v_0, v_3\} + \{v_0, v_1\}) \\
&= \{v_0, v_2\} + \{v_0, v_3\} + \{v_1, v_2\} + \{v_1, v_3\}.
\end{aligned}$$

The last equality above follows since we are using coefficients in $F = \mathbb{F}_2$, the field of two elements, in which $1 + 1 = 0$. Hence $\{v_0, v_1\} + \{v_0, v_1\}$ cancels to give zero.

Definition 10.1 *The d^{th}* homology group *of a simplicial complex K (with coefficients in F) is the abelian group*

$$H_d(K) = \frac{\ker \partial_d}{\operatorname{im} \partial_{d+1}}.$$

Elements of $\ker \partial_d$ *are called d-*cycles*, and elements of* $\operatorname{im} \partial_{d+1}$ *are called d-*boundaries *of K.*

10.5.3 Homology example

Let us apply these definitions to compute the homology groups of the example complex J from Figure 10.14, and verify that these groups agree with our intuition. In this case, observe

$$\begin{aligned}
C_0 &= \{a\{0\} + b\{1\} + c\{2\} + d\{3\} \mid a, b, c, d \in F\}, \\
C_1 &= \{a\{0,1\} + b\{0,2\} + c\{0,3\} + d\{1,2\} + e\{2,3\} \mid a, b, c, d, e \in F\}, \\
C_2 &= \{a\{0,2,3\} \mid a \in F\},
\end{aligned}$$

TABLE 10.1: The chain, cycle, boundary, and homology groups of the example simplicial complex J from Figure 10.14.

d	0	1	2	3	≥ 4
C_d	F^4	F^5	F	0	0
$\ker \partial_d$	F^4	F^2	0	0	0
$\operatorname{im} \partial_{d+1}$	F^3	F	0	0	0
H_d	F	F	0	0	0

and $C_d = \{0\}$ for all $d > 2$. Hence, $C_0 \cong F^4$, $C_1 \cong F^5$, and $C_2 \cong F$. Next, observe that the boundary operators are generated by the following equations:

$$\begin{aligned}
&\partial_0(\{v_0\}) = 0 \text{ for all vertices } \{v_0\} \text{ of } J\\
&\partial_1(\{v_0, v_1\}) = \{v_1\} + \{v_0\} = \{v_0\} + \{v_1\} \text{ for any edge } \{v_0, v_1\} \text{ of } J\\
&\partial_2(\{0,2,3\}) = \{2,3\} + \{0,3\} + \{0,2\} = \{0,2\} + \{0,3\} + \{2,3\}\\
&\partial_d \text{ is the zero map for } d > 2.
\end{aligned}$$

It is straightforward to verify that

$$\begin{aligned}
\ker \partial_1 &= \{0, \{0,1\} + \{0,2\} + \{1,2\},\\
&\qquad\qquad \{0,2\} + \{0,3\} + \{2,3\}, \{0,1\} + \{0,3\} + \{1,2\} + \{2,3\}\}\\
&= \{a(\{0,1\} + \{0,2\} + \{1,2\}) + b(\{0,2\} + \{0,3\} + \{2,3\}) \mid a, b \in F\}\\
&\cong F^2.
\end{aligned}$$

To see the second equality above, note that

$$(\{0,1\}+\{0,2\}+\{1,2\})+(\{0,2\}+\{0,3\}+\{2,3\}) = \{0,1\}+\{0,3\}+\{1,2\}+\{2,3\}.$$

We furthermore compute

$$\begin{aligned}
\operatorname{im} \partial_2 &= \{0, \{0,2\} + \{0,3\} + \{2,3\}\}\\
&= \{a(\{0,2\} + \{0,3\} + \{2,3\}) \mid a \in F\}\\
&\cong F.
\end{aligned}$$

Hence, we find $H_1(J) = \frac{F^2}{F} \cong F$, as expected. In a similar way, one may compute the remaining cycle groups (elements of the kernel ∂_d) and boundary groups (elements of the image ∂_{d+1}), from which it follows that $H_0(J) = F$ and $H_d(J) = 0$ for $d \geq 2$. These computations are summarized in Table 10.1.

10.5.4 Homology computation using linear algebra

The reader may object to the ad-hoc nature of the sequence of equations leading to $\ker \partial_1 \cong F^2$ and $\operatorname{im} \partial_2 \cong F$ in the above example. Indeed, an attractive feature of simplicial homology is that it may be computed algorithmically with techniques of linear algebra. Here, we demonstrate a portion of this algorithmic approach for the above simplical complex, and leave the details of the intermediate steps as an exercise.

As an example, let us consider the first boundary map ∂_1 associated to the example complex J. From the perspective of linear algebra, ∂_1 is just a linear map from the vector space $C_1 = F^5$ to the vector space $C_0 = F^4$. Hence, we may represent ∂_1 as a matrix, say, M, and compute the dimension of both the image of ∂_1 (i.e., the rank of M) and the kernel of ∂_1 (i.e., the nullity of M) through simple Gaussian elimination (as in Section 3.2.2.1). In addition, as long as we take care to maintain a consistent labeling of both the rows and the columns of M, we may simultaneously compute generating sets for both of these vector spaces. We will use not only row operations but also column operations.

In light of the definition of ∂_1, observe

$$M = \begin{array}{c} \\ \{0\} \\ \{1\} \\ \{2\} \\ \{3\} \end{array} \begin{array}{c} \begin{array}{ccccc} \{0,1\} & \{0,2\} & \{0,3\} & \{1,2\} & \{2,3\} \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}.$$

For example, the second column of M contains ones in the rows associated to $\{0\}$ and $\{2\}$ since $\partial_1(\{0,2\}) = \{0\} + \{2\}$. Next, we perform Gaussian elimination while keeping track of the labels associated to each row and column. First, swap rows to obtain

$$\begin{array}{c} \\ \{1\} \\ \{2\} \\ \{3\} \\ \{0\} \end{array} \begin{array}{c} \begin{array}{ccccc} \{0,1\} & \{0,2\} & \{0,3\} & \{1,2\} & \{2,3\} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}.$$

Add the first two columns to the fourth column to obtain

$$\begin{array}{c} \\ \{1\} \\ \{2\} \\ \{3\} \\ \{0\} \end{array} \begin{array}{c} \begin{array}{ccccc} \{0,1\} & \{0,2\} & \{0,3\} & z_1 & \{2,3\} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array},$$

TABLE 10.2: The homology groups of some common spaces, the last three of which are pictured in Figure 10.2.

Space X	$H_0(X)$	$H_1(X)$	$H_2(X)$
Circle	F	F	0
Figure eight	F	F^2	0
two-sphere	F	0	F
Torus	F	F^2	F
Double torus	F	F^4	F

where $z_1 = \{0,1\} + \{0,2\} + \{1,2\}$. Add the second two columns to the fifth column to obtain

$$\begin{array}{c} \\ \{1\} \\ \{2\} \\ \{3\} \\ \{0\} \end{array} \begin{array}{c} \begin{array}{ccccc} \{0,1\} & \{0,2\} & \{0,3\} & z_1 & z_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array},$$

where $z_2 = \{0,2\} + \{0,3\} + \{2,3\}$. Note that adding column i to column j affects the label on column j. By contrast, adding row i to row j affects the label on row i. We add the first row to the fourth row to obtain

$$\begin{array}{c} \\ \{0\}+\{1\} \\ \{2\} \\ \{3\} \\ \{0\} \end{array} \begin{array}{c} \begin{array}{ccccc} \{0,1\} & \{0,2\} & \{0,3\} & z_1 & z_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}.$$

Continuing in this way, we add the second row to the fourth row, and the third row to the fourth row, to obtain our final diagonalization

$$\begin{array}{c} \\ \{0\}+\{1\} \\ \{0\}+\{2\} \\ \{0\}+\{3\} \\ \{0\} \end{array} \begin{array}{c} \begin{array}{ccccc} \{0,1\} & \{0,2\} & \{0,3\} & z_1 & z_2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$

This matrix has rank three and nullity two. Further, the first three rows form a basis for $\operatorname{im} \partial_1$, and the last two columns z_1 and z_2 form a basis for $\ker \partial_1$.

In Table 10.2 we list the homology groups of some common spaces.

Readers interested in solidifying their understanding of homology may try the following exercises. We then apply these ideas to datasets in Section 10.6.

Exercise 10.1 *Compute the one-dimensional homology of a figure-eight. You could use, for example, the simplicial complex model given in Figure 10.12 (right).*

Exercise 10.2 *Compute the two-dimensional homology of the boundary of a tetrahedron, which consists of four vertices, six edges, and four triangles (or two-simplices), but no three-simplices.*

Exercise 10.3 *Compute the one-dimensional homology of the one-skeleton of a tetrahedron (which contains four vertices and six edges). Should you expect to find three one-dimensional holes or four one-dimensional holes?*

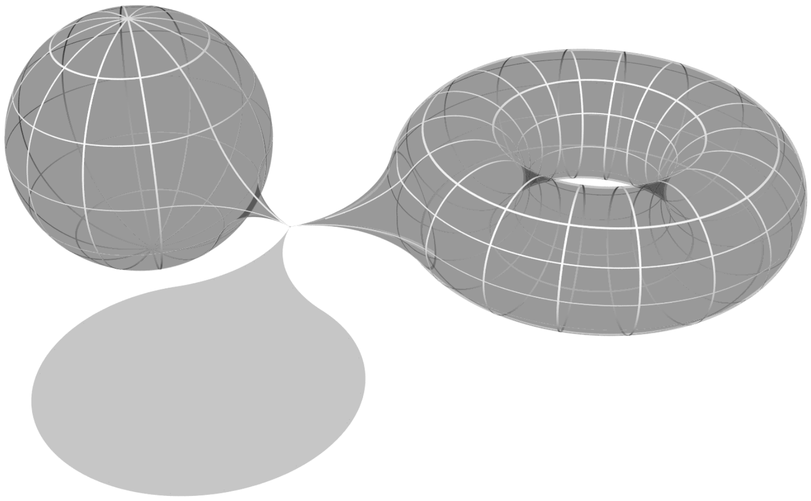


FIGURE 10.15: A sphere, a disc, and a torus attached at a point.

Exercise 10.4 *Consider the topological space X consisting of a sphere, a disk, and a torus attached at a point (the so-called* wedge sum *of these spaces) in Figure 10.15. First, write down a triangulation of X. Let V, E, and F denote the total number of distinct vertices, edges, and faces of your triangulation, respectively. Compute the number $\chi = V - E + F$, which is called the* Euler characteristic *of X.*

Given a topological space Y, let $\beta_k(Y)$ denote the dimension of $H_k(Y)$ as an F-vector space. We call $\beta_k(Y)$ the k^{th} Betti number *of Y. Now, given that the k^{th} Betti number of a wedge of two topological spaces is the sum of the k^{th} Betti number of each (for $k > 0$), compute the following alternating sum for the space X:*

$$\psi := \sum_{k \geq 0} (-1)^k \beta_k(X).$$

What do you observe about the relationship between χ and ψ? If this observation is true in general, then using the fact that homology groups are topological invariants, would this imply that the Euler characteristic is another example of a topological invariant which remains constant under allowable transformations of the space? In this way, homology can be viewed as a generalization of the Euler characteristic.

10.6 Persistent homology

In Chapter 5 we learned about hierarchical clustering, in which one studies how the clusters within a dataset form and merge as one increases some notion of scale. In this section, we study a generalization of hierarchichal clustering, in which one studies how the homology groups of a dataset change as one increases the scale.

Consider for example a finite subset $X \subseteq \mathbb{R}^n$, and its Vietoris–Rips complex $\mathrm{VR}(X; r)$ as defined in Section 10.4. If we consider an increasing sequence of scale parameters $r_1 \leq r_2 \leq r_3 \leq \ldots \leq r_{m-1} \leq r_m$, then we obtain an increasing sequence of simplicial complexes (Figure 10.16 (top))

$$\mathrm{VR}(X; r_1) \subseteq \mathrm{VR}(X; r_2) \subseteq \mathrm{VR}(X; r_3) \subseteq \ldots \subseteq \mathrm{VR}(X; r_{m-1}) \subseteq \mathrm{VR}(X; r_m).$$

Persistent homology will give us a language to describe not only the holes in $\mathrm{VR}(X; r_j)$ at each scale r_j, but also how the holes in $\mathrm{VR}(X; r_j)$ relate to those in $\mathrm{VR}(X; r_{j+1})$—namely which holes at scale r_{j+1} are new (not present at scale r_j), which holes at scale r_j die at scale r_{j+1}, and which holes remain or *persist* from scale r_j to scale r_{j+1}. The same can be done with Čech complexes instead of Vietoris–Rips complexes.

More generally, let $K_1 \subseteq K_2 \subseteq K_3 \subseteq \ldots \subseteq K_{m-1} \subseteq K_m$ be *any* nested sequence of topological spaces, each one contained inside the next. (The prior paragraph considers the case when $K_i = \mathrm{VR}(X; r_i)$.) We can apply i-dimensional homology to get a sequence

$$H_i(K_1) \to H_i(K_2) \to H_i(K_3) \to \ldots \to H_i(K_{m-1}) \to H_i(K_m)$$

of homology groups with maps in-between.[5] The *i-dimensional persistent homology* of this increasing sequence of spaces can be represented as a set of intervals (Figure 10.16 (middle)), referred to as the *persistence barcode*. In this barcode representation, each interval corresponds to an i-dimensional topological feature, i.e., to an i-dimensional topological hole. An interval's

[5]We remark that even though $K_j \subseteq K_{j+1}$ is an inclusion, the map $H_i(K_j) \to H_i(K_{j+1})$ need not be an inclusion in any sense.

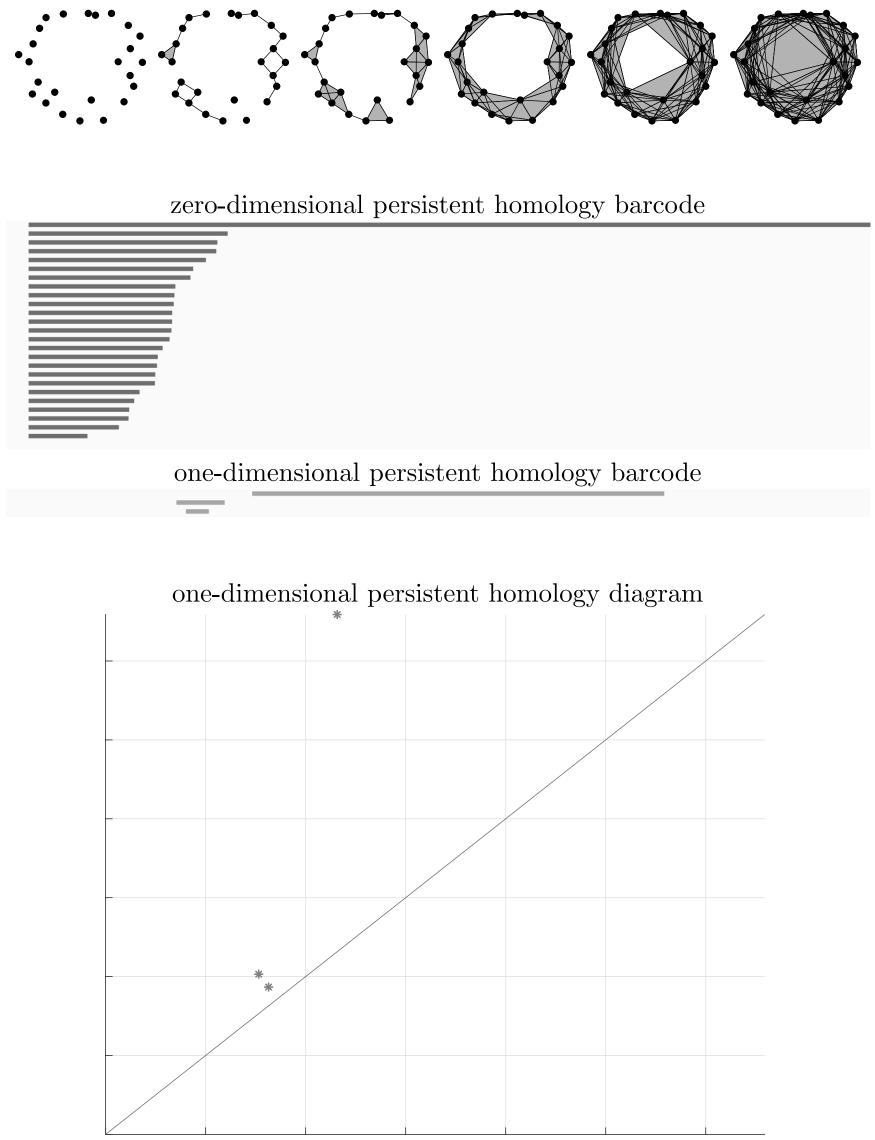


FIGURE 10.16: (Top) A dataset and its Vietoris–Rips complexes at five different choices of scale. (Middle) The zero-dimensional and one-dimensional persistent homology intervals. The horizontal axis is the scale parameter r in the construction of the Vietoris–Rips complex. (Bottom) The one-dimensional persistent homology diagram. The horizontal axis is the birth time, and the vertical axis is the death time.

birth-time corresponds to the first index at which that topological feature appears, and its death-time corresponds to the index when the topological feature becomes trivial.[6]

In persistent homology barcodes, the horizontal axis corresponds to the filtration value. For example, in Figure 10.16 (middle), the horizontal axis corresponds to the scale parameter r in the definition of the Vietoris–Rips complex. By contrast, the vertical axis in persistent homology barcodes has no meaning, in the sense that if we swap the vertical order of two intervals (in the same dimension), then the persistent homology barcode is considered to be unchanged.

