DATABASE MINING

WHAT IS DATABASE MINING?

Did you ever consider buying an Edsel? Could the Three Mile Island nuclear accident have been avoided? How can major national grocery chains adjust their product admixture and availability according to the perceived buying habits of shoppers possessing different demographic indicators? What makes a mountain a mountain and not a mountain range?

Underlying these questions lies the crux to our understanding of what data mining *is* and what data mining *is not.* Let us consider each question above briefly. Could the Ford Motor Company have predicted the failure of the Edsel, thereby reducing its risk and, as it turned out, Ford's losses? Was their failure due to abysmal marketing analysis or due to fickle buyers? Could any analysis of customer data have indicated customer preferences that would have prevented this disaster for Ford? Was the problem exacerbated due to negative buyer perceptions, especially once unfavorable publicity enveloped the Edsel?

Shortly after the Three Mile Island nuclear plant was shut down, it became apparent that, if the information shown on multiple display units in multiple settings had been designed to focus the attention of operators (presumably using better user interfaces), then the problem might have been avoided. Were the data analyzed incorrectly or incompletely or merely communicated in a haphazard fashion to the operators on duty? Were the displays not designed to be properly informative to provide operators with the opportunity to make informed decisions and take appropriate actions?

When a single male walks into a grocery store late on a Friday or Saturday evening to purchase disposable diapers, is it very advantageous to the store owners to place beer and chips on display adjacent to the diapers, for that time period, only to move them in favor of another product set soon thereafter? Can a pay-per-view cable service marker associate products in mixtures that account for not only the buying habits of their customers but also to localize their marketing strategies, say, according to buying habits cross-referenced by postal areas organized in a hierarchy from "rich postal code" to "poor postal code"? [By "rich (poor) postal code" is meant a geographical locale in which the population enjoy a high (low) standard of living.]

How do we learn what constitutes a mountain? Is it a "saddle-point" that separates a mountain from a mountain range or does that merely distinguish one mountain from another or worse, just illustrate an aberration in a single mountain? Where does a mountain end? How high above sea level and what steepness classifies mountainess? How do we distinguish a mountain from a hill, a peak, a butte, a mesa, a dome, a bluff, a volcano, a sierra, . . .? Perhaps we learn such things from being told, or by rote, or by analogy, or by some other means. More likely we form our understanding of, in this case, symbolic concepts by constant refinement and reinforcement (both negative and positive) of our model of the concept, hence the terms *concept learning* or *concept formation* are applied.

At first blush, it would appear that the first two questions delve into a realm of design and analysis which is not exclusively or primarily the purview of knowledge discovery in databases or data mining, as it has also been known. Although the terms *knowledge discovery in databases* and *data mining* have tended to be used interchangeably by researchers in the past, a recent article by Fayyad et al. (1) explains their differences and delineates the KDD process. Fayyad et al.'s excellent article defines the KDD process, basic data mining algorithms and applications at a basic level, unifying concepts wherever possible. Certainly there were factors surrounding the failure of the Edsel that defied proper marketing characterization at that time. Perhaps the human factors engineering that went into the design of information display units at Three Mile Island might have been better informed with some careful formative analysis of operator evaluation and skill testing in an actual job setting situation.

Answers to the third question and others like it audaciously converge at the heart of what is known as *data mining.* Properly so the answers delineate one of many types of classificatory, associative, evolutionary, characterizations or analyses of some given data.

Broader issues which are concentrated in machine learning (ML) research are well served when we attempt to answer questions of the fourth type, in this case symbolic concept learning. ML has evolved rapidly over the past two decades and most recently ML researchers have embraced a variety of ML techniques in their efforts to improve the quality of learning systems. Two primary goals of machine learning are to understand and model the learning behavior of humans and, more pragmatically, to provide increasing levels of automation in the knowledge acquisition process. Bonded by common goals, ML research has emphasized different approaches. Rule induction (RI), neural networks, genetic algorithms, analytic learning, Bayesian learning, reinforcement learning, and case-based reasoning (CBR) have emerged as ML paradigms. Rooted in neurobiology (neural nets), evolution theories (genetic algorithms), formal logic (analytic methods), heuristic search (rule induction) and studies of human memory (case-based reasoning) have provided researchers with analogical models to study. Langley (2) reviews these major paradigms and describes some applications of rule induction, the most widely studied methodology.