The information in a persistence barcode can equivalently be presented as a persistence diagram. Indeed, in a persistence diagram, each interval I with birth-time b and death-time d is represented as the point (b, d) in the plane $\mathbb{R}^2$. Since we have $b \leq d$ for each such interval, i.e. since each topological feature is born before it dies, a persistence diagram is a collection of points in the plane above the diagonal line $y = x$. See Figure 10.16 (bottom).

If we are computing the persistent homology of a filtration that consists of increasing Vietoris–Rips complexes or Čech complexes of a finite dataset $X \subseteq \mathbb{R}^n$, then the way to read the persistent homology intervals is as follows. If r is smaller than the distance between the two closest points in X, then $\mathrm{VR}(X; r) = X$ is just a finite set of points with no interesting topology, and hence we are not interested in the topology of $\mathrm{VR}(X; r)$ for very small scale parameters r. Similarly, if r is larger than the diameter of X, then $\mathrm{VR}(X; r)$ is a simplex on vertex set X, which again has no interesting topology, and hence we are not interested in the topology of $\mathrm{VR}(X; r)$ for very large scale parameters r. Instead, we are interested in an intermediate range of scale parameters, and furthermore, we are most interested in those features which persist for a longer range. See for example the persistent homology intervals in Figure 10.16. The one long zero-dimesional interval and the one long one-dimensional interval correspond to the true homology of the circle, which has one connected component and a single one-dimensional hole. By contrast, the shorter zero-dimensional and one-dimensional intervals should be disregarded as sampling noise. In general, it is not easy to distinguish intervals corresponding to true features of the dataset from those corresponding to noise coming from the sampling.

Part of the reason for the popularity of persistent homology is that it is computable! Indeed, the algorithm for persistent homology is a cubic $O(n^3)$ operation, where n is the number of simplices that appear [144]. (This can be reduced to $O(n^{2.376})$, where the exponent 2.376 is coming from the computational complexity of matrix multiplication [348].)

[6]Trivial means either zero, or alternatively, a linear combination of other topological features.

We return now to two of the datasets mentioned in Section 10.2, in order to describe their persistent homology, and how that persistent homology recovers important patterns within the datasets.

The first such dataset from Section 10.2.1 is of 3×3 pixel patches taken from black-and-white photographs. Only the high-contrast patches, the ones that are far from being monochromatic, are considered. Each patch is normalized to have unit "contrast norm," and to have the same average value. The most common such patches, according to a certain estimate of density, are shown via a PCA projection in Figure 10.3 (left). From this PCA projection, which looks like a circle with a cross inside it, one might guess that this dataset has four one-dimensional holes. However, the persistent homology barcodes in Figure 10.17 show five robust one-dimensional features. These five one-dimensional features allowed the researchers in [81] to develop the three-circle model for this dataset in Figure 10.18. This model indeed has five one-dimensional holes, as illustrated in the same figure. The conclusion of the three-circle model is that the most common 3×3 image patches are linear gradients at all angles, forming a circle. The next most common patches are quadratic gradients at all angles, forming the two additional circles in the three-circle model. See the website for this book [84] for a tutorial that computes these persistent homology barcodes.

Is there value of having a compact model for a dataset? One can use a compact model for data compression. Indeed, a 3×3 image patch could be stored not as a list of 9 real numbers, but instead as a point on the (one-dimensional) three-circle model plus an error vector. This is the first step

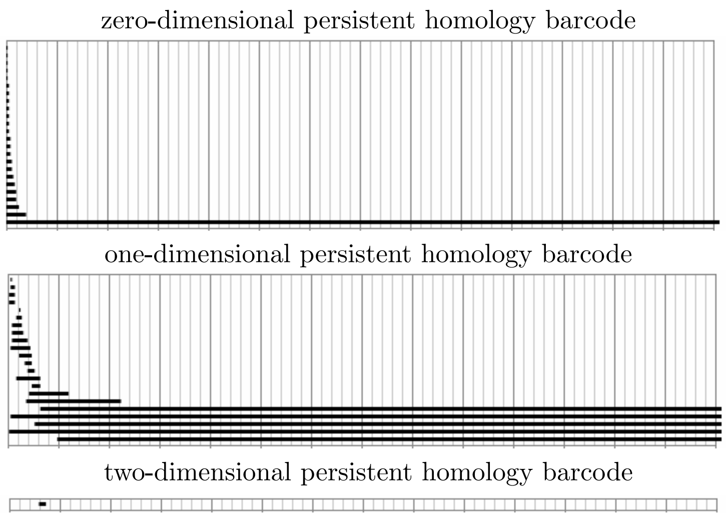


FIGURE 10.17: zero-, one-, and two-dimensional persistent homology barcodes for the dataset of 3×3 optical image patches. We have five significant one-dimensional intervals.

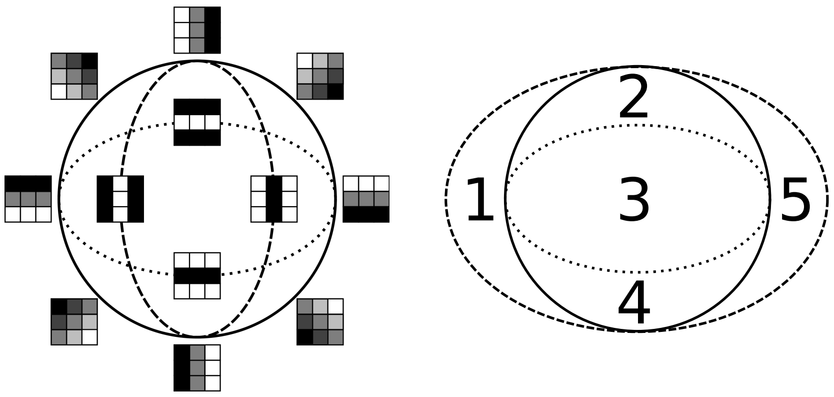


FIGURE 10.18: (Left) The three-circle model for the dataset of 3×3 optical image patches. (Right) A topological deformation of this model, which shows the five one-dimensional holes.

towards a Huffman-type code [236], as used for example in JPEG [488]. In [81], the three-circle model is extended to give a Klein bottle model for 3×3 image patches, which does provide a reasonable image compression scheme. The Klein bottle model has also been used in [386], along with Fourier analysis, to build a rotation-invariant texture classification algorithm.

A second interesting dataset is the space of conformations of the cyclo-octane molecule, considered in Section 10.2.2. Cyclo-octane C_8H_{16} consists of a ring of 8 carbons atoms, along with 16 hydrogen atoms. The locations of the carbon atoms can be used to map each molecule conformation to a point in $\mathbb{R}^{24}$, where the dimension $24 = 8 \cdot 3$ is obtained as there are eight carbon atoms in the molecule, and each carbon location is represented by three coordinates x, y, z. In Figure 10.19, we compute the persistent homology of this dataset, obtaining a single connected component, and two significant two-dimensional homology features. See the website for this book [84] for a tutorial that computes these persistent homology barcodes. These homology signatures agree with the homology of the union of a sphere with a Klein bottle, glued together along two circles of singularities. As many shapes have these same homology signatures, the persistent homology does not determine the shape uniquely. However, persistent homology gives a data scientist partial clues towards such a model. The sphere with Klein bottle model is obtained in the papers [336, 337], who furthermore obtain a triangulation of this dataset (a representation of the dataset as a two-dimensional simplicial complex).

It is important to reiterate that while homology is an invariant of a space, it is not a complete invariant. This means that two shapes that can be bent or stretched to get from one to the other necessarily have the same homology groups, but the reverse direction does not hold! Two shapes which have the same homology groups may or may not be deformable to get from one to the

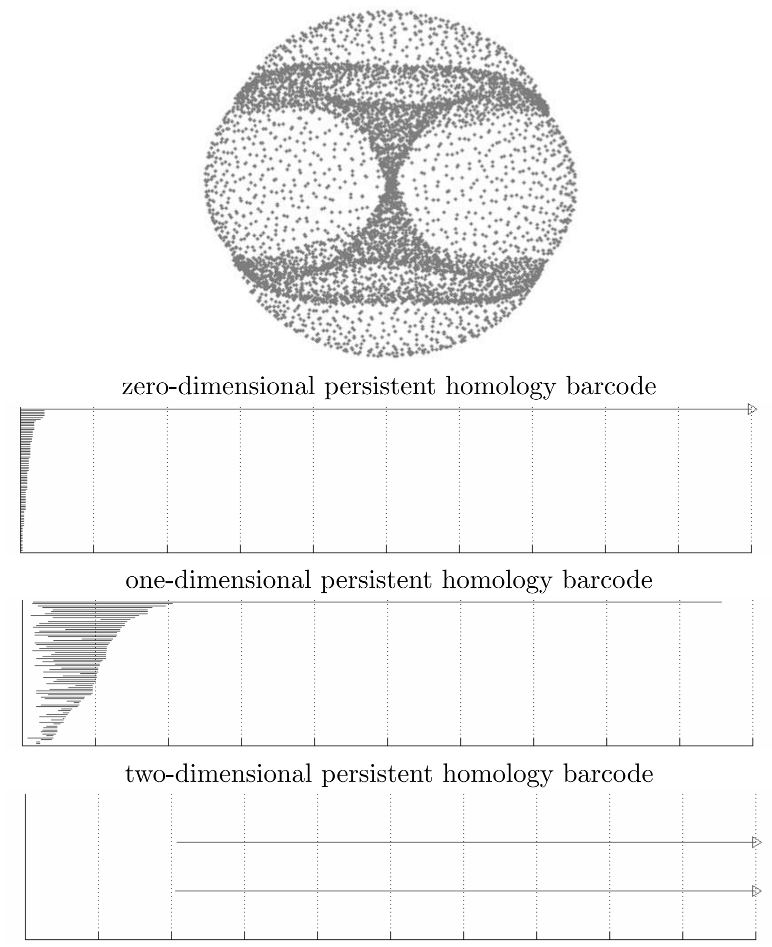


FIGURE 10.19: (Top) A PCA projection of a dataset of different conformations of the cyclo-octane molecule. This shape is a sphere glued to a Klein bottle (the “hourglass”), along two circles of singularity. (Bottom) zero-, one-, and two-dimensional persistent homology barcodes for this dataset. The top image is from [337], used with permission.

other. By analogy, the parity (even or oddness) of an integer gives you partial clues as to what the integer is, though it certainly doesn't determine the number uniquely! This is why persistent homology gives a data practitioner only partial clues (homology groups) as to what the underlying space is; persistent homology does not uniquely identify the underlying space. It is an active area of research to find a geometric model for a dataset that matches computed homology groups.

Chemists are interested not only in the space of conformations of a molecule such as cyclo-octane, but also in the energy associated to such conformations. Indeed, the conformations of low energy are the ones most likely to be found in nature. Furthermore, the energy landscape determines what path the molecule takes in order to transition between two different low-energy states. In order to describe a real-valued energy function on a domain of possible confirmations, one first needs to understand the shape of this domain. The paper [337] furthermore plots the energy function of the cyclo-octane molecule and the transition paths between low-energy configurations; doing so relies on being able to separate the domain of definition into its spherical and Klein bottle pieces.

In the 3×3 image patch example and the cyclo-octane molecule example, we use persistent homology to estimate the homology groups of an unknown underlying space from only a finite dataset sample. There is by now a robust theory bounding the error in such approximations, as described in Appendix 10.10 on the stability of persistent homology.

10.7 Sublevelset persistence

So far in this chapter, we have mainly applied persistent homology to an increasing sequence of spaces of the form

$$\mathrm{VR}(X;r_1) \subseteq \mathrm{VR}(X;r_2) \subseteq \mathrm{VR}(X;r_3) \subseteq \ldots \subseteq \mathrm{VR}(X;r_{m-1}) \subseteq \mathrm{VR}(X;r_m),$$

where X is a dataset, where $r_1 \le r_2 \le r_3 \le \ldots \le r_{m-1} \le r_m$, and where $\mathrm{VR}(X;r_i)$ is the corresponding Vietoris–Rips complex at scale r_i. However, as pointed out in Section 10.6, persistent homology can be applied to any increasing sequence of topological spaces $Y_1 \subseteq Y_2 \subseteq Y_3 \subseteq \ldots \subseteq Y_{m-1} \subseteq Y_m$. Sublevelset persistent homology fits into this framework.

Suppose one is given a topological space Y equipped with a real-valued function $f\colon Y \to \mathbb{R}$. Let $r_1 \le r_2 \le r_3 \le \ldots \le r_{m-1} \le r_m$ be an increasing sequence of real numbers. For each i, we let the i^{th} sublevelset $Y_i = \{y \in Y \mid f(y) \le r_i\}$ consist of all points in Y whose values under f are at most r_i. See for example Figure 10.20 (top), which shows a torus Y equipped with the height function $f\colon Y \to \mathbb{R}$, along with some of its sublevelsets. Since $r_1 \le$

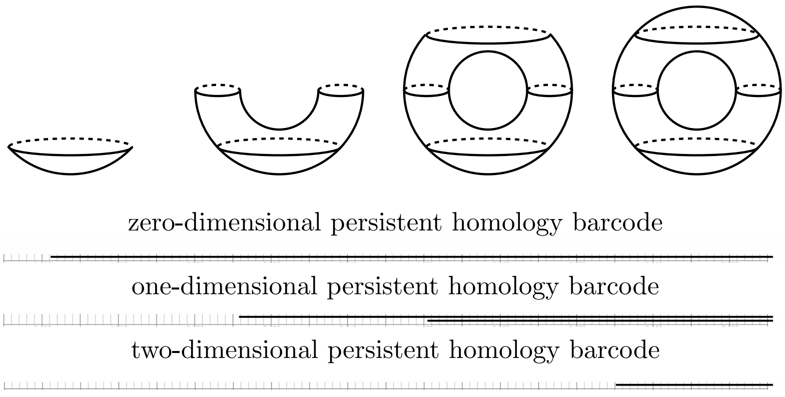

zero-dimensional persistent homology barcode
one-dimensional persistent homology barcode
two-dimensional persistent homology barcode

FIGURE 10.20: (Top) Several sublevelsets for a torus equipped with its height function. (Bottom) The corresponding sublevelset persistent homology barcodes.

$r_2 \leq r_3 \leq \ldots \leq r_{m-1} \leq r_m$, we have an increasing sequence of sublevelsets $Y_1 \subseteq Y_2 \subseteq Y_3 \subseteq \ldots \subseteq Y_{m-1} \subseteq Y_m$, and hence we are in a setting where we can apply persistent homology. The resulting persistent homology is called the *sublevelset persistence* of the space Y equipped with the real-valued function $f: Y \to \mathbb{R}$.

Even if the domain Y is simple topologically, such as the planar domain in the Rayleigh-Bénard convection example in Figure 10.6, sublevelset persistence can reveal interesting geometric and topological properties of a complicated function $f: Y \to \mathbb{R}$.

10.8 Software and exercises

The growth of applied and computational topology is, in part, due to the availability of free and open-source software packages. Two of the original software packages are the Computational Homology Project (CHomP) [350], and a series of PLEX software packages, the latest of which is Javaplex [458]. More recent software packages include GUDHI [461], R-TDA [154, 155], and Ripser [26], which even has a light weight version allowing one to compute persistent homology in an Internet browser. Scikit-TDA [422] features several methods for incorporating topological features in machine learning applications.

We have made some code for persistent homology available at the website for this book [84]. This code relies on the Ripser software package. A tutorial for this code appears on the same website. Below we give a sample of the exercises that are covered in greater detail in that online tutorial.