Returning to questions of the third type, we can observe that large databases are commonplace and typically the data themselves and data interrelations are exceedingly complex. It is not sufficient to report the results of database mining for decision-making. Relevant results must first be discovered and these results must then be presented in an appropriate way. For example, such a discovery-driven database mining system applied to the cable television's customer database may discover many different groups of cable subscribers with common characteristics, for example, college students with little money who share a house relying on a single cable outlet for multiple televisions, married couples with children subscribing to extra arts and entertainment channels, and so on. By recognizing the marketing manager's goal, the discovery-driven system not only identifies the most appropriate groupings, but furthermore establishes which of the company's subscribers in the group will be good candidates for each type of promotional campaign that can be executed by the cable company.

Setting the Stage

We generally make tradeoffs when we design computer systems. Historically we have devised algorithms and written programs in which the major tradeoff has been between computing speed (time) versus computer memory (storage). (With present-day fast processors and large capacity memories and disks, we can attempt to solve problems given up a generation ago as unreasonable. Parallel processing has added another dimension of capability to our repertoire of problem solving tools.) A subtler form of the traditional space/time tradeoff is that of search versus inference: when does the cost of retrieving information exceed the cost of recreating that information? It is impractical to predict all possible valid inferences that can be made from a database relational structure and the values of the attributes, and many of these inferences would be meaningless. However, determining the search/inference tradeoff is useful. This tradeoff underlies the knowledge discovery in databases (KDD) or data mining process.

Data mining and knowledge discovery are largely misused terms. Since many software analytical tool vendors pervade today's business environment, "data mining" and KDD have been used somewhat indiscriminately, resulting in a variety of definitions which includes all tools employed to help users analyze and understand their data. We use more focused definitions. Knowledge discovery is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data (3). Data mining is a step in the KDD process consisting of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data (3). Essentially, database mining is a decision support process where we search for patterns of information in a database. This process may be performed by skilled data analysts, but in this case it is very difficult, or the process may be performed by an intelligent data mining program, which automatically searches the database and discovers patterns (finds information) on its own. Subsequently, this information is presented in a suitable form, with graphs, reports, text, hypertext, and so forth.

Information extracted from a database can be used for prediction or classification, to identify relations between database records, or to provide a database summary. A number of operations comprise database mining, each of which is supported by a variety of techniques and technologies such as rule induction, conceptual clustering, neural networks, and so forth. In many domains (marketing data analysis, financial data analysis, fraud detection, etc.) information extraction requires the cooperative use of several data mining operations, techniques, and/or technologies.

We discuss database mining in the context of relational database management systems. However, we do so with the knowledge that database mining techniques are and have been applied to other data representations and data stores, including object-oriented, temporal, spatial data, text-based, image, distributed, parallel, and multimedia domains. In principle, these techniques can be generalized to other kinds of databases as well, such as object-oriented, heterogeneous, and multimedia databases.

Database mining can be distinguished from other analytical tools in their exploration of data interrelationships. Many available analytical tools rely on the user to hypothesize specific data interrelationships and then they help to confirm or deny those hypotheses. These tools are of limited effectiveness, due to a number of factors, including the posing of appropriate questions and managing the complexity of the attribute space in a reasonable time. In contract, most available analytical tools have been optimized to address some specific issue(s). Query analysis and report generation tools handle usability issues, permitting users to develop SQL queries through graphical user interfaces (GUI). Statistical and rough sets analysis package the relationships to be investigated from among a few variables. Multidimensional analysis and relational on-line analytic processing (OLAP) tools precompute aggregation/generalization/specialization hierarchies along various dimensions in order to respond quickly to queries. Visualization tools permit multidimensional relationships to be illustrated by combining spatial and nonspatial attributes (location, size, color, etc.).

In contrast, database mining employs (inductive) discovery-based approaches to unearth significant data relationships. Database mining algorithms examine numerous multidimensional data relationships concurrently, identifying notable data relationships. To automatically determine which data relationships are interesting is the focus of some exciting current research. Furthermore, database mining has become an important contemporary research investigation for the 1990s, by database *and* machine learning researchers.

Because of the growth in the size and number of existing databases, the knowledge discovery process exceeds human abilities to analyze this data. The expanded reliance on databases as a corporate resource is also creating a need and an opportunity to develop computer methods for extracting knowledge from databases. We characterize the major database mining functions and techniques inspired by Morton's (4) summary.

Functions and Techniques

Database mining *applications* can be classified into sets of problems that share similar characteristics across different application domains. Different agencies and different applications may utilize different parameterizations of the application. Nonetheless, the same approaches and models used to develop a bank's fraud-detection capability might also be used to develop medical insurance fraud-detection applications if we could specify which domain-specific attributes in the data repository are used in the analysis and how they are used.