Exercise 10.5 *Using the dataset available at the GitHub repository, compute the persistent homology barcodes of the image processing dataset described in Sections 10.2.1 and 10.6. Do your barcodes agree with the homology of the three circle model?*

Exercise 10.6 *Using the dataset available at the GitHub repository, compute the persistent homology barcodes of the cyclo-octane molecule dataset described in Sections 10.2.2 and 10.6.*

Exercise 10.7 *Compute the persistent homology of a dataset of your own choosing!*

Exercise 10.8 *Compute the persistent homology of the Vietoris–Rips complex of the five points* $\{(1,0),(1,2),(0,3),(-1,2),(-1,0)\}$ *in the plane. Can you explain all of the persistent homology barcodes that appear?*

Exercise 10.9 *Compute the persistent homology of the Vietoris–Rips complex of an evenly-spaced* 20×20 *grid of 400 points on the flat torus* $S^1 \times S^1$ *in* $\mathbb{R}^4$*, which is the set of all points* $(\cos(\alpha), \sin(\alpha), \cos(\beta), \sin(\beta))$ *with* $\alpha, \beta \in [0, 2\pi)$.

Exercise 10.10 *Compute the persistent homology of the Vietoris–Rips complex of 500 points sampled uniformly at random from the unit sphere* S^2 *in* $\mathbb{R}^3$.

Exercise 10.11 *Select 400 points uniformly at random (or approximately uniformly at random) from the annulus* $\{(x,y) \in \mathbb{R}^2 \mid 0.95^2 \leq x^2 + y^2 \leq 1.05^2\}$. *Compute the persistent homology of its Vietoris–Rips complexes.*

Exercise 10.12 *Select 400 points uniformly at random (or approximately uniformly at random) from the "coconut shell"* $\{(x,y,z) \in \mathbb{R}^3 \mid 0.95^2 \leq x^2 + y^2 + z^2 \leq 1.05^2\}$. *Compute the persistent homology of its Vietoris–Rips complexes.*

Exercise 10.13 *Write a script that will select n even-spaced points from the unit circle in the plane. Compute the persistent homology of the Vietoris–Rips complex of 4, 6, 9, 12, 15, and 20 equally spaced points on the circle.*

Do you get ever homology above dimension 1?

Exercise 10.14 *Find a planar dataset* $Z \subseteq \mathbb{R}^2$ *and a filtration value* r *such that the Vietoris–Rips complex* $\mathrm{VR}(Z; r)$ *has nonzero homology in dimension 2. Do a software computation to confirm your answer.*

Exercise 10.15 *Find a planar dataset* $Z \subseteq \mathbb{R}^2$ *and a filtration value* r *such that the Vietoris–Rips complex* $\mathrm{VR}(Z; r)$ *has nonzero homology in dimension 6. Do a software computation to confirm your answer.*

Exercise 10.16 *Let X be the 8 vertices of the cube in $\mathbb{R}^3$: $X = \{(\pm 1, \pm 1, \pm 1)\}$. Equip X with the Euclidean metric. Compute the persistent homology of the Vietoris–Rips complex of X. Do you get ever homology above dimension 2?*

Exercise 10.17 *One way to produce a torus is to take a square $[0,1] \times [0,1]$ and then identify opposite sides. This is called a flat torus. More explicitly, the flat torus is the quotient space $([0,1] \times [0,1])/ \sim$, where $(0,y) \sim (1,y)$ for all $y \in [0,1]$ and where $(x,0) \sim (x,1)$ for all $x \in [0,1]$. The Euclidean metric on $[0,1] \times [0,1]$ induces a metric on the flat torus. For example, in the induced metric on the flat torus, the distance between $(0,1/2)$ and $(1,1/2)$ is zero, since these two points are identified. The distance between $(1/10,1/2)$ and $(9/10,1/2)$ is $2/10$, by passing through the point $(0,1/2) \sim (1,1/2)$.*

Write a script that selects 400 random points from the square $[0,1] \times [0,1]$ and then computes the 400×400 distance matrix for these points under the induced metric on the flat torus. Compute the persistent homology of this metric space.

Exercise 10.18 *One way to produce a Klein bottle is to take a square $[0,1] \times [0,1]$ and then identify opposite edges, with the left and right sides identified with a twist. This is called a flat Klein bottle. More explicitly, the flat Klein bottle is the quotient space $([0,1] \times [0,1])/ \sim$, where $(0,y) \sim (1,1-y)$ for all $y \in [0,1]$ and where $(x,0) \sim (x,1)$ for all $x \in [0,1]$. The Euclidean metric on $[0,1] \times [0,1]$ induces a metric on the flat Klein bottle. For example, in the induced metric on the flat Klein bottle, the distance between $(0,4/10)$ and $(1,6/10)$ is zero, since these two points are identified. The distance between $(1/10,4/10)$ and $(9/10,6/10)$ is $2/10$, by passing through the point $(0,4/10) \sim (1,6/10)$.*

Write a script that selects 400 random points from the square $[0,1] \times [0,1]$ and then computes the 400×400 distance matrix for these points under the induced metric on the flat Klein bottle. Compute the persistent homology of this metric space.

Change from $\mathbb{F}_2$ coefficients to $\mathbb{F}_3$ (here $\mathbb{F}_3$ is the finite field with three elements) coefficients and see how the persistent homology changes.

Exercise 10.19 *One way to produce a projective plane is to take the unit sphere S^2 in $\mathbb{R}^3$ and then identify antipodal points. More explicitly, the projective plane is the quotient space $S^2/(x \sim -x)$. The Euclidean metric on S^2 induces a metric on the projective plane.*

Write a script that selects 400 random points from the unit sphere S^2 in $\mathbb{R}^3$ and then computes the 400×400 distance matrix for these points under the induced metric on the projective plane. Compute the persistent homology of this metric space.

Change from $\mathbb{F}_2$ to $\mathbb{F}_3$ coefficients and see how the persistent homology changes.

10.9 References

There are by now a large number of survey papers and books on applied topology and topological data analysis. We will mention only a small subset of these, but we encourage the reader to also see the references within. Some excellent survey papers and books on applied topology include [144, 187, 524]. See [80, 186] for survey papers on topological data analysis. Two of the seminal papers on persistent homology include [145] and [523]. The image processing dataset considered in this chapter is introduced in the paper [81], and the cyclo-octane molecule dataset is introduced in the paper [337]. See [467] for an example application of topology to agent-based modeling, as mentioned in Section 10.2.3. Two example papers on the interaction between topological data analysis and machine learning include [5, 73].

Any method in data science needs to be robust to some amount of noise. Indeed, scientific measurements are accurate only up to a certain number of significant figures. Even worse, data entries could be corrupted while being stored or transferred. One of the most important properties of persistent homology is the stability theorem, which says that persistent homology is robust, to some extent, to noise. The machinery behind this theorem, including important notions of distance between datasets (the Hausdorff and Gromov-Hausdorff distances), and notions of distance between persistence diagrams (the bottleneck distance) would have taken us too far astray from our data science goals of the chapter, but the reader may be interested in learning about them in Appendix 10.10.

10.10 Appendix: stability of persistent homology

Persistent homology describes the topology of a single dataset, but it is often important to have methods for comparing different datasets. Figure 10.21 shows two datasets sampled from a circle with noise. The datasets appear similar, but must their persistence barcodes reflect that? The barcodes will certainly not be identical (in particular, the data indicated by $\square$ has fewer points than the one marked with $*$, and so its zero-dimensional persistent homology barcode will have fewer intervals at small scales). Can we quantify how different they are?

To answer these questions, we introduce a distance for persistence diagrams. An important result, called the stability theorem (Theorem 8), guarantees that similar datasets have persistence diagrams which are close with respect to this distance.

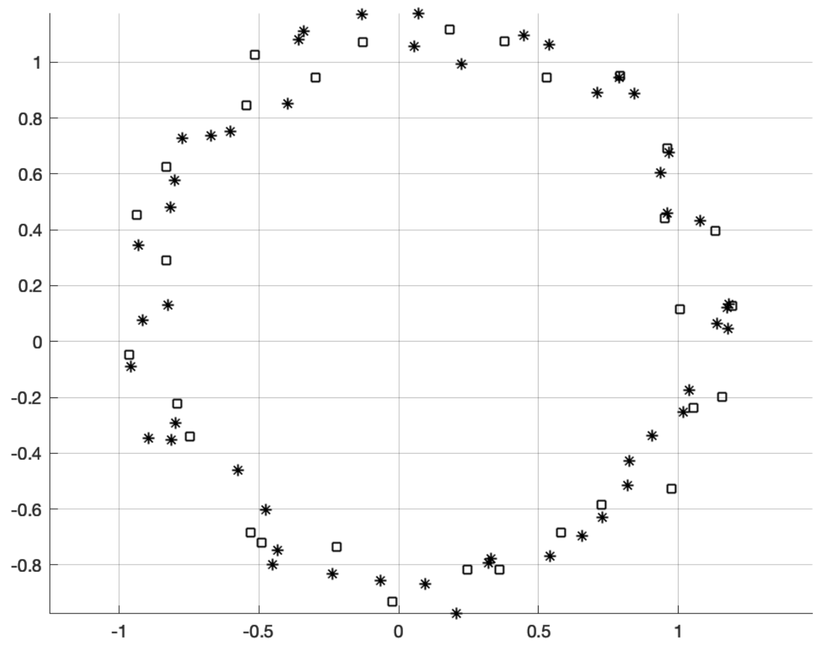


FIGURE 10.21: Two sets of points sampled from a circle with noise.

10.10.1 Distances between datasets

Before comparing persistence diagrams, we will need a precise way of saying how similar two datasets are. Given two finite subsets $U \subset \mathbb{R}^n$ and $V \subset \mathbb{R}^n$, what we need is a way to measure the distance from U to V. This is surprisingly tricky!

We will model a dataset U mathematically as a metric space. That is, for us a dataset U is a set equipped with a notion of distance $d(u, u')$ between any two points $u, u' \in U$ that satisfies the metric space assumptions (such as the triangle inequality, etc.). For simplicity, in this section we will think of all of our datasets U as subsets of Euclidean space, namely $U \subseteq \mathbb{R}^n$, where the distance d on U is simply the restriction of the standard distance function d on Euclidean space $\mathbb{R}^n$.

To begin, let's consider how to measure the distance between a single point, x, and a finite set V in $\mathbb{R}^n$. Intuitively this is the distance between x and the point in V closest to x. Formally,

$$d(x, V) = \min_{v \in V} d(x, v).$$

To extend this to a distance between two sets, U and V, we need to then consider all of the points in U, and so we define

$$\vec{d}(U, V) = \max_{u \in U} \min_{v \in V} d(u, v).$$

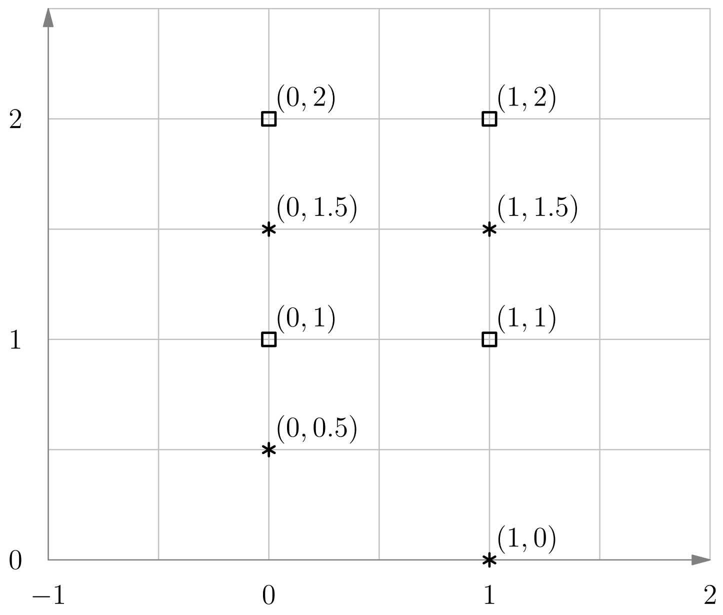


FIGURE 10.22: A set of points U marked with $\square$ and V marked with $*$, in which $\vec{d}(U, V) \neq \vec{d}(V, U)$.

Observe that in Figure 10.22, $\vec{d}(U, V) = 0.5$ and is realized by the distance between $(0, 1)$ and $(0, 1.5)$ (as well as several other pairs of points). However, $\vec{d}(V, U) = 1$ and is realized by the distance between $(1, 0)$ and $(1, 1)$, so the distance $\vec{d}$ is not symmetric. To fix this, we symmetrize $\vec{d}$ to get the *Hausdorff distance*, d_{H}:

$$d_{\mathrm{H}}(U, V) = \max\{\vec{d}(U, V), \vec{d}(V, U)\}.$$

While d_{H} is a candidate for a distance on datasets, it is not yet the correct notion of distance for our purposes. Consider Figure 10.23. It depicts two copies of the same data, translated to different locations in the plane. The location of the data should not matter—a circle is a circle regardless of where it is placed—and since we are doing topology we care only about the shape, not the location. So we need to allow ourselves to align the data as well as possible before computing the Hausdorff distance. This refinement is called the *Gromov-Hausdorff distance*, and it relies on the notion of isometry.

Definition 10.2 *A function f is an isometric embedding of U if for all $u_i, u_j \in U$, we have $d(u_i, u_j) = d(f(u_i), f(u_j))$.*

Definition 10.3 *Let U and V be finite subsets of $\mathbb{R}^n$, let Z be any metric space, and let $f\colon \mathbb{R}^n \to Z$ and $g\colon \mathbb{R}^n \to Z$ be isometric embeddings of U and*

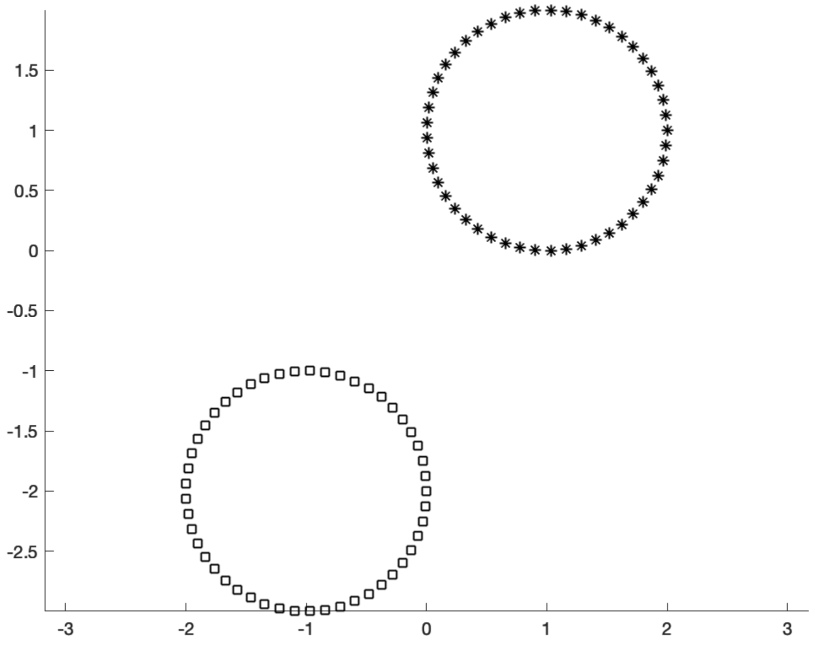


FIGURE 10.23: The same point set as in Figure 10.21, translated to two different locations. The Hausdorff distance is not zero.

V, respectively. The Gromov-Hausdorff distance, $d_{\mathrm{GH}}(U, V)$ is the minimal Hausdorff distance between $f(U)$ and $g(V)$, where the minimum is taken over all possible isometric embeddings and all metric spaces Z.

In many cases the reader can think of the Gromov-Hausdorff distance as the Hausdorff distance computed after aligning the two datasets as well as possible; that is, after rotating, translating, or reflecting U in order to make it line up with V. Exercise 10.23 shows that this is not completely accurate, but it can guide the intuition.

The Gromov-Hausdorff distance is our final measure of the distance between datasets. If $d_{\mathrm{GH}}(U, V) = 0$, then U and V can be "aligned" so that they are exactly the same. It is symmetric (because the Hausdorff distance is symmetric) and $d_{\mathrm{GH}}(U, V) \geq 0$ because it is an extension of the Hausdorff distance. Furthermore, Exercise 10.22 shows that it satisfies the triangle inequality.

Readers who want hands-on experience with the Hausdorff and Gromov-Hausdorff distances may be interested in the following exercises. Alternatively, the reader could skip directly to Section 10.10.2 to learn about the bottleneck distance between persistent diagrams, which will be used in the stability theorems in Section 10.10.3.

Exercise 10.20 *Why is* $\min_{u\in U}\max_{v\in V} d(u,v)$ *not a good measure of distance?*

Exercise 10.21 *What is the Hausdorff distance between the* x*-axis and the* y*-axis in* $\mathbb{R}^2$*? What is the Gromov-Hausdorff distance?*

Exercise 10.22 *Show that the Gromov-Hausdorff distance satisfies the triangle inequality, that is, for any three metric spaces* U*,* V*, and* W*,*

$$d_{\mathrm{GH}}(U,W) \le d_{\mathrm{GH}}(U,V) + d_{\mathrm{GH}}(V,W).$$

Exercise 10.23 *The reader may wonder why we introduce the metric space* Z *in the definition of Gromov-Hausdorff distance since it might appear more natural to consider isometries* f *and* g *from* $\mathbb{R}^n$ *to* $\mathbb{R}^n$*. Such functions are just translations, rotations, and reflections. Here we give an example where the optimal isometric embedding is into a non-Euclidean space. (This exercise was adopted from [342].)*

1. *Let* X *consist of the points* $(-1,0)$*,* $(1,0)$*, and* $(0,\sqrt{3})$ *in* $\mathbb{R}^2$ *(or more generally, an abstract metric space of three points where all distances are* 2*). Let* Y *be a single point. Find the location of* Y *that minimizes the Hausdorff distance* $d_{\mathrm{H}}(X,Y)$*, and determine that distance.*

2. *Now consider an abstract metric space* Z *consisting of points* $\{p,q,r,s\}$ *with distances given by the table:*

$d(-,-)$	p	q	r	s
p	0	2	2	1
q	2	0	2	1
r	2	2	0	1
s	1	1	1	0

   Show that Z *is a metric space.*

3. *Find isometric embeddings* $f\colon X \to Z$ *and* $g\colon Y \to Z$ *such that the Hausdorff distance between* $f(X)$ *and* $g(Y)$ *is smaller than the distance in part 1.*

4. *Prove that there does not exist an isometry from* Z *into* $\mathbb{R}^n$*. (Hint:* p*,* q*, and* r *must form an equilateral triangle. Where can* s *go?)*

10.10.2 Bottleneck distance and visualization

The previous section gave us a notion of distance between datasets. Here we develop a notion of distance to use on persistence diagrams. A persistence diagram consists of a set of points in the plane with coordinates (x_1, x_2) such that $x_2 \ge x_1$. Hypothetically, we could measure distances between persistence diagrams with the Hausdorff distance; however, it is again better to align the sets of points in some way. We are also going to measure distance in the plane differently. Specifically, we want to use the L^∞ or "max" distance.

Definition 10.4 *If $x = (x_1, x_2)$ and $y = (y_1, y_2)$ are points in $\mathbb{R}^2$, then the L^∞ distance between x and y is*

$$\|x - y\|_\infty = \max\{|x_1 - y_1|, |x_2 - y_2|\}.$$

Exercise 10.24 *The unit circle is the set of points x in $\mathbb{R}^2$ with $\|x\| = 1$. Using the same definition with the L^∞ distance to the origin, what does the unit circle look like?*

We can use the L^∞ metric on the plane to define a distance between persistence diagrams. Each point in a persistence diagram represents a topological feature (a loop, enclosed volume, etc.) in the data, and the coordinates of that point represent the birth and death times. To say that two diagrams U and V are similar should mean that each feature in the data represented by U has a corresponding feature in the data of V with similar birth and death times. This suggests we should define a distance between U and V as something like

$$d(U, V) = \min_{\phi} \max_{u \in U}\{\|u - \phi(u)\|_\infty\},$$

where ϕ is a bijective (one-to-one and onto) function $\phi\colon U \to V$.

A bijection is necessary because we should not match multiple features in U to a single feature in V, or vice versa. (To see that this is important: Say U has three features and V has only one, all with the same birth and death values. Then a non-bijective matching would match all of the features in U to the feature in V, and the minimum L^∞ distance would be zero!) However, U and V probably have a different number of features, and thus a different number of points in their diagrams, so there may not be any bijections.

A resolution is found by considering the diagonal in the plane; that is, the points u where $u_1 = u_2$. Topologically such a point would correspond to a feature which is born and dies at exactly the same time. In practice persistence diagrams often have many points very close to the diagonal, due to noise causing many short-lived features. This suggests a fix to the problem of bijections: add as many points as necessary on the diagonal so that U and V have the same number of points. The easiest way to do this is simply to add the entire diagonal to both diagrams, as illustrated in Figure 10.24. With this change, our proposed definition becomes the bottleneck distance.

Definition 10.5 *Let U and V be persistence diagrams, and $\Delta = \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1 = x_2 \geq 0\}$. Let $\tilde{U} = U \cup \Delta$ and $\tilde{V} = V \cup \Delta$. The bottleneck distance between U and V is*

$$d_{\mathrm{B}}(U, V) = \inf_{\phi} \sup_{u \in U} \|u - \phi(u)\|_\infty,$$

where the infimum is taken over all bijections between $\tilde{U}$ and $\tilde{V}$.

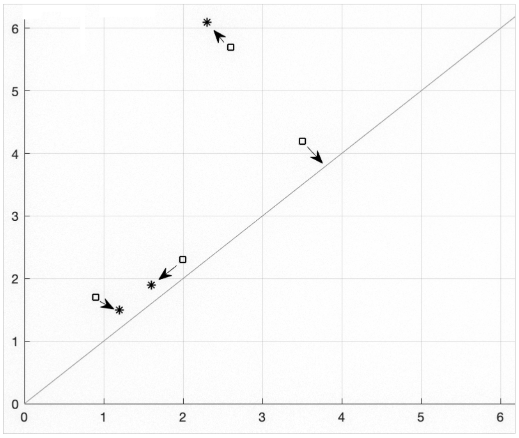


FIGURE 10.24: Two persistence diagrams, indicated by $\square$ and $*$, with arrows showing the optimal matching. Note that one point is matched to a point on the line Δ.

Let us pause to think about what the bottleneck distance means. The distance $\|u - \phi(u)\|_\infty$ is the larger of the difference in birth times between u and the point to which it is matched by ϕ and the difference in death times between these. The infimum over ϕ says that we choose the bijection which minimizes this. So the bottleneck distance describes the maximum amount we have to adjust either birth or death times to match the features in U with those in V.

10.10.3 Stability results

The idea of the following stability result is that if two datasets are close in the Gromov-Hausdorff distance, then the persistence diagrams obtained from them should be close in the bottleneck distance. In particular, two datasets sampled from the same space, but with noise, will typically be close in Gromov-Hausdorff distance. Consequently they will have persistence diagrams with essentially the same features occurring at the same times, and thus have small bottleneck distance. The precise statement of the stability theorem is as follows.

Theorem 8 (Stability theorem for point clouds) *Suppose X and Y are datasets, U is the persistence diagram of the Vietoris–Rips filtration of X, and V is the persistence diagram of the Vietoris–Rips filtration of Y. Then*

$$d_{\mathrm{B}}(U, V) \leq 2d_{\mathrm{GH}}(X, Y).$$

The same theorem holds for other filtrations as well, for example Čech complexes. The proof can be found in [92].

In Section 10.10.1 a filtration by sublevelsets was defined. For such a filtration one might ask if changing the function by which sublevelsets are defined affects the persistence diagram. We can measure the distance between two functions f and g with another type of L^∞ distance.

Definition 10.6 *The L^∞ distance between two functions f and g from $\mathbb{R}^n$ to $\mathbb{R}^m$ is*

$$\|f - g\|_\infty = \sup_{x \in \mathbb{R}^n} \|f(x) - g(x)\|_\infty .$$

The stability theorem for functions says that similar functions give similar persistence diagrams.

Theorem 9 (Stability theorem for functions) *Let $U(f)$ and $U(g)$ be the persistence diagrams corresponding to the sublevelsets of two functions f and g defined on the same space. Then*

$$d_{\mathrm{B}}(U(f), U(g)) \leq \|f - g\|_\infty .$$

This was first shown in [103].

Lastly, it is worth noting that persistent homology is stable with respect to Gromov-Hausdorff distance, but a noisy version of a space is not necessarily close to the original space in the Gromov-Hausdorff sense. The two datasets in Figure 10.25 differ by a single point, but their Gromov-Hausdorff distance is large and their barcodes are quite different! Dealing with more noise that does not lie near the dataset is an ongoing area of research in topological data analysis; see for example [93].

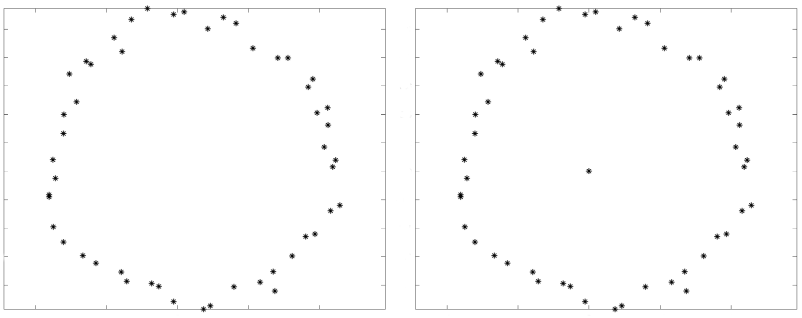


FIGURE 10.25: An example of non-Hausdorff noise. Adding in only a single point at the center creates a new dataset with quite large Hausdorff and Gromov-Hausdorff distances to the original dataset.

Bibliography

[1] P.-A. Absil, Robert Mahony, and Rodolphe Sepulchre. Riemannian geometry of grassmann manifolds with a view on algorithmic computation. *Acta Applicandae Mathematicae*, 80:199–220, 2004.

[2] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, 2009.

[3] Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003.

[4] Henry Adams, Johnathan Bush, and Joshua Mirth. Supplementary resources for Chapter 10 of *Data Science for Mathematicians*. https://github.com/ds4m/topological-data-analysis, 2020.

[5] Henry Adams, Sofya Chepushtanova, Tegan Emerson, Eric Hanson, Michael Kirby, Francis Motta, Rachel Neville, Chris Peterson, Patrick Shipman, and Lori Ziegelmeier. Persistence images: A vector representation of persistent homology. *Journal of Machine Learning Research*, 18(8):1–35, 2017.

[6] Rasmus Bro Age Smilde and Paul Geladi. *Multi-Way Analysis: Applications in the Chemical Sciences*. Wiley, 2004.

[7] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[8] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Rec.*, 27(2):94–105, June 1998.

[9] Alan Agresti. *Categorical Data Analysis*. Wiley, Hoboken, New Jersey, 3rd edition, 2012.

[10] Nir Ailon and Bernard Chazelle. The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on Computing*, 39(1):302–322, 2009.

[11] M. Akritas. *Probability & Statistics for Engineers and Scientists with R.* Pearson, 2015.

[12] Salem Alelyani, Jiliang Tang, and Huan Liu. Feature selection for clustering: A review. In *Data Clustering*, pages 29–60. Chapman and Hall/CRC, 2018.

[13] Aliaga, Cobb, Cuff, Garfield, Gould, Lock, Moore, Rossman, Stephenson, Utts, Velleman, and Witmer. GAISE college report (revised printing). `http://www.amstat.org/education/gaise/`, 2012.

[14] J. Álvarez-Vizoso, Robert Arn, Michael Kirby, Chris Peterson, Bruce Draper, Geometry of curves in $\mathbb{R}^n$ from the local singular value decomposition, *Linear Algebra and its Applications*, Volume 571, pp. 180–202, ISSN 0024-3795, https://doi.org/10.1016/j.laa.2019.02.006, 2019.

[15] Xavier Alvarez-Vizoso, Michael Kirby, and Chris Peterson. Integral invariants from covariance analysis of embedded Riemannian manifolds. *preprint arXiv:1804.10425, submitted*, 2018.

[16] Javier Álvarez-Vizoso, Michael Kirby, Chris Peterson, Manifold curvature learning from hypersurface integral invariants, Linear Algebra and its Applications, Volume 602, pp. 179–205, ISSN 0024-3795, https://doi.org/10.1016/j.laa.2020.05.020, 2020.

[17] Amazon. Amazon Web Services. `https://aws.amazon.com`. Accessed: 2020-01-05.

[18] AMPL Optimization inc. AMPL for courses website. `https://ampl.com/try-ampl/ampl-for-courses/`.

[19] Edgar Anderson. The irises of the gaspe peninsula. *Bull. Am. Iris Soc.*, 59:2–5, 1935.

[20] Howard Anton. *Elementary Linear Algebra.* Wiley, 11th edition, 2013.

[21] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.

[22] Alex Arslan et al. JuliaOpt: Optimization packages for the Julia language. `https://www.juliaopt.org/`.

[23] Sheldon Axler. Down with determinants! *American Mathematical Monthly*, 102:139–154, 1995.

[24] Sherif Azary and Andreas Savakis. Grassmannian sparse representations and motion depth surfaces for 3d action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 492–499, 2013.

[25] R.M. Baron and D.A. Kenny. The moderator-mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *Journal of Personality and Social Psychology*, 51:1173–1182, 1986.

[26] Ulrich Bauer. Ripser: efficient computation of Vietoris–Rips persistence barcodes. *arXiv preprint arXiv:1908.02518*, 2019.

[27] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[28] Peter N. Belhumeur and David J. Kriegman. What is the set of images of an object under all possible illumination conditions? *International Journal of Computer Vision*, 28(3):245–260, 1998.

[29] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[30] Richard E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, NJ, 1961.

[31] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Princeton University Press, Princeton, NJ, 2009.

[32] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*, volume 2. SIAM, 2001.

[33] Yoshua Bengio. MILA and the future of Theano. https://groups.google.com/forum/#!topic/theano-users/7Poq8BZutbY, 2017.

[34] Sandra Benítez-Peña, Rafael Blanquero, Emilio Carrizosa, and Pepa Ramírez-Cobo. Cost-sensitive feature selection for support vector machines. *Computers & Operations Research*, 106:169–178, 2018.

[35] Alex Berg, Jia Deng, and L. Fei-Fei. Large scale visual recognition challenge (ilsvrc), 2010. *URL http://www. image-net. org/challenges/LSVRC*, 3, 2010.

[36] Michael W. Berry and Murray Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. SIAM, 2nd edition, 2005.

[37] Dimitris Berstimas and Nathan Kallus. From predictive to prescriptive analytics. *Management Science*, 2019. Articles in Advance, ISSN 0025-1909 (print), ISSN 1526-5501 (online).

[38] Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.

[39] Dimitris Bertsimas, Arthur Delarue, and Sebastien Martin. Optimizing schools' start time and bus routes. *Proceedings of the National Academy of Sciences*, 116(13):5943–5948, 2019.

[40] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[41] Dimitris Bertsimas and Jack Dunn. *Machine Learning Under a Modern Optimization Lens.* Dynamic Ideas LLC, Charlestown, MA, 2019.

[42] Dimitris Bertsimas, Jack Dunn, and Nishanth Mundru. Optimal prescriptive trees. *INFORMS Journal on Optimization*, 1(2):164–183, 2019.

[43] Dimitris Bertsimas, Jack Dunn, Colin Pawlowski, and Ying Daisy Zhuo. Robust classification. *INFORMS Journal on Optimzation*, 1(1):2–34, 2019.

[44] Dimitris Bertsimas and Angela King. Logistic regression: From art to science. *Statistical Science*, 32(3):367–384, 2017.

[45] Dimitris Bertsimas, Nikita Korolko, and Alexander M. Weinstein. Identifying exceptional responders in randomized trials: An optimization approach. *INFORMS Journal on Optimization*, 2019. To appear. https://pubsonline.informs.org/doi/pdf/10.1287/ijoo.2018.0006.

[46] Dimitris Bertsimas, Colin Pawlowski, and Ying Daisy Zhuo. From predictive methods to missing data imputation: An optimization approach. *Journal of Machine Learning Research*, 18(196):1–39, 2018.

[47] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*, volume 6. Athena Scientific, Belmont, MA, 1997.

[48] J. R. Beveridge, B. A. Draper, J. M. Chang, M. Kirby, H. Kley, and C. Peterson. Principal angles separate subject illumination spaces in YDB and CMU-PIE. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):351–363, Feb 2009.

[49] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[50] JM Bibby, JT Kent, and KV Mardia. *Multivariate Analysis.* Academic Press, London, 1979.

[51] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: Applications to image and text data. In *Knowledge Discovery and Data Mining*, pages 245–250. ACM Press, New York, 2001.

[52] Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016.

[53] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, U.K., 1995.

[54] Natalie J. Blades, G. Bruce Schaalje, and William F. Christensen. The second course in statistics: Design and analysis of experiments? *The American Statistician*, 69(4):326–333, 2015.