Different database mining *functions* are used to extract the relevant relationships from the data, for example, characteristic, discrimination, association, sequence-based analysis, and data evolution regularities, clustering, classification, and estimation relationships. These different approaches are also the names generally given to the types of *rules* discovered by the database mining system (5).

If we have a large transaction database, where each transaction consists of a set of items, and a taxonomy on the items, we find *association rules* (6) between items at any level of the taxonomy. Consider a collection of items and a set of records, each of which contain some number of items from the given collection. For example, a rule such as "people who buy automobiles tend to buy gasoline" may hold even if rules that "people who borrow automobiles tend to buy gasoline," and "people who buy electric cars tend to buy gasoline" do not hold. Thus an *association* approach operates against our set of records and return affinities or patterns that exist among the collection of items. Such patterns can be expressed by rules such as "63% of all the records examined that contain items, 1, 2 and 3 also contain items 4 and 5," where 63% refers the confidence factor of the rule.

IBM has identified a common application that can be built using association rules called *market basket analysis.* Initially, market-basket analysis treated the purchase of a number of items as a single transaction in order to find trends across large numbers of transactions so as to understand and exploit consumer buying patterns. Information from this analysis can then be used to adjust inventories, modify display or inventory placements. Association approaches can be applied equally well to services that develop targeted marketing campaigns or determine common (or uncommon) practices. In the financial sector, association approaches can be used to analyze customers' account portfolios and identify sets of financial services that people often purchase together. This explains the case where a retail operator wishes to determine, from his transaction database, the set of product identifiers listed under the same transaction identifier. An association operator can discover this information over the point of sales transaction log, which contains among other information, transaction identifiers and product identifiers. Thus, by employing an association approach, the market basket analysis application can determine affinities such as "20% of the time that a specific pay-per-view service is subscribed, viewers also buy a set of batched services, specific to their geographic location." If we express resultant item affinities in these terms, we are expressing a confidence rating. Thus when we use a rule such as, "80 percent of all sales in which beer was purchased also included potato chips" we can set a confidence threshold to eliminate discovery of all but the most common trends. Results of the association analysis (for example, the attributes involved in the rules themselves) may trigger additional analysis.

Another *association rule* example is the analysis of claims submitted by patients to a medical services insurance agency. Each claim contains information about medical procedures that were performed on a given patient during one visit. By defining the set of items to be the collection of all medical procedures that can be performed on a patient and the records to correspond to each claim form, the application can find, using the association function, relationships among medical procedures that are often performed together.

The most commonly applied database mining function, *classification,* employs a set of preclassified examples that can classify the database records at large. Detecting fraud and identifying credit-risk applications are activities particularly well suited to this type of analysis. Discovering interesting patterns in various grants information system databases is also a good application for classification approaches. For example, the Natural Sciences and Engineering Research Council (NSERC) of Canada was able to discover that, although hardware designs from Quebéc received their proportion of the number of grants according to population demographic information available to Council, Quebécers far outpaced the rest of Canada in the amount of funds received per grant: simply put, Quebéc has good hardware designers.

Some *classification* approaches use *decision tree* or *neural network*-based classification algorithms. In these approaches the classification algorithms require a training set of preclassified example transactions, which the classifier training algorithm uses to determine the set of parameters required for proper discrimination. Classification approaches encode these parameters into a model called a *classifier.* A classifier can be used predictively to classify new records into these same predefined classes.

Data evolution regularities or *sequence-based analysis* can best be illustrated as an analysis that deals with a collection of items as part of a point-in-time transaction. A problem occurs when there is additional information to coalesce, say, a sequence of purchases (for example, an account number, a credit card, or a frequent flyer number) over time. In this situation, not only may the coexistence of items within a transaction be important, but also the order in which those items appear across ordered transactions and the amount of time between transactions. Rules which can capture these relationships can be used, for example, to identify a typical set of harbinger purchases that might predict a specific subsequent purchase.

In addition, the contents of a database may change over time, and it may be important to catch *data evolution regularities* in a dynamically evolving database. The study of data evolution regularities is appealing, since users are often interested in finding regularities or trends of data evolution rather than examining a large volume of data over time in a database. For example, it is interesting to find the characteristics of the growth or shrinkage of certain mutual funds in a stock market or to discover the trend of changes in some census data or weather patterns.