[55] Anthony Blaom, Franz Kiraly, Thibaut Lienart, and Sebastian Vollmer. alan-turing-institute/MLJ.jl v0.5.3: A Julia machine learning framework, November 2019.

[56] Bokeh Development Team. Bokeh: Python library for interactive visualization. https://bokeh.pydata.org/en/latest/, 2018.

[57] Sean Borman. The expectation maximization algorithm: a short tutorial. http://www.seanborman.com/publications, 2004.

[58] George E. P. Box. Robustness in the strategy of scientific model building. In Robert L. Launer and Graham N. Wilkinson, editors, *Robustness in Statistics*, pages 201–236, Academic Press, NY, 1979.

[59] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.

[60] Pratik Prabhanjan Brahma, Dapeng Wu, and Yiyuan She. Why deep learning works: A manifold disentanglement perspective. *IEEE transactions on neural networks and learning systems*, 27(10):1997–2008, 2015.

[61] Andrew Bray and Mine Çetinkaya Rundel. "Labs for R" blog post. *The OpenIntro Project*, July 2016.

[62] T.C. Bressoud and D. White. *Introduction to Data Systems.* Springer-Verlag, 2020.

[63] T.C. Bressoud and G. Thomas. A novel course in data systems with minimal prerequisites. *SIGCSE Conference: the 50th ACM Technical Symposium*, 2019.

[64] Guy Brock, Vasyl Pihur, Susmita Datta, and Somnath Datta. clValid: An R package for cluster validation. *Journal of Statistical Software*, 25(4):1–22, 2008.

[65] D. S. Broomhead, R. Jones, and G. P. King. Topological dimension and local coordinates from time series data. *J. Phys. A: Math. Gen*, 20:L563–L569, 1987.

[66] D.S. Broomhead, R. Indik, A.C. Newell, and D.A. Rand. Local adaptive Galerkin bases for large-dimensional dynamical systems. *Nonlinearity*, 4:159–197, 1991.

[67] D.S. Broomhead and Gregory P. King. Extracting qualitative dynamics from experimental data. *Physica*, 20 D:217–236, 1986.

[68] D.S. Broomhead and M. Kirby. A new approach for dimensionality reduction: Theory and algorithms. *SIAM J. of Applied Mathematics*, 60(6):2114–2142, 2000.

[69] D.S. Broomhead and M. Kirby. The Whitney reduction network: a method for computing autoassociative graphs. *Neural Computation*, 13:2595–2616, 2001.

[70] S. Allen Broughton and Kurt Bryan. *Discrete Fourier Analysis and Wavelets: Applications to Signal and Image Processing*. Wiley, Hoboken, NJ, 2nd edition, 2018.

[71] M. W. Brown, S. Martin, S. N. Pollock, E. A. Coutsias, and J. P. Watson. Algorithmic dimensionality reduction for molecular structure analysis. *Journal of Chemical Physics*, 129:064118, 2008.

[72] Kurt Bryan and Tanya Leise. The $25,000,000,000 eigenvector: The linear algebra behind Google. *SIAM Review*, 48(3):569–581, 2006.

[73] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *The Journal of Machine Learning Research*, 16(1):77–102, 2015.

[74] Christopher J. C. Burges. Dimension reduction: A guided tour. *Foundations and Trends in Machine Learning*, 2(4):275–365, 2010.

[75] Andriy Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019. (freely available on the web).

[76] Alice Calaprice, editor. *The Ultimate Quotable Einstein*. Princeton University Press, 2010.

[77] California ISO. Homepage, 2019. http://www.caiso.com/Pages/default.aspx.

[78] Francesco Camastra. Data dimensionality estimation methods: a survey. *Pattern recognition*, 36(12):2945–2954, 2003.

[79] Francesco Camastra and Antonino Staiano. Intrinsic dimension estimation: Advances and open problems. *Information Sciences*, 328:26–41, 2016.

[80] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[81] Gunnar Carlsson, Tigran Ishkhanov, Vin de Silva, and Afra Zomorodian. On the local behavior of spaces of natural images. *International Journal of Computer Vision*, 76:1–12, 2008.

[82] Miguel Á. Carreira-Perpiñán. A review of dimension reduction techniques. Technical report, Department of Computer Science, University of Sheffield, 1997.

[83] Kevin M Carter, Raviv Raich, and Alfred O Hero III. On local intrinsic dimension estimation and its applications. *IEEE Transactions on Signal Processing*, 58(2):650–663, 2009.

[84] Nathan Carter et al. Data Science for Mathematicians: online resources for the text. https://ds4m.github.io/, 2020.

[85] George Casella and Edward I. George. Explaining the Gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[86] Filippo Castiglione, Francesco Pappalardo, Massimo Bernaschi, and Santo Motta. Optimization of HAART with genetic algorithms and agent-based models of HIV infection. *Bioinformatics*, 23(24):3350–3355, 2007.

[87] John Chakerian and Susan Holmes. Computational tools for evaluating phylogenetic and hierarchical clustering trees. *Journal of Computational and Graphical Statistics*, 21(3):581–599, 2012.

[88] Rudrasis Chakraborty and Baba C Vemuri. Recursive frechet mean computation on the grassmannian and its applications to computer vision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4229–4237, 2015.

[89] B. Chance, S. Cohen, S. Grimshaw, J. Hardin, T. Hesterberg, R. Hoerl, N. Horton, M. Mallone, R. Nichols, and D. Nolan. Curriculum guidelines for undergraduate programs in statistical science. http://www.amstat.org/education/pdfs/guidelines2014-11-15.pdf, 2014.

[90] Jen-Mei Chang. *Classification on the Grassmannians: Theory and Applications*. Colorado State University, 2008.

[91] Jen-Mei Chang, Michael Kirby, Holger Kley, Chris Peterson, Bruce Draper, and J Ross Beveridge. Recognition of digital images of the human face at ultra low resolution via illumination spaces. In *Computer Vision–ACCV 2007*, pages 733–743. Springer, 2007.

[92] Frédéric Chazal, David Cohen-Steiner, Marc Glisse, Leonidas J Guibas, and Steve Y Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry*, pages 237–246. ACM, 2009.

[93] Frédéric Chazal, David Cohen-Steiner, and Quentin Mérigot. Geometric inference for probability measures. *Foundations of Computational Mathematics*, 11(6):733–751, 2011.

[94] Sofya Chepushtanova and Michael Kirby. Sparse Grassmannian embeddings for hyperspectral data representation and classification. *IEEE Geoscience and Remote Sensing Letters*, 14(3):434–438, March 2017.

[95] Sofya Chepushtanova, Michael Kirby, Chris Peterson, and Lori Ziegelmeier. Persistent homology on Grassmann manifolds for analysis of hyperspectral movies. In *Computational Topology in Image Context*, pages 228–239. Springer International Publishing. Cham, Switzerland.

[96] Chess.com member "pete". Lc0 Wins Computer Chess Championship, Makes History. https://www.chess.com/news/view/lc0-wins-computer-chess-championship-makes-history, 2019.

[97] Bertrand Clarke, Ernest Fokoue, and Hao Helen Zhang. *Principles and Theory for Data Mining and Machine Learning*. Springer Science & Business Media, New York, NY, 2009.

[98] William S. Cleveland. Data science: An action plan for expanding the technical areas of the field of statistics. *International Statistical Review*, 69(1):21–26, 2001.

[99] G. Cobb. The introductory statistics course: A ptolemaic curriculum? *Technology Innovations in Statistics Education*, 1:1, 2007.

[100] G. Cobb. Teaching statistics: Some important tensions. *Chilean Journal of Statistics*, 2(1):31–62, April 2011.

[101] G. Cobb, B. Hartlaub, J. Legler, R. Lock, T. Moore, A. Rossman, A. Cannon, and J. Witmer. *Stat2: Building Models for a World of Data*. WH Freeman, New York, NY, 2012.

[102] James J. Cochran, editor. *INFORMS Analytics Body of Knowledge*. John Wiley and Sons, Inc., Hoboken, NJ, 2019.

[103] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[104] COIN-OR Foundation. COIN-OR Website. https://www.coin-or.org.

[105] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[106] Ronan Collobert et al. Torch5 homepage (archived). https://web.archive.org/web/20081205015134/http://torch5.sourceforge.net/, 2008.

[107] Drew Conway. The Data Science Venn Diagram. http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram, March 2013.

[108] John H Conway, Ronald H Hardin, and Neil JA Sloane. Packing lines, planes, etc.: Packings in Grassmannian spaces. *Experimental Mathematics*, 5(2):139–159, 1996.

[109] Carlos Cordoba et al. Spyder: The Scientific Python Development Editor. https://www.spyder-ide.org. Accessed: 2020-01-05.

[110] R. V. Craiu and L. Sun. Choosing the lesser evil: trade-off between false discovery rate and non-discovery rate. *Statistica Sinica*, 18:861–879, 2008.

[111] Daniel Crevier. *AI: The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, New York, 1993.

[112] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[113] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[114] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The NEOS server. *IEEE Journal on Computational Science and Engineering*, 5(3):68–75, 1998.

[115] George B. Dantzig, Alex Orden, and Philip Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

[116] David Darmon, Elisa Omodei, and Joshua Garland. Followers are not enough: A multifaceted approach to community detection in online social networks. *PloS one*, 10(8):e0134860, 2015.

[117] Sanjoy Dasgupta. Learning mixtures of Gaussians. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, page 634. IEEE Computer Society, 1999.

[118] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures and Algorithms*, 22(1):60–65, 2003.

[119] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*, 1st edn. John Wiley & Sons, Inc., New York, NY, 2003.

[120] James Davis, Guillermo Gallego, and Huseyin Topaloglu. Assortment planning under the multinomial logit model with totally unimodular constraint structures. Unpublished manuscript, 2013.

[121] Marcos López de Prado. *Advances in Financial Machine Learning.* Wiley, 2018.

[122] Vin de Silva and Joshua B. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Stanford University, 2004.

[123] Angela Dean, Daniel Voss, and Danel Draguljic. *Design and Analysis of Experiments.* Springer, New York, 2nd edition, 2017.

[124] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, Curran Associates, Inc., Red Hook, NY, 2012.

[125] M. Dell, B.F. Jones, and B.A. Olken. What do we learn from the weather? the new climate-economy literature. *Journal of Economic Literature*, 52:740–798, 2014.

[126] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[127] Jacob Devlin and Ming-Wei Chang. Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing. https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html, 2018.

[128] Persi Diaconis. Group representations in probability and statistics. *Lecture notes-monograph series*, 11:i–192, 1988.

[129] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[130] Robert P. Dobrow. *Introduction to Stochastic Processes with R.* John Wiley & Sons, Inc., Hoboken, New Jersey, 2016.

[131] Elizabeth D. Dolan. The NEOS Server 4.0 administrative guide. Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

[132] David Donoho. 50 years of data science. *Journal of Computational and Graphical Statistics*, 26(4):745–766, 2017.

[133] Matt Dowle and Arun Srinivasan. *data.table: Extension of 'data.frame'*, 2019. R package version 1.12.2.

[134] Allen Downey. *Think Python.* O'Reilly Media, 2nd edition, updated for Python 3. Sebastopol, California, 2016.

[135] Adrian A. Dragulescu and Cole Arendt. *xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files*, 2018. R package version 0.6.1.

[136] Dheeru Dua and Casey Graff. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2017.

[137] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[138] Jennifer G Dy and Carla E Brodley. Feature selection for unsupervised learning. *Journal of Machine Learning Research*, 5(Aug):845–889, 2004.

[139] Jean Pierre Eckmann and David Ruelle. Fundamental limitations for estimating dimensions and lyapunov exponents in dynamical system. *Physica D: Nonlinear Phenomena*, 56:185–187, 1992.

[140] Dirk Eddelbuettel and Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011.

[141] Alan Edelman, Tomás A Arias, and Steven T Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[142] Alan Edelman and Yuyang Wang. The GSVD: Where are the ellipses?, matrix trigonometry, and more. *arXiv preprint arXiv:1901.00485*, 2019.

[143] Edelsbrunner, Letscher, and Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28(4):511–533, 2002.

[144] Herbert Edelsbrunner and John L Harer. *Computational Topology: An Introduction*. American Mathematical Society, Providence, RI, 2010.

[145] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 454–463. IEEE, 2000.

[146] Albert Einstein. On the method of theoretical physics. The Herbert Spencer Lecture, Oxford University, 1933.

[147] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[148] Brian Everitt and Torsten Hothorn. *An Introduction to Applied Multivariate Analysis with R*. Springer Science & Business Media, New York, NY, 2011.

[149] Brian S Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. Cluster analysis. 330 pages, John Wiley & Sons Ltd., Chichester, West Sussex, UK, 2011.

[150] M. Ezekiel. *Methods of Correlation Analysis.* Wiley, New York, 1930.

[151] Kenneth Falconer. *Fractal geometry: mathematical foundations and applications.* John Wiley & Sons, 2004.

[152] Kenneth J. Falconer. Dimensions and measures of quasi self-similar sets. *Proc. Amer. Math. Soc. 106 (1989), 543-554*, 1989.

[153] Elin Farnell. *Algorithms and geometric analysis of data sets that are invariant under a group action.* PhD thesis, Colorado State University, 2010.

[154] Brittany T. Fasy, Jisu Kim, Fabrizio Lecci, Clément Maria, David L. Millman, Vincent Rouvreau. The included GUDHI is authored by Clément Maria, Dionysus by Dmitriy Morozov, PHAT by Ulrich Bauer, Michael Kerber, and Jan Reininghaus. *TDA: Statistical Tools for Topological Data Analysis*, 2019. R package version 1.6.9.

[155] Brittany Terese Fasy, Jisu Kim, Fabrizio Lecci, and Clément Maria. Introduction to the R package TDA. *arXiv preprint arXiv:1411.1830*, 2014.

[156] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

[157] Jacob Feldman, Dennis Zhang, Xiaofei Liu, and Nannan Zhang. Taking assortment optimization from theory to practice: Evidence from large field experiments on Alibaba. Unpublished manuscript, 2018.

[158] Ronen Feldman and James Sanger. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data.* Cambridge University Press. 2006.

[159] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[160] Lloyd Fisher and John W. Van Ness. Admissible clustering procedures. *Biometrika*, 58(1):91–104, 1971.

[161] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[162] Frederic Font, Gerard Roma, and Xavier Serra. 2013. Freesound technical demo. In *Proceedings of the 21st ACM International Conference on Multimedia (MM'13)*. Association for Computing Machinery, New York, NY, USA, 411–412. DOI:https://doi.org/10.1145/2502081.2502245.

[163] Institute for Operations Research and the Management Sciences. Operations research and analytics. `https://www.informs.org/Explore/Operations-Research-Analytics`, 2019.

[164] Python Software Foundation. Python: A dynamic, open source programming language. `https://www.python.org`.

[165] Robert Fourer. Linear programming software survey. *OR/MS Today*, 46(3):51–53, 2019. `https://doi.org/10.1287/orms.2019.03.05`.

[166] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* 6th edition, Brooks/Cole, Belmont, California, 2009.

[167] Chris Fraley and Adrian E Raftery. Mclust: Software for model-based cluster analysis. *Journal of Classification*, 16(2):297–306, 1999.

[168] Chris Fraley and Adrian E Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, 97(458):611–631, 2002.

[169] Daniel Freund, Shane G. Henderson, and David B. Shmoys. Minimizing multimodular functions and allocating capacity in bike-sharing systems. In Friedrich Eisenbrand and Jochen Koenemann, editors, *Integer Programming and Combinatorial Optimization*, pages 186–198. Springer International Publishing, Cham, Switzerland, 2017.

[170] Daniel Freund, Ashkan Norouzi-Fard, Alice J. Paul, Carter Wang, Shane G. Henderson, and David B. Shmoys. *Analytics for the Sharing Economy: Mathematics, Engineering and Business Perspectives*, chapter Data-Driven Rebalancing Methods for Bike-Share Systems. Springer, 2019. Forthcoming.

[171] J.E.F. Friedl. *Mastering Regular Expressions: Powerful Techniques for Perl and Other Tools.* A Nutshell Handbook. O'Reilly, Sebastopol, California, 1997.

[172] Jerome H. Friedman and Jacqueline J. Meulman. Clustering objects on subsets of attributes (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(4):815–849, 2004.

[173] Audrey Qiuyan Fu, Steven Russell, Sarah J Bray, Simon Tavaré, et al. Bayesian clustering of replicated time-course gene expression data with weak signals. *The Annals of Applied Statistics*, 7(3):1334–1361, 2013.

[174] Ekaterina Galkina Cleary, Jennifer M. Beierlein, Navleen Surjit Khanuja, Laura M. McNamee, and Fred D. Ledley. Contribution of NIH funding to new drug approvals 2010–2016. *Proceedings of the National Academy of Sciences*, 115(10):2329–2334, 2018.

[175] Kyle A. Gallivan, Anuj Srivastava, Xiuwen Liu, and Paul Van Dooren. Efficient algorithms for inferences on grassmann manifolds. In *IEEE Workshop on Statistical Signal Processing, 2003*, pages 315–318. IEEE, 2003.

[176] Michael R. Garey and David S. Johnson. *Computers and Intractability*, volume 29. WH Freeman, New York, 2002.

[177] Karen Moffett Gary W. Stuart and Jeffery J. Leader. A comprehensive vertebrate phylogeny using vector representations of protein sequences from whole genomes. *Molecular Biology and Evolution*, 19(4):554–562, 2002.

[178] Assad, A. A., Gass, S. I. *An Annotated Timeline of Operations Research: An Informal History*. Netherlands: Springer, 2005.

[179] Jason Gauci, Edoardo Conti, and Kittipat Virochsiri. Horizon: The first open source reinforcement learning platform for large-scale products and services, 2018. https://code.fb.com/ml-applications/horizon/.

[180] Stern, H. S., Rubin, D. B., Vehtari, A., Carlin, J. B., Dunson, D. B., Gelman, A. *Bayesian Data Analysis*, Third Edition. United Kingdom: Taylor & Francis, 2014.

[181] Andrew Gelman and Xiao-Li Meng. *Applied Bayesian Modeling and Causal Inference from Incomplete-Data Perspectives*. Wiley, 1st edition, 2004.

[182] Andrew Gelman and Deborah Nolan. *Teaching Statistics: A Bag of Tricks*. Oxford University Press, 2nd edition, 2017.

[183] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):643–660, June 2001.

[184] Tomojit Ghosh, Michael Kirby, and Xiaofeng Ma. *Dantzig-Selector Radial Basis Function Learning with Nonconvex Refinement*, pages 313–327. Springer International Publishing, Cham, Switzerland, 2017.

[185] Tomojit Ghosh, Xiaofeng Ma, and Michael Kirby. New tools for the visualization of biological pathways. *Methods*, 132:26–33, 2018.

[186] Robert Ghrist. Barcodes: The persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

[187] Robert W Ghrist. *Elementary Applied Topology*, volume 1. Createspace Seattle, 2014.

[188] Javier Girón, Josep Ginebra, and Alex Riba. Bayesian analysis of a multinomial sequence and homogeneity of literary style. *The American Statistician*, 59(1):19–30, 2005.

[189] GitHub staff et al. Atom: A hackable text editor for the 21st century. `https://atom.io`. Accessed: 2020-01-31.

[190] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, Sardinia, Italy, 2010.

[191] Ram Gnanadesikan, Jon R Kettenring, and Shiao Li Tsao. Weighting and selection of variables for cluster analysis. *Journal of Classification*, 12(1):113–136, 1995.

[192] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*, 4th edition. Johns Hopkins University Press, Baltimore, 2013.

[193] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4), 2015.

[194] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[195] Google. Google Cloud. `https://cloud.google.com`. Accessed: 2020-01-05.

[196] James Gosling, Bill Joy, Guy L. Steele, Gilad Bracha, and Alex Buckley. *The Java Language Specification, Java SE 8 Edition*. Addison-Wesley Professional, Harlow, 1st edition, 2014.

[197] John C Gower. A general coefficient of similarity and some of its properties. *Biometrics*, pages 857–871, 1971.

[198] John C Gower and Gavin JS Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 18(1):54–64, 1969.

[199] P. Grassberger and I. Procaccia. Measuring the strangeness of strange attractors. *Physica D*, 9:189, 1983.

[200] Misha Gromov. Geometric, algebraic, and analytic descendants of nash isometric embedding theorems. *Bulletin of the American Mathematical Society*, 54(2):173–245, 2017.

[201] William Gropp and Jorge J. Moré. Optimization environments and the NEOS server. In Martin D. Buhman and Arieh Iserles, editors, *Approximation Theory and Optimization*, pages 167–182. Cambridge University Press, 1997.

[202] Oleksandr Grygorash, Yan Zhou, and Zach Jorgensen. Minimum spanning tree based clustering algorithms. In *2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, pages 73–81. IEEE, Arlington, VA, 2006.

[203] Matthias Günther. Isometric embeddings of riemannian manifolds. In *Proceedings of the International Congress of Mathematicians*, volume 1, pages 1137–1143, 1991.

[204] Arpit Gupta. How alexa is learning to converse more naturally. https://developer.amazon.com/blogs/alexa/post/15bf7d2a-5e5c-4d43-90ae-c2596c9cc3a6/how-alexa-is-learning-to-converse-more-naturally, 2018.

[205] Isabelle Guyon, Ulrike Von Luxburg, and Robert C Williamson. Clustering: Science or art. In *NIPS 2009 Workshop on Clustering Theory*, Vancouver, BC, pages 1–11, 2009.

[206] Michael Hahsler, Matthew Piekenbrock, S Arya, and D Mount. dbscan: Density based clustering of applications with noise (dbscan) and related algorithms. *R package version*, pages 1–0, 2017.

[207] David J. Hand. Measuring classifier performance: A coherent alternative to the area under the roc curve. *Machine Learning*, 77:103–123, 2009.

[208] Mehrtash T Harandi, Conrad Sanderson, Sareh Shirazi, and Brian C Lovell. Graph embedding discriminant analysis on grassmannian manifolds for improved image set matching. In *CVPR 2011*, pages 2705–2712. IEEE, 2011.

[209] Frank E. Harrell. *Regression Modeling Strategies With Applications to Linear Models, Logistic Regression, and Survival Analysis.* Springer, New York, 2001.

[210] David Harrison and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, 1978.

[211] William E. Hart, Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. *Pyomo: Optimization Modeling in Python*, volume 67. Springer Science & Business Media, New York, second edition, 2017.

[212] William E. Hart, Jean-Paul Watson, and David L. Woodruff. Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

[213] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer Science+Business Media, LLC, New York, NY, 2009.

[214] Jun He, Laura Balzano, and Arthur Szlam. Incremental gradient on the grassmannian for online foreground and background separation in subsampled video. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1568–1575. IEEE, Providence, RI, 2012.

[215] D. Hedeker and R. D. Gibbons. *Longitudinal Data Analysis, 1st edition.* Wiley-Interscience, Providence, RI, 2006.

[216] Rainer Hegger and Holger Kantz. Improved false nearest neighbor method to detect determinism in time series data. *Physical Review E*, 60(4):4970, 1999.

[217] T. C. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. *The American Statistician*, 69(4):371–386, 2015.

[218] Tomoyuki Higuchi. Approach to an irregular time series on the basis of the fractal theory. *Physica D: Nonlinear Phenomena*, 31(2):277–283, 1988.

[219] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research.* McGraw-Hill, Providence, RI, 9th edition, 2010.

[220] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[221] J-B Hiriart-Urruty. From convex optimization to nonconvex optimization. Necessary and sufficient conditions for global optimality. In *Nonsmooth optimization and related topics*, pages 219–239. Springer, Providence, RI, 1989.

[222] M.W. Hirsch. *Differential Topology.* Graduate Texts in Mathematics 33. Springer-Verlag, Providence, RI, 1976.

[223] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, Providence, RI, 1995.

[224] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[225] Robert Hogg, Elliot Tanis, and Dale Zimmerman. *Probability and Statistical Inference*. Pearson, Providence, RI, 9th edition, 2014.

[226] David I Holmes. A stylometric analysis of mormon scripture and related texts. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 155(1):91–120, 1992.

[227] Yi Hong, Roland Kwitt, Nikhil Singh, Brad Davis, Nuno Vasconcelos, and Marc Niethammer. Geodesic regression on the grassmannian. In *13th European Conference on Computer Vision*, pages 632–646. Zurich, Switzerland, Springer, 2014.

[228] Jan A. C. Hontelez, Mark N. Lurie, Till Bärnighausen, Roel Bakker, Rob Baltussen, Frank Tanser, Timothy B. Hallett, Marie-Louise Newell, and Sake J. de Vlas. Elimination of HIV in South Africa through expanded access to antiretroviral therapy: A model comparison study. *PLoS Medicine*, 10(10):e1001534, 2013.

[229] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1994.

[230] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2nd edition, 2012.

[231] N. J. Horton, E. R. Brown, and L. Qian. Use of R as a toolbox for mathematical statistics exploration. *The American Statistician*, 58(4), 2004.

[232] Nicholas J Horton and Ken P Kleinman. Much ado about nothing. *The American Statistician*, 61(1):79–90, 2007. PMID: 17401454.

[233] Jinggang Huang, Ann B Lee, and David Bryant Mumford. Statistics of range images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 324–332, 2000.

[234] Yuping Huang, Panos M. Pardalos, and Qipeng P. Zheng. *Electrical Power Unit Commitment: Deterministic and Two-Stage Stochastic Programming Models and Algorithms*. SpringerBriefs in Energy. Springer, New York, 2017.

[235] Zhiwu Huang, Jiqing Wu, and Luc Van Gool. Building deep networks on grassmann manifolds. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[236] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[237] JP Huke. Embedding nonlinear dynamical systems: A guide to Takens' theorem. Manchester Institute for Mathematical Sciences, University of Manchester, http://eprints.maths.manchester.ac.uk/, 2006.

[238] D. Hundley and M. Kirby. Estimation of topological dimension. In *Proceedings of the Third SIAM International Conference on Data Mining*, pages 194–202, San Fransico, 2001.

[239] D. Hundley, M. Kirby, and M. Anderle. Blind source separation using the maximum signal fraction approach. *Signal Processing*, 82(10):1505–1508, 2002.

[240] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[241] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15 – 17, 1976.

[242] Kaggle Inc. Kaggle. `https://www.kaggle.com`. Accessed: 2020-01-05.

[243] Alan Julian Izenman. Modern multivariate statistical techniques. *Regression, classification and manifold learning*, 2008.

[244] Julien Jacques and Christophe Biernacki. Model-based clustering for multivariate partial ranking data. *Journal of Statistical Planning and Inference*, 149:201–217, 2014.

[245] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[246] Prateek Jain and Purushottam Kar. Non-convex optimization for machine learning. *Foundations and Trends in Machine Learning*, 10(3-4):142–336, 2017.

[247] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer, New York, 2013.

[248] Steven Janke and Frederick Tinsley. *Introduction to Linear Models and Statistical Inference*. Wiley-Interscience, 1st edition, 2005.

[249] P. Jarvis, S. Wolfe, M. Sierhuis, R. Nado, and F. Y. Enomoto. Agent-based modeling and simulation of collaborative air traffic flow management using Brahms. *SAE International Journal of Aerospace*, 3(1):39–45, 2010.

[250] E.R. Jessup and James Martin. Taking a new look at the latent semantic analysis approach to information retrieval. In Michael Berry, editor, *Computational Information Retrieval*, pages 121–144, SIAM, Philadelphia, 2001.

[251] JetBrains s.r.o. PyCharm: The Python IDE for Professional Developers. `https://www.jetbrains.com/pycharm`. Accessed: 2020-01-05.

[252] Nanjing Jian, Daniel Freund, Holly M. Wiberg, and Shane G. Henderson. Simulation optimization for a large-scale bike-sharing system. In *Proceedings of the 2016 Winter Simulation Conference*, pages 602–613. IEEE Press, 2016.

[253] William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into Hilbert space. In *Contemporary Mathematics*, volume 26, pages 189–206, 1984.

[254] I.T. Jolliffe. *Principal Component Analysis.* Springer, 2nd edition, 2002.

[255] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[256] D. Kaplan. *Statistical Modeling: A Fresh Approach.* Ingram, 2009.

[257] D. Kaplan. *Data Computing: An Introduction to Wrangling and Visualization with R.* Project MOSAIC, 2015.

[258] Rasa Karbauskaitė and Gintautas Dzemyda. Fractal-based methods as a technique for estimating the intrinsic dimensionality of high-dimensional data: A survey. *Informatica*, 27:257–281, 01 2016.

[259] K. Karhunen. Über lineare methoden in der wahrscheinlichkeitsrechnung. *Ann. Acad. Sci. Fennicae*, 37, 1946.

[260] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 302–311. ACM, 1984.

[261] Leonard Kaufman and Peter J. Rousseeuw. Clustering large applications (program clara). *Finding Groups in Data: An Introduction to Cluster Analysis*, pages 68–125, John Wiley and Sons, Inc., Hoboken, New Jersey, 1990.

[262] Leonard Kaufman and Peter J. Rousseeuw. Monothetic analysis (program mona). *Finding Groups in Data: An Introduction to Cluster Analysis*, pages 68–125, John Wiley and Sons, Inc., Hoboken, New Jersey, 1990.

[263] Leonard Kaufman and Peter J. Rousseeuw. Partitioning around medoids (program pam). *Finding Groups in Data: An Introduction to Cluster Analysis*, pages 68–125, John Wiley and Sons, Inc., Hoboken, New Jersey, 1990.

[264] Balázs Kégl. Intrinsic dimension estimation using packing numbers. In *Advances in Neural Information Processing Systems*, pages 697–704, 2003.

[265] Matthew B Kennel, Reggie Brown, and Henry DI Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical Review A*, 45(6):3403, 1992.

[266] Leonid G Khachiyan. A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093–1096, 1979.

[267] W. Kim, B y Choi, E k Hong, S k Kim, and D. Lee. A taxonomy of dirty data. *Data Mining and Knowledge Discovery*, 7:81–99, 2003.

[268] M. Kirby. *Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns.* Wiley, 2001.

[269] M. Kirby and L. Sirovich. Application of the Karhunen-Loève procedure for the characterization of human faces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(1):103–108, 1990.

[270] Michael Kirby. *Linear Algebra for Data Science* (forthcoming).

[271] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016.

[272] Donald E. Knuth. Literate programming. *Comput. J.*, 27(2):97–111, May 1984.

[273] Eric D Kolaczyk. *Statistical Analysis of Network Data: Methods and Models.* Springer Publishing Company, Incorporated, 2009.

[274] Tamara G. Kolda and Dianne P. O'Leary. A semidiscrete matrix decomposition for latent semantic indexing in information retrieval. *ACM Transactions on Information Systems*, 16(4):322–346, 1998.

[275] Imre Risi Kondor. *Group Theoretical Methods in Machine Learning.* Columbia University, 2008.

[276] Miroslav Kramár, Rachel Levanger, Jeffrey Tithof, Balachandra Suri, Mu Xu, Mark Paul, Michael F Schatz, and Konstantin Mischaikow. Analysis of Kolmogorov flow and Rayleigh–Bénard convection using persistent homology. *Physica D: Nonlinear Phenomena*, 334:82–98, 2016.

[277] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE J.*, 37(2):233–243, 1991.

[278] Mark A. Kramer. Autoassociative neural networks. *Comput. Chem. Engng.*, 16(4):313–328, 1992.

[279] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[280] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[281] John Kruschke. *Doing Bayesian Data Analysis.* Academic Press, 2nd edition, 2015.

[282] Harold W. Kuhn. Variants of the Hungarian method for assignment problems. *Naval Research Logistics Quarterly*, 3(4):253–258, 1956.

[283] Max Kuhn. *caret: Classification and Regression Training*, 2020. R package version 6.0-85.

[284] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling.* Springer, 2013.

[285] S. Kuiper and J. Sklar. *Practicing Statistics: Guided Investigations for the Second Course.* Pearson, 2012.

[286] Sriram Kumar and Andreas Savakis. Robust domain adaptation on the l1-grassmannian manifold. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 103–110, 2016.

[287] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 614–623. Association for Computing Machinery, 1998.

[288] Michael Kutner, Christopher Nachtsheim, John Neter, and William Li. *Applied Linear Statistical Models.* McGraw-Hill/Irwin, 5th edition, 2004.

[289] Gitta Kutyniok, Ali Pezeshki, Robert Calderbank, and Taotao Liu. Robust dimension reduction, fusion frames, and grassmannian packings. *Applied and Computational Harmonic Analysis*, 26(1):64–76, 2009.

[290] Henry Kvinge and Mark Blumstein. Letting symmetry guide visualization: multidimensional scaling on groups. *arXiv preprint arXiv:1812.03362*, 2018.

[291] Henry Kvinge, Elin Farnell, Michael Kirby, and Chris Peterson. A gpu-oriented algorithm design for secant-based dimensionality reduction. In *2018 17th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 69–76. IEEE, 2018.

[292] Henry Kvinge, Elin Farnell, Michael Kirby, and Chris Peterson. Monitoring the shape of weather, soundscapes, and dynamical systems: A new statistic for dimension-driven data analysis on large datasets. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1045–1051. IEEE, 2018.