Both database contents and database structures (schemes) may evolve over the lifetime of a database. Issues on schema evolution have been studied in multi-database, heterogeneous database and object-oriented database research (7–10). This kind of evolution introduces an extra dimension of complexity; for that reason most researchers focus on the evolution of database contents and assume that the database schemes are stable, not evolving over time. Extensions of the data evolutionary regularity approaches to nested relational, deductive, and temporal databases are briefly discussed in (11) which, in principle, should generalize to object-oriented, heterogeneous, and multimedia databases.

In a dynamically evolving database, data evolution may involve a large volume of data. To discover data evolution regularity, actual evolving data should be first extracted from the database. Then, database mining techniques can induce generalized rules or trends of evolution from the extracted data. For example, an attribute-oriented generalization method, which has been developed to discover knowledge rules in relational databases (integrating the learning process with database operations), can be extended to the discovery of data evolution regularities. In addition to extracting *classification rules,* which summarize the general characteristics of a set of data that satisfy certain data evolution criteria, such as the characteristics of mutual funds whose capital gain increased over 10% in 1996, and *discrimination rules,* which distinguish the general properties of a set of evolving data from a set of contrasting data, where the contrasting data can be a set of stable data or another set of evolving data, such as the rule that distinguishes the top-10 performers of this year's mutual funds from those in the last year, we can detect the general trend of evolution. We do so by describing how a particular set of data evolves over a period of time, for example, how the stock price changes for computer companies over the past six months. A detailed example of this phenomenon is given in (11).

A relatively large number of approaches, which assign database records with a great number of attributes into a smaller set of groups or "segments," have been called *clustering approaches.* Normally one of the early steps performed in the database mining process, clustering occurs automatically and identifies the distinguishing database characteristics, subsequently partitioning the space defined by database attributes along natural boundaries which can be used as a starting point for exploring further relationships.

Clustering is best used for finding groups of items that are similar and support population segmentation models, such as demographic-based customer segmentation. These groups can be fixed in advanced (*supervised* clustering) or determined by the system (*unsupervised* clustering). Subsequent additional analytical and other database mining analyses can determine characteristics of these segments with respect to some desired outcome. Thus, the buying habits of multiple population segments might be cross-referenced and compared with determine targeting strategies for a new sales campaign. Clustering applications include direct mailing, risk analysis (finding groups that exhibit a similar payment history pattern), medical diagnosis, retail analysis (finding products that sell with a similar seasonal pattern).

Estimation methods are another variation on the classification approach in which "scores" are created along various data dimensions. Thus the results of data mining using estimation techniques can be used to determine, say, a more general assessment of health-risk indicators for an individual or group of individuals, rather than merely determining if someone is a good or bad health risk (binary classification).

Often the approaches discussed thus far are used in association with each other and in association with other analytical techniques including, but not limited to, decision trees, neural networks, rule induction, case-based reasoning, fuzzy logic, genetic algorithms, and fractal transforms. In many cases, the technique is rooted elsewhere: in neurobiology (neural nets), evolution theories (genetic algorithms), formal logic (analytic methods), heuristic search (rule induction), and studies of human memory (case-based reasoning).

Decision trees represent an efficient and easy technique for database mining, which offer a synthetic view of data. They can handle large numbers of records with many fields with predictable response. Decision trees work with symbolic and numerical data, require no special knowledge, and perform basic navigation, modeling and prediction within the same formalism.

Consider the example of a credit officer in a bank. Data about last year's customers who were granted a small loan are available to the officer, where the customers are described by age, wages, status, and the like. One field, success, in the database, shows whether the customer had trouble paying back the loan. A decision tree analysis program, after importing the data, builds a tree, with the leftmost node as the root of the tree and the rightmost nodes as leaves. Each node contains a subset of the initial population, with the root containing the entire population. Nodes can display a variety of information. For example, in the decision tree shown in Fig. 1, each node contains the number of customers in the node, the number and percentage of customers in this node with trouble repaying the loan, and the number and percentage of customers in this node with no trouble paying back the loan.

The most discriminating field is used to split the population, that is, the field that best separates problem customers from no-problem customers. The population in a node is split according to the value of a field; in this case, the customers are split according to housing type. The process is repeated, isolating subsets of customers with higher success (or failure) rate.

Reading the decision tree from root to leaves, we determine that people who rent their home and earn more than $35,000 have an 85% success rate repaying their loans, whereas people who do not rent and who applied after seeing a TV-commercial for the loan have a 12% success rate.