[293] Henry Kvinge, Elin Farnell, Michael Kirby, and Chris Peterson. Too many secants: A hierarchical approach to secant-based dimensionality reduction on large data sets. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2018.

[294] Hee-Dae Kwon. Optimal treatment strategies derived from a HIV model with drug-resistant mutants. *Applied Mathematics and Computation*, 188(2):1193–1204, 2007.

[295] Brigitte Lahme and Rick Miranda. Karhunen-loeve decomposition in the presence of symmetry. I. *IEEE Transactions on Image Processing*, 8(9):1183–1190, 1999.

[296] Greg Lamp et al. ggplot: A package for plotting in Python. `http://yhat.github.io/ggpy`. Accessed: 2020-01-05.

[297] Peter Lancaster and Miron Tismenetsky. *The Theory of Matrices: With Applications*. Academic Press, 2nd edition, 1985.

[298] Andrea Lancichinetti, Santo Fortunato, and Janos Kertesz. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009.

[299] Bruce E Landon, Jukka-Pekka Onnela, Nancy L Keating, Michael L Barnett, Sudeshna Paul, A James O'Malley, Thomas Keegan, and Nicholas A Christakis. Using administrative data to identify naturally occurring networks of physicians. *Medical Care*, 51(8):715, 2013.

[300] David Lane. Rice virtual lab in statistics. `http://onlinestatbook.com/rvls/index.html`.

[301] Serge Lang. *Graduate Texts in Mathematics: Algebra*. Springer, 2002.

[302] Amy N. Langville and Carl D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Pearson, 2004.

[303] Ben Lauwens. Simjulia: A combined continuous time/discrete event process oriented simulation framework. `https://simjuliajl.readthedocs.io`.

[304] Averill M. Law. *Simulation Modeling and Analysis*. McGraw-Hill, 5th edition, 2015.

[305] Neil D Lawrence. A unifying probabilistic perspective for spectral dimensionality reduction: Insights and new models. *Journal of Machine Learning Research*, 13(May):1609–1638, 2012.

[306] Jeffery J. Leader. *Numerical Analysis and Scientific Computation.* Princeton University Press, 20011.

[307] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[308] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al. Comparison of learning algorithms for handwritten digit recognition. In *International Conference on Artificial Neural Networks*, volume 60, pages 53–60. Perth, Australia, 1995.

[309] Friedrich Leisch. Neighborhood graphs, stripes and shadow plots for cluster visualization. *Statistics and Computing*, 20(4):457–469, 2010.

[310] Friedrich Leisch and Evgenia Dimitriadou. *mlbench: Machine Learning Benchmark Problems*, 2010. R package version 2.1-1.

[311] Jian Li, Qian Du, and Caixin Sun. An improved box-counting method for image fractal dimension estimation. *Pattern Recognition*, 42(11):2460–2469, 2009.

[312] Mengyi Liu, Ruiping Wang, Zhiwu Huang, Shiguang Shan, and Xilin Chen. Partial least squares regression on grassmannian manifold for emotion recognition. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 525–530. ACM, 2013.

[313] Tao Liu, Joseph W. Hogan, Michael J. Daniels, Mia Coetzer, Yizhen Xu, Gerald Bove, Allison K. DeLong, Rami Kantor, Lauren Ledingham, Millicent Orido, and Lameck Diero. Improved HIV-1 viral load monitoring capacity using pooled testing with marker-assisted deconvolution. *Journal of acquired immune deficiency syndromes*, 75(5):580, 2017.

[314] Tao Liu, Joseph W. Hogan, Lisa Wang, Shangxuan Zhang, and Rami Kantor. Optimal allocation of gold standard testing under constrained availability: Application to assessment of HIV treatment failure. *Journal of the American Statistical Association*, 108(504):1173–1188, 2013.

[315] R. Lock, P. Lock, K. Lock, E. Lock, and D. Lock. Statkey website. `http://www.lock5stat.com/StatKey/index.html`.

[316] R. Lock, P. Lock, K. Lock, E. Lock, and D. Lock. *Unlocking the Power of Statistics.* Wiley, 2012.

[317] Robin Lock and Patti Lock. Introducing statistical inference to biology students through bootstrapping and randomization. *PRIMUS*, 18:39–48, 01 2008.

[318] M. Loève. *Probability Theory.* von Nostrand, Princeton, N.J., 1955.

[319] E. Lorenz. Empirical orthogonal eigenfunctions and statistical weather prediction. Science Report No. 1, Statistical Forecasting Project 1, M.I.T., Cambridge, MA, 1956.

[320] Edward Norton Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences. 20 (2)*, 1963.

[321] Joseph Lotem, Dvir Netanely, Eytan Domany, and Leo Sachs. Human cancers overexpress genes that are specific to a variety of normal human tissues. *Proceedings of the National Academy of Sciences*, 102(51):18556–18561, 2005.

[322] Robin Lougee-Heimer. The Common Optimization INterface for Operations Research. *IBM Journal of Research and Development*, 47(1):57–66, January 2003.

[323] John Loughrey and Pádraig Cunningham. Overfitting in wrapper-based feature subset selection: The harder you try the worse it gets. In Max Bramer, Frans Coenen, and Tony Allen, editors, *Research and Development in Intelligent Systems XXI*, pages 33–43, London, 2005. Springer London.

[324] Miles Lubin and Iain Dunning. Computing in operations research using Julia. *INFORMS Journal on Computing*, 27(2), 2015.

[325] R. Duncan Luce. *Individual Choice Behavior: A Theoretical Analysis.* Courier Corporation, 1959.

[326] Ontje Lünsdorf and Stefan Scherfke. Simpy: a process-based discrete-event simulation framework based on standard Python. `https://simpy.readthedocs.io`.

[327] David Lusseau, Karsten Schneider, Oliver J. Boisseau, Patti Haase, Elisabeth Slooten, and Steve M Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

[328] Bei Ma and Hailin Zhang. Recognition of faces using texture-based principal component analysis and grassmannian distances analysis. In *International Conference on Graphic and Image Processing (ICGIP 2011)*, volume 8285, page 82856C. International Society for Optics and Photonics, 2011.

[329] P. Macnaughton-Smith, W.T. Williams, M.B. Dale, and L.G. Mockett. Dissimilarity analysis: a new technique of hierarchical sub-division. *Nature*, 202(4936):1034, 1964.

[330] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[331] Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2019. R package version 2.1.0.

[332] Benoit B. Mandelbrot. Stochastic models for the earth's relief, the shape and the fractal dimension of the coastlines, and the number-area rule for islands. *Proceedings of the National Academy of Sciences*, 72(10):3825–3828, 1975.

[333] Olvi L. Mangasarian, W. Nick Street, and William H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, 1995.

[334] K.V. Mardia, J.T. Kent, and J.M. Bibby. *Multivariate Analysis.* Academic Press, 1979.

[335] Tim Marrinan, J. Ross Beveridge, Bruce Draper, Michael Kirby, and Chris Peterson. Flag manifolds for the characterization of geometric structure in large data sets. In *Numerical Mathematics and Advanced Applications-ENUMATH 2013*, pages 457–465. Springer, 2015.

[336] S. Martin and J. P. Watson. Non-manifold surface reconstruction from high-dimensional point cloud data. *Computational Geometry*, 44:427–441, 2011.

[337] Shawn Martin, Aidan Thompson, Evangelos A. Coutsias, and Jean-Paul Watson. Topology of cyclo-octane energy landscape. *The journal of chemical physics*, 132(23):234115, 2010.

[338] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[339] Leland McInnes, John Healy, and James Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[340] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[341] N. Meinshausen, A. Hauser, J.M. Mooij, J. Peters, P. Versteeg, and P. Bühlmann. Methods for causal inference from gene perturbation experiments and validation. *Proceedings of the National Academy of Sciences*, 113:7361–7368, 2016.

[342] Facundo Mémoli. Gromov–Hausdorff distances in Euclidean spaces. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008.

[343] B.D. Meyer. Natural and quasi-experiments in economics. *Journal of business & economic statistics*, 13:151–161, 1995.

[344] Zlatko Drmač Michael W. Berry and Elizabeth R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[345] Microsoft. Microsoft Azure Cloud Products. https://azure.microsoft.com. Accessed: 2020-01-05.

[346] Microsoft. Visual Studio Code. https://code.visualstudio.com. Accessed: 2020-01-31.

[347] Greg Miller. A scientist's nightmare: Software problem leads to five retractions. *Science*, 314(5807):1856–1857, 2006.

[348] Nikola Milosavljević, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the twenty-seventh annual symposium on Computational geometry*, pages 216–225. ACM, 2011.

[349] Marvin Minsky and Seymour Papert. Perceptron: an introduction to computational geometry. *The MIT Press, Cambridge, expanded edition*, 19(88):2, 1969.

[350] Konstantin Mischaikow, Hiroshi Kokubu, Marian Mrozek, Paweł Pilarczyk, Tomas Gedeon, Jean-Philippe Lessard, and Marcio Gameiro. Chomp: Computational homology project. *Software available at http://chomp.rutgers.edu*, 2014.

[351] M. Mitchell. Strategically using general purpose statistics packages: A look at stata, sas and spss. *UCLA Technical Report Series, Report Number 1, Version Number 1*, 2007.

[352] David S. Moore. Should mathematicians teach statistics? *The College Mathematics Journal*, 19(1):3–7, 1988.

[353] D.R. Morse, J.H. Lawton, M.M. Dodson, and M.H. Williamson. Fractal dimension of vegetation and the distribution of arthropod body lengths. *Nature*, 314(6013):731, 1985.

[354] Davoud Moulavi, Pablo A. Jaskowiak, Ricardo J.G.B. Campello, Arthur Zimek, and Jörg Sander. Density-based clustering validation. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 839–847. SIAM, 2014.

[355] Fionn Murtagh and Pierre Legendre. Ward's hierarchical clustering method: Clustering criterion and agglomerative algorithm. *arXiv preprint arXiv:1111.6285*, 2011.

[356] Rahul Nair and Elise Miller-Hooks. Fleet management for vehicle sharing operations. *Transportation Science*, 45(4):524–540, 2011.

[357] John Nash. C^1 isometric imbeddings. *Annals of Mathematics*, pages 383–396, 1954.

[358] John Nash. The imbedding problem for Riemannian manifolds. *Annals of Mathematics*, pages 20–63, 1956.

[359] Yurii Nesterov. *Lectures on Convex Optimization*, volume 137. Springer, 2018.

[360] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point Polynomial Algorithms in Convex Programming*, volume 13. SIAM, 1994.

[361] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[362] Raymond T Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge & Data Engineering*, 14(5):1003–1016, 2002.

[363] Jorge Nocedal and Stephen Wright. *Numerical Optimization.* Springer Science & Business Media, 2006.

[364] Deborah Nolan and Jamis Perrett. Teaching and learning data visualization: Ideas and assignments. *The American Statistician*, 70:260–269, 2016.

[365] Donald J Norris. Introduction to artificial intelligence. In *Beginning Artificial Intelligence with the Raspberry Pi*, pages 1–15. Springer, 2017.

[366] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Stphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. The Scala language specification, 2004.

[367] U.S. Department of Education. College scorecard. https://collegescorecard.ed.gov/data/, 2019. Accessed: 2019-12-14.

[368] E. Oja. Data compression, feature extraction, and autoassociation in feedforward neural networks. In T. Kohonen, K. Mäkisara., O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 737–745, NY, 1991. Elsevier Science.

[369] Paul G Okubo and Keiiti Aki. Fractal geometry in the San Andreas fault system. *Journal of Geophysical Research: Solid Earth*, 92(B1):345–355, 1987.

[370] Travis E. Oliphant. Guide to NumPy (2nd ed.). *CreateSpace Independent Publishing Platform*, North Charleston, SC, USA, 2015.

[371] Jack J. Olney, Paula Braitstein, Jeffrey W. Eaton, Edwin Sang, Monicah Nyambura, Sylvester Kimaiyo, Ellen McRobie, Joseph W. Hogan, and Timothy B. Hallett. Evaluating strategies to improve HIV care outcomes in Kenya: A modelling study. *The Lancet HIV*, 3(12):e592–e600, 2016.

[372] ASA/MAA Joint Committee on Undergraduate Statistics. Second courses in applied statistics. http://www.amstat.org/education/pdfs/second-course-syllabus.pdf, February 2016.

[373] OpenAI. Gym: A toolkit for developing and comparing reinforcement learning algorithms. https://gym.openai.com/.

[374] World Health Organization. Antiretroviral therapy for HIV infection in adults and adolescents: Recommendations for a public health approach-2010 revision. https://www.who.int/hiv/pub/arv/adult2010/en/, 2010.

[375] Nina Otter, Mason A. Porter, Ulrike Tillmann, Peter Grindrod, and Heather A. Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science*, 6(1):17, 2017.

[376] Michael Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM, 2001.

[377] Cathy O'Neil. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. Crown Publishing Group, USA, 2016.

[378] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw. Geometry from a time series. *Physical Review Letters*, 45(9):712–716, 1980.

[379] Daniel P. Palomar and Yonina C. Eldar. *Convex optimization in signal processing and communications*. Cambridge university press, 2010.

[380] E. Paradis and K. Schliep. ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, 35:526–528, 2018.

[381] M Parimala, Daphne Lopez, and N.C. Senthilkumar. A survey on density based clustering algorithms for mining large spatial databases. *International Journal of Advanced Science and Technology*, 31(1):59–66, 2011.

[382] Vishal M. Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Visual domain adaptation: A survey of recent advances. *IEEE signal processing magazine*, 32(3):53–69, 2015.

[383] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Phil. Mag. S.*, 2(11):559–572, 1901.

[384] Roxy Peck, Chris Olsen, and Jay Devore. *Introduction to Statistics & Data Analysis*. Brooks Cole, 5th edition, 2015.

[385] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[386] Jose A. Perea and Gunnar Carlsson. A Klein-bottle-based dictionary for texture representation. *International journal of computer vision*, 107(1):75–97, 2014.

[387] Dominique Perrault-Joncas and Marina Meila. Metric learning and manifolds: Preserving the intrinsic geometry. *Preprint Department of Statistics, University of Washington*, 2012.

[388] Robert Piziak and P.L. Odell. *Matrix Theory: From Generalized Inverses to Jordan Form*. Chapman and Hall/CRC, 2007.

[389] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, 10(2):191–218, 2006.

[390] Warren B. Powell. Clearing the jungle of stochastic optimization. *INFORMS Tutorials in Operations Research*, pages 109–137, 2014. http://dx.doi.org/10.1287/educ.2014.0128.

[391] Warren B. Powell. Stochastic optimization challenges in energy, 2016. 4th International Conference on Computational Sustainability https://www.youtube.com/watch?v=dOIoTFX8ejM.

[392] Warren B. Powell. Tutorial on stochastic optimization in energy – part II: An energy storage illustration. *IEEE Transactions on Power Systems*, 31(2):1468 – 1475, 2016. https://doi.org/10.1109/TPWRS.2015.2424980.

[393] Warren B. Powell. A unified framework for stochastic optimization. *European Journal of Operational Research*, pages 795–821, 2019. https://doi.org/10.1016/j.ejor.2018.07.014.

[394] Randall Pruim, Daniel T Kaplan, and Nicholas J Horton. The mosaic package: Helping students to 'think with data' using R. *The R Journal*, 9(1):77–102, 2017.

[395] Weiliang Qiu and Harry Joe. *clusterGeneration: Random Cluster Generation (with Specified Degree of Separation)*, 2015. R package version 1.3.4.

[396] Fernando A. Quintana. A predictive view of bayesian clustering. *Journal of Statistical Planning and Inference*, 136(8):2407–2429, 2006.

[397] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019.

[398] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Code for the paper "language models are unsupervised multitask learners". https://github.com/openai/gpt-2, 2019.

[399] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *URL https://openai.com/blog/better-language-models*, 2019.

[400] Matthias Raess. Exploring elastic net and multivariate regression. Available on arXiv at https://arxiv.org/abs/1607.06763, 2016.

[401] Fred Ramsey and Dan Schafer. *The Statistical Sleuth: A course in methods of data analysis*. Brooks/Cole Cengage Learning, 2013.

[402] Stephen Raudenbush and Anthony Bryk. *Hierarchical Linear Models: Applications and Data Analysis Methods*. Advanced Quantitative Techniques in the Social Sciences. SAGE Publications, 2nd edition, 2001.

[403] Tal Raviv, Michal Tzur, and Iris A. Forma. Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics*, 2(3):187–229, 2013.

[404] Jeffrey Reaser, Eric Wilbanks, Karissa Wojcik, and Walt Wolfram. *Language variety in the new South: Contemporary perspectives on change and variation*. UNC Press Books, 2018.