Neural networks establish prediction models by clustering information into groups and predicting into which groups new records will aggregate. An initial training phase assists the neural network to "learn" where the training data is drawn from a subset of the data. The predictive capability or accuracy of the neural network can then be determined by authenticating the trained network against multiple other subsets of the data, and this process is repeated until the predictive power remains relatively stable. Neural networks have been used effectively to model nonlinear data, noisy data, or data with missing values. Since neural networks work on numeric data, symbolic or continuous data must first be discretized.

Representing knowledge in comprehensible condition-action rules, *rule induction* (RI) learns general domain-specific knowledge from a set of training data. Most RI systems con-

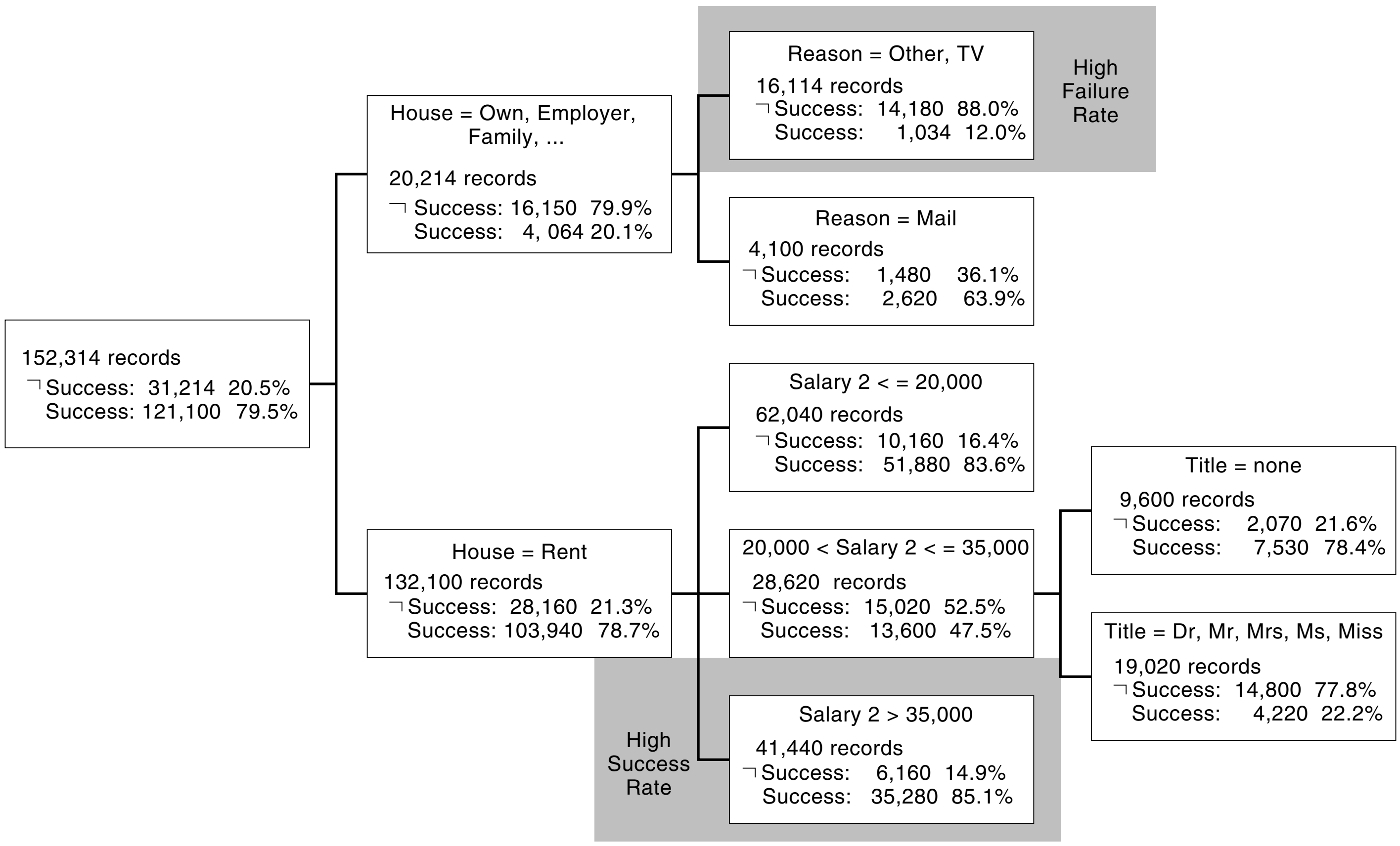


Figure 1. An example of a decision tree.

duct heuristic search through the hypothesis space of rules or decision trees. RI systems typically use a statistical evaluation function to select attributes or attribute-value pairs for incorporation into the knowledge base. A noise handling technique *pre-* or *postpruning* removes imperfect or noisy training data. Many RI systems have been developed and applied to real-world domains to discover knowledge from observed data. Example systems include C4.5 (12), AQ15 (13), and CN2 (14); Clark (15) provides an overview of RI techniques and strategies for noise abatement. Despite their successes, RI systems do not recognize exceptions well, nor do rules represent continuous functions well.

Case based reasoning (CBR) represents knowledge by storing descriptions of previously experienced, specific cases. Previous solutions are adapted to solve new cases by retrieving similar past cases. Common retrieval schemes are variations of the nearest neighbor method, in which similarity metrics are used to identify cases *nearest* to the current case. An overview of the foundational issues related to CBR is presented in (16). Although CBR is a relatively new learning method, a number of commercial tools have been developed.

CBR can learn nonlinearly separable categories of continuous functions and CBR is incremental by nature, unlike most inductive methods, which have difficulty extending or refining their rule set during the problem-solving stage. CBR, however, does have limitations: it does not yield concise representations of concepts which can be understood easily by humans, and CBR systems are usually sensitive to noise.

Genetic algorithms employ optimization techniques that use processes such as genetic combination, mutation, and natural selection in a design based on the concepts of natural evolution. They can generate potential cases, based on a loosely constructed model, where such cases may converge to the “best” example (based on any number of considerations). Typical applications of genetic algorithm include direct marketing strategies, where it is desirable to know the optimal profile of the ideal customer who is likely to invest more than $25,000 in mutual funds per year, or risk analysis, medical diagnosis, and the like.

Fractal transforms hold interesting promise as a technique to employ in database mining. They are typically used in lossless data compression algorithms. Thus the possibility exists that pattern-matching approaches based on these techniques can exploit substantially reduced dataset sizes to increase performance.

Every one of these techniques have advantages and disadvantages in terms of their performance, training requirements if any, the problem characteristics they address, their discrimination capabilities, and so on. Most algorithms are often tunable, using a variety of parameters aimed at providing better performance. Also most approaches to database mining employ some hybrid admixture of these techniques.

Issues

Database mining is attractive to executives and professional analysts who need to make sense out of large masses of complex data. Programs that can analyze an entire database and identify the relationships relevant to the executive are touted as a panacea for all data analysis problems. This situation has not yet been realized.

Database mining techniques and algorithms in contemporary use have evolved from earlier research into pattern recognition and machine learning within an artificial intelligence paradigm. Current database mining programs have concentrated on the development of fast, responsive algorithms, which can handle very large databases; the human computer interface and features which make the use of such tools attractive to business users have only more recently attracted sufficient attention as to merit nontrivial development efforts.

This current state of affairs presents challenges to researchers and users, some of which are indicated below.