[405] Leonard Richardson. Beautiful soup documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc/, April 2007.

[406] Armin Rigo, Maciej Fijalkowski, et al. cffi 1.13.2: Foreign Function Interface for Python calling C code. https://cffi.readthedocs.io/en/latest/.

[407] Thomas Roelleke. *Information Retrieval Models: Foundations and Relationships.* Morgan & Claypool, 2013.

[408] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[409] S. Ross. *Introduction to Probability and Statistics for Engineers and Scientists*, 4th edition. Academic Press, 2009.

[410] Sheldon Ross. *Simulation.* Academic Press, 5th edition, 2012.

[411] A. Rossman and B. Chance. Applet collection. http://www.rossmanchance.com/applets/.

[412] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[413] RStudio Team. *RStudio: Integrated Development Environment for R.* RStudio, Inc., Boston, MA, 2015.

[414] Ulrich Rüde, Karen Willcox, Lois Curfman McInnes, and Hans De Sterck. Research and education in computational science and engineering. *SIAM Review*, 60(3):707–754, 2018.

[415] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[416] Paat Rusmevichientong, Zuo-Jun Max Shen, and David B. Shmoys. Dynamic assortment optimization with a multinomial logit choice model and capacity constraint. *Operations Research*, 58(6):1666–1680, 2010.

[417] Paat Rusmevichientong, David Shmoys, Chaoxu Tong, and Huseyin Topaloglu. Assortment optimization under the multinomial logit model with random choice parameters. *Production and Operations Management*, 23(11):2023–2039, 2014.

[418] Guillaume Sagnol and Maximilian Stahlberg. Picos: A Python interface to conic optimization solvers. https://picos-api.gitlab.io/picos/.

[419] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[420] Deepayan Sarkar. *Lattice: Multivariate Data Visualization with R.* Springer, New York, 2008. ISBN 978-0-387-75968-5.

[421] Tim Sauer, James A. Yorke, and Martin Casdagli. Embedology. *Journal of statistical Physics*, 65(3-4):579–616, 1991.

[422] Nathaniel Saul and Chris Tralie. Scikit-TDA: Topological Data Analysis for Python. https://doi.org/10.5281/zenodo.2533369, 2019.

[423] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[424] Jaap C Schouten, Floris Takens, and Cor M van den Bleek. Estimation of the dimension of a noisy attractor. *Physical Review E*, 50(3):1851, 1994.

[425] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[426] M.J. Schuemie, P.B. Ryan, W. DuMouchel, M.A. Suchard, and D. Madigan. Interpreting observational studies: why empirical calibration is needed to correct p-values. *Stat. Med.*, 33:209–218, 2014.

[427] Rachel Schutt and Cathy O'Neil. *Doing Data Science: Straight Talk from the Frontline*. O'Reilly Media, Inc., 2013.

[428] Jean-Pierre Serre. *Linear Representations of Finite Groups*, volume 42. Springer, 1977.

[429] Bahar Shahnavaz, Lucie Zinger, Sébastien Lavergne, Philippe Choler, and Roberto A Geremia. Phylogenetic clustering reveals selective events driving the turnover of bacterial community in alpine tundra soils. *Arctic, Antarctic, and Alpine Research*, 44(2):232–238, 2012.

[430] David A Shaw and Rama Chellappa. Regression on manifolds using data-dependent regularization with applications in computer vision. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 6(6):519–528, 2013.

[431] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[432] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Departmental Papers (CIS)*, page 107, 2000.

[433] Naum Zuselevich Shor. *Minimization Methods for Non-differentiable Functions*, volume 3. Springer Science & Business Media, 2012.

[434] Robert Shumway and David Stoffer. *Time Series Analysis and its Applications: With R Examples*. Springer, 4th edition, 2017.

[435] David Silver and Demis Hassabis. AlphaGo: Mastering the Ancient Game of Go. https://ai.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html, 2016.

[436] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017.

[437] L. Sirovich and M. Kirby. A low-dimensional procedure for the characterization of human faces. *J. of the Optical Society of America A*, 4:529–524, 1987.

[438] Colin Sparrow. *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors*. Springer, 1982.

[439] David I. Spivak. Metric realization of fuzzy simplicial sets. Self published notes, available online at https://www.semanticscholar.org/paper/METRIC-REALIZATION-OF-FUZZY-SIMPLICIAL-SETS-Spivak/a73fb9d562a3850611d2615ac22c3a8687fa745e, 2012.

[440] Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright. *Optimization for machine learning*. MIT Press, 2012.

[441] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[442] Cédric St-Jean. ScikitLearn.jl: Julia implementation of the scikit-learn API. https://github.com/cstjean/ScikitLearn.jl, November 2019.

[443] Henry Stark. *Image Recovery: Theory and Application*. Elsevier, 2013.

[444] G.W. Stewart. *Matrix Algorithms, Vol. I: Basic Decompositions*. SIAM, 1998.

[445] G.W. Stewart. *Matrix Algorithms, Vol. II: Eigensystems*. SIAM, 2001.

[446] Stochastic Programming Society. Stochastic programming resources. Project website: https://www.stoprog.org/resources. Accessed 25 July 2019.

[447] V. Stodden. Reproducing statistical results. *Annu. Rev. Stat. Appl*, 2:1–19, 2015.

[448] Thomas Strohmer and Robert W. Heath Jr. Grassmannian frames with applications to coding and communication. *Applied and computational harmonic analysis*, 14(3):257–275, 2003.

[449] Sublime HQ Pty Ltd. Sublime Text. http://www.sublimetext.com. Accessed: 2020-01-05.

[450] M. Sumida and H. Topaloglu. An approximation algorithm for capacity allocation over a single flight leg with fare-locking. *INFORMS Journal on Computing*, pages 83–99, 2019.

[451] James J. Swain. Simulation software survey. *OR/MS Today*, 44(5):38–49, 2017.

[452] Panagiotis Symeonidis and Andreas Zioupos. *Matrix and Tensor Factorization Techniques for Recommender Systems*. Springer International Publishing, 2016.

[453] Tableau Software. Tableau. https://www.tableau.com. Accessed: 2020-01-05.

[454] Sima Taheri, Pavan Turaga, and Rama Chellappa. Towards view-invariant expression analysis using analytic shape manifolds. In *Face and Gesture 2011*, pages 306–313. IEEE, 2011.

[455] F. Takens. Detecting Strange Attractors in Turbulence. *Dynamical Systems and Turbulence; Proceeding of a symposium held at University of Warwick 1979-1980*, pages 366–381, 1980.

[456] Kalyan Talluri and Garrett Van Ryzin. Revenue management under a general discrete choice model of consumer behavior. *Management Science*, 50(1):15–33, 2004.

[457] Ryuta Tamura, Ken Kobayashi, Yuichi Takano, Ryuhei Miyashiro, Kazuhide Nakata, and Tomomi Matsui. Mixed integer quadratic optimization formulations for eliminating multicollinearity based on variance inflation factor. *Journal of Global Optimization*, 73(2):431–446, February 2019.

[458] Andrew Tausz, Mikael Vejdemo-Johansson, and Henry Adams. Javaplex: A research software package for persistent (co)homology. In *International Congress on Mathematical Software*, pages 129–136, 2014. Software available at http://appliedtopology.github.io/javaplex/.

[459] The CVXPY authors. CVXPY 1.0: Convex optimization, for everyone. https://www.cvxpy.org/.

[460] The Eclipse Foundation. Eclipse IDE: The Leading Open Platform for Professional Developers. http://www.eclipse.org/. Accessed: 2020-01-31.

[461] The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015.

[462] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[463] H.J. Thode. *Testing for Normality.* Marcel Dekker, New York, 2002.

[464] Peter W. Foltz Thomas K. Landauer, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284, 1998.

[465] N. Tintle, B. Chance, G. Cobb, S. Roy, T. Swanson, and J. Vander-Stoep. Combating anti-statistical thinking using simulation-based methods throughout the undergraduate curriculum. *Faculty Work: Comprehensive List*, 2015.

[466] N. Tintle, J. VanderStoep, V l Holmes, B. Quisenberry, and T. Swanson. Development and assessment of a preliminary randomization-based introductory statistics curriculum. *Journal of Statistics Education*, 19(1), 2011.

[467] Chad M Topaz, Lori Ziegelmeier, and Tom Halverson. Topological data analysis of biological aggregation models. *PloS one*, 10(5):e0126383, 2015.

[468] Linus Torvalds et al. Git version control system 2.25.0. https://git-scm.com/.

[469] Paolo Toth and Daniele Vigo. *Vehicle routing: Problems, methods, and applications.* SIAM, 2014.

[470] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra.* SIAM, 1997.

[471] J.W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, Reading, MA, 1977.

[472] Pavan Turaga and Rama Chellappa. Locally time-invariant models of human activities using trajectories on the grassmannian. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2435–2441. IEEE, 2009.

[473] Alexander J. Tybl. An overview of spatial econometrics. https://arxiv.org/abs/1605.03486, 2016.

[474] Iñaki Ucar, José Alberto Hernández, Pablo Serrano, and Arturo Azcorra. Design and analysis of 5G scenarios with simmer: An R package for fast DES prototyping. *IEEE Communications Magazine*, 56(11):145–151, November 2018.

[475] Iñaki Ucar and Bart Smeets. simmer: Discrete-event simulation package for R. https://r-simmer.org/.

[476] Iñaki Ucar, Bart Smeets, and Arturo Azcorra. simmer: Discrete-event simulation for R. *Journal of Statistical Software*, 90(2):1–30, 2019.

[477] United States Energy Information Administration. EIA-930 data users guide and known issues, 2019. https://www.eia.gov/realtime_grid/docs/userguide-knownissues.pdf.

[478] United States Energy Information Administration. Electricity, 2019. https://www.eia.gov/electricity/data.php.

[479] United States Energy Information Administration. Homepage, 2019. https://www.eia.gov/.

[480] United States Energy Information Administration. Today in energy: Prices, 2019. https://www.eia.gov/todayinenergy/prices.php.

[481] United States Energy Information Administration. U.S. electric system operating data, 2019. https://www.eia.gov/beta/realtime_grid/.

[482] U.S. General Services Administration. data.gov. https://www.data.gov. Accessed: 2020-01-05.

[483] J. H. van Hateren and A. van der Schaaf. Independent component filters of natural images compared with simple cells in primary visual cortex. *Proc. R. Soc. Lond. B*, 265:359–366, 1998.

[484] Charles F Van Loan. Generalizing the singular value decomposition. *SIAM Journal on Numerical Analysis*, 13(1):76–83, 1976.

[485] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[486] R. D. De Veaux, M. Agarwal, M. Averett, B. Baumer, A. Bray, T. Bressoud, L. Bryant, L. Cheng, A. Francis, R. Gould, A. Kim, M. Kretchmar, Q. Lu, A. Moskol, D. Nolan, R. Pelayo, S. Raleigh, R. Sethi, M. Sondjaja, N. Tiruviluamala, P. Uhlig, T. Washington, C. Wesley, D. White, and P. Ye. Curriculum guidelines for undergraduate programs in data science. *Annual Review of Statistics and Its Application*, 4, 2017.

[487] Tyler Vigen. Spurious correlation website. http://www.tylervigen.com/spurious-correlations.

[488] Gregory K Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.

[489] Tiesheng Wang and Pengfei Shi. Kernel grassmannian distances and discriminant analysis for face recognition from image sets. *Pattern Recognition Letters*, 30(13):1161–1165, 2009.

[490] Xinchao Wang, Wei Bian, and Dacheng Tao. Grassmannian regularized structured multi-view embedding for image classification. *IEEE Transactions on Image Processing*, 22(7):2646–2660, 2013.

[491] Joe H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[492] Michael Waskom, Olga Botvinnik, Drew O'Kane, Paul Hobson, Joel Ostblom, Saulius Lukauskas, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Alistair Miles, Yoav Ram, Thomas Brunner, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, Brian, and Adel Qalieh. mwaskom/seaborn: v0.9.0 (july 2018), July 2018.

[493] Philip D. Wasserman and Tom Schwartz. Neural networks. ii. what are they and why is everybody so interested in them now? *IEEE Expert*, 3(1):10–15, 1988.

[494] R.L. Wasserstein and N.A. Lazar. The ASA's statement on p-values: Context, process, and purpose. *The American Statistician*, 70:129–133, 2016.

[495] S. Watanabe. Karhunen–Loève expansion and factor analysis. In *Trans. 4th. Prague Conf. on Inf. Theory, Statist. Decision Functions, and Random Proc.*, pages 635–660, Prague, 1965.

[496] Jean-Paul Watson, David L. Woodruff, and William E. Hart. Pysp: modeling and solving stochastic programs in Python. *Mathematical Programming Computation*, 4(2):109–149, 2012.

[497] P.J. Werbos. New tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

[498] David White. A project based approach to statistics and data science. *PRIMUS*, 29(9):997–1038, 2019.

[499] H. Whitney. Differentiable manifolds. *Annals of Math.*, 37:645–680, 1936.

[500] Hadley Wickham. testthat: Get started with testing. *The R Journal*, 3:5–10, 2011.

[501] Hadley Wickham. Tidy data. *The Journal of Statistical Software*, 59, 2014.

[502] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York, 2016.

[503] Hadley Wickham. *rvest: Easily Harvest (Scrape) Web Pages*, 2019. R package version 0.3.4.

[504] Hadley Wickham and Jennifer Bryan. *readxl: Read Excel Files*, 2019. R package version 1.3.1.

[505] Hadley Wickham, Romain François, Lionel Henry, and Kirill Müller. *dplyr: A Grammar of Data Manipulation*, 2019. R package version 0.8.3.

[506] Wikipedia contributors. Women in the United States House of Representatives—Wikipedia, The Free Encyclopedia, 2020. [Online; accessed 05-January-2020].

[507] C. J. Wild, M. Pfannkuch, and M. Regan. Towards more accessible conceptions of statistical inference. *Journal of the Royal Statistical Society*, 174(2):247–295, 2011.

[508] C. J. Wild, M. Pfannkuch, M. Regan, and R. Parsonage. Accessible conceptions of statistical inference: Pulling ourselves up by the bootstraps. *International Statistical Review*, 85(1):84–107, 2017.

[509] L. Wilkinson. *The Grammar of Graphics, 2nd edition.* Springer, 2005.

[510] Niklaus Wirth. *Algorithms+Data Structures=Programs.* Prentice Hall, 1976.

[511] Wisconsin Institutes for Discovery. NEOS Server: State-of-the-Art Solvers for Numerical Optimization. https://neos-server.org/neos/.

[512] John H Wolfe. Pattern clustering by multivariate mixture analysis. *Multivariate Behavioral Research*, 5(3):329–350, 1970.

[513] M. Wood. The role of simulation approaches in statistics. *Journal of Statistics Education*, 13(3), 2005.

[514] J. Wooldridge. *Introductory Econometrics: a Modern Approach.* South-Western Cengage Learning, Mason, Ohio, 2012.

[515] Robert E. D. Woolsey. Where were we, where are we, where are we going, and who cares? *Interfaces*, 23(5):40–46, 1993.

[516] Robert E. D. Woolsey and Richard L. Hewitt. *Real World Operations Research: The Woolsey Papers.* Lionheart Publishing, Inc., Marietta, Georgia, 2003.

[517] Stephen J. Wright. *Primal-dual interior-point methods*, volume 54. SIAM, 1997.

[518] Dan Zhang and Daniel Adelman. An approximate dynamic programming approach to network revenue management with customer choice. *Transportation Science*, 43(3):381–394, 2009.

[519] Ying Zhao and George Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 515–524. ACM, 2002.

[520] Ji Zhu and Trevor Hastie. Kernel logistic regression and the import vector machine. *Journal of Computational and Graphical Statistics*, 14(1):185–205, 2005.

[521] Y. Zhu, L. M. Hernandez, P. Mueller, Y. Dong, and M. R. Forman. Data acquisition and preprocessing in studies on humans: What is not taught in statistics classes? *The American Statistician*, 67(4):235–241, 2013.

[522] A. Zomorodian. *Advances in Applied and Computational Topology.* American Mathematical Society, 2012.

[523] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete and Computational Geometry*, 33(2):249–274, 2005.

[524] Afra J Zomorodian. *Topology for computing*, volume 16. Cambridge university press, 2005.

[525] J. Álvarez Vizoso, M. Kirby, and C. Peterson. Manifold curvature from covariance analysis. In *2018 IEEE Statistical Signal Processing Workshop (SSP)*, pages 353–357, June 2018.

[526] Åke Björck. *Numerical Methods for Least Squares Problems.* SIAM, 1996.

Index