- *The Tools Gap.* This gap is due to a number of factors. Most database mining systems require significant pre- and postprocessing of data in order to operate. Preprocessing activities involve any operations required to gain task relevant data for analysis by the database mining program, from the selection of pertinent subset(s) of data to complex data transformations to bridge any *representational gap.* Postprocessing often involves selection of subsets of the results of database mining and the application of visualization methods for reporting purposes.
- *Limited or Inappropriate Information.* Often designed for purposes different from database mining, databases do not always possess the attributes that would simplify the learning task. Another example is nonconclusive data, in which attributes essential to the application domain are not present in the database. For example, we cannot diagnose malaria from a medical database if it does not contain red blood cell counts.
- *Missing Data, Noise, Dirty Data.* Most database mining programs lack a higher-level data model and thus have no domain-specific (semantic) structure; as such, they assume all information must be factual. Users must take precautions to ensure that the data under analysis are "clean," which could require detailed analysis of the attribute values fed to the discovery system. This function is generally performed when the data warehouse, if any, is created nowadays. Databases often are tainted with errors in the data. Attributes which rely on subjective or measurement conjectures may misclassify results. Errors in either attribute values or class information are called *noise.* Missing data are handled by simply disregarding missing values or omitting the corresponding records; by inferring missing values from known values or via procedure invocation in object-oriented databases or by default value substitution, or average over the missing values using Bayesian techniques. Statistical techniques also can treat noisy data and separate different types of noise.
- *Uncertainty, Updates, and Irrelevant Fields.* Uncertainty refers to the severity of the error and the degree of precision in the data. Dynamic databases have ever-changing contents as data are added, modified, or removed. How can we ensure that the rules are up-to-date and consistent with the most current information? The discovery system should also be time-sensitive since some data values vary over time and the system is affected by the data's *timeliness.* Another example illustrates relevance: postal code fields are fundamental to studies that establish a geographical connection to a product offering and the sales of a product.
- *Explanatory Dyslexia.* Many database mining tools perform their analyses using complex algorithms that are not easily understood by users; for example, they don't always generate "if-then" rules that use the original data's attributes by name. These systems cannot easily "explain" their results. Approaches capable of generating the desired information about the underlying attributes, such as decision trees and rule induction methods, may require nontrivial additional postprocessing and/or visualization.
- *Data Representation Gap.* Modern database mining systems obtain source data from large relational database systems in which the information is partially normalized and the attributes mined span multiple tables. Sometimes database mining engines supply the conditioning code to provide the denormalized representation they require. Large central fact tables in data warehouses designed using star schema often combine denormalized data into one flat table. Sometimes, the database mining tools may require discretized continuous variables or remap time-series information.
- *Flexible Databases.* Parallel relational database systems possess data that are distributed over multiple disks and accessed by many CPUs. New database architectures require significant preprocessing for subsequent use by a data mining program. This conditioning requirement increases as new systems tempt database designers, such as functional, applicative, object-oriented, distributed, concurrent, parallel, inferential, associative, procedural, connectionist, declarative, networked, non-monotonic, temporal, holographic, etc., running on new hardware, firmware, wetware, neuralware, etc.

For the remainder of this exposition, we will present data mining concepts and examples in the context of relational databases, drawing distinctions whenever the relational database context is not wholly appropriate.

The research and development goals that were proposed for one of the earliest data mining systems, the DBLEARN system (17–19) is an instructive place to begin. DBLEARN designers outlined the following goals:

- Knowledge discovery should be performed efficiently in a variety of databases, including new generation database systems (extended-relational, deductive, object-oriented and active databases), and new database applications (including spatial, engineering, and multimedia databases);
- Knowledge to be discovered includes characteristic rules, discriminant rules, data dependency rules, data evolution regularities, quantitative rules (credibility association), etc.;
- The system applies different learning techniques, including attribute-oriented induction, constrained induction, inductive logic programming, etc., integrates well with

existing database systems with high performance, and is robust at handling noise and exceptional data and at dynamic discovery and adjustment of concept hierarchies; and
- Discovered knowledge will be applied to intelligently querying data and knowledge, classification of unknown cases, diagnostic decision making, and control of dynamic processes.

We explain the basic database mining process by describing DBLEARN, in order to illustrate commonplace data mining techniques and principles that are in general usage currently and were embodied in the DBLEARN system design. Subsequently we will illustrate increasingly sophisticated data mining techniques, utilizing more advanced systems and examples.

INTRODUCING THE DATABASE MINING PROCESS: DBLEARN

DBLEARN's designers (17,20) promoted and developed an attribute-oriented generalization method for knowledge discovery in databases. The method integrates machine learning exemplars, especially learning-from-examples techniques (21), with set-oriented database operations and extracts generalized data from actual data in databases. An attribute-oriented concept tree ascension technique is applied in generalization, which substantially reduces the computational complexity of database learning processes. Different kinds of knowledge rules, including characteristic rules, discrimination rules, quantitative rules, association rules, and data evolution regularities, can be discovered efficiently using the attributed-oriented approach. In addition to learning in relational databases, the approach can be and has been applied to knowledge discovery in nested relational and deductive databases. Learning can also be performed with databases containing noisy data and exceptional cases using database statistics. Furthermore, rules discovered can be used to query database knowledge, answering cooperative queries and facilitating semantic query optimization.

Database Mining Primitives

Data mining in relational databases requires three primitives for the specification of a discovery task: task-relevant data, background knowledge, and the expected representations of the learned results. We can subsequently generalize our results from relational databases to other databases as well.

Characterizing the features of science graduate students requires only data relevant to science graduates, but these data may extend over several relations. Thus, a query can be used to collect *task-relevant data* from the database. Task-relevant data can be viewed as examples for learning and learning-from-examples is an important strategy for knowledge discovery in databases. Most learning-from-examples algorithms partition the set of examples into positive and negative sets and perform generalization using the positive data and specialization using the negative ones. Unfortunately, a relational database does not explicitly store negative data, and thus no explicitly specified negative examples can be used for specialization. Therefore, a database induction process relies only on generalization, which must be performed cautiously, to avoid overgeneralization.

After the desired database tables have been selected and the task relevant data have been identified, it is sometimes necessary to transform the data. The type of data mining operation performed and the data mining technique used dictate the transformations, which vary from conversions of one type of data to another, for example, converting nominal values into numeric ones so that they can be processed by a neural network, to new attribute definition, that is, derived attributes.

Concept hierarchies represent background knowledge necessary to control the generalization process. Different levels of concepts can be organized into a taxonomy of concepts which is partially ordered according to a general-to-specific ordering. The most general concept is the null description, described by a reserved word "ANY," and the most specific concepts correspond to the specific values of attributes in the database (17). Using a concept hierarchy, the rules learned can be represented in terms of generalized concepts and stated in a simple and explicit form, which is desirable to most users.

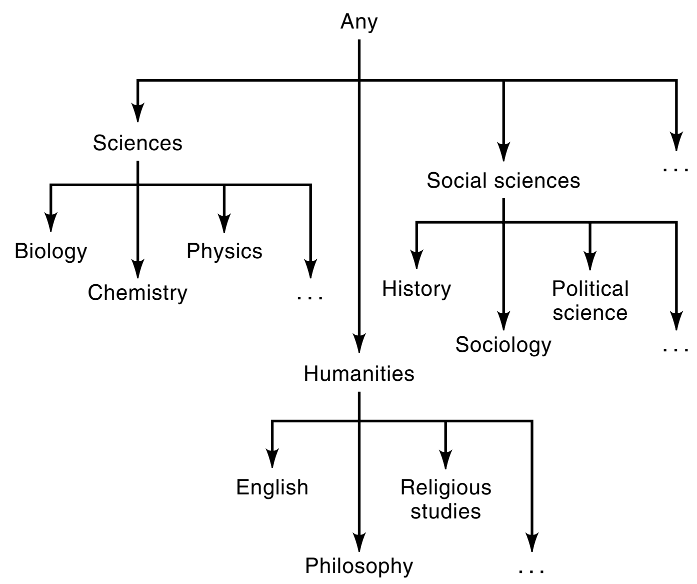
A concept hierarchy table of a typical university database for three attributes is shown in Table 1. Concept hierarchies can be provided by knowledge engineers or domain experts. Moreover, many conceptual hierarchies are actually stored in the database implicitly. For example, the information that "Vancouver is a city of British Columbia, which, in turn, is a province of Canada," is usually stored in the database if there are "city," "province," and "country" attributes. Such hierarchical relationships can be made explicit at the schema level by indicating "city province country." The taxonomy of all the cities stored in the database can be retrieved and used in the learning process.

Some concept hierarchies can be discovered automatically or semi-automatically. Numerical attributes can be organized as discrete hierarchical concepts, and the hierarchies can be constructed automatically based on database statistics. Such automatic construction can be performed by first obtaining the distribution of attribute values in the database, then setting the range of the values and performing refined classifications in tightly clustered subranges. For example, for an attribute "GPA" (grade point average), an examination of the values in the database discloses that GPA falls between 0 to 4, and most GPA's for graduates are clustered between 3 and 4. One may classify 0 to 1.99 into one class, and 2 to 2.99 into another, but give finer classifications for those between 3 and 4. Even for attributes with discrete values, statistical techniques can be used under certain circumstances. For example, if the birthplace of most employees are clustered in Canada and scattered in many different countries, the highest level concepts of the attribute can be categorized as "Canada" and "foreign." Thus, the available concept hierarchies can be modified based on database statistics. Moreover, the concept hierarchy of an attribute can also be automatically discovered or refined based on its relationship with other attributes.

Different concept hierarchies can be constructed on the same attribute based on different viewpoints or preferences. For example, the birthplace could be organized according to administrative regions such as provinces or countries, geographic regions such as east-coast, west-coast, or the sizes of the city, such as, metropolis, small-city, town, countryside, and so on. Usually, a commonly referenced concept hierarchy is associated with an attribute as the default concept hierar-

Table 1. Example Concept Hierarchy Tables

Attribute	Concept	Values
Major	Sciences	Biology, Chemistry, Physics, . . .
	Humanities	English, Philosophy, Religious studies, . . .
	Social Sciences	Political Science, Sociology, History, . . .
	Any	Science, Humanities, Social Sciences, . . .
Birth-place	British Columbia	Vancouver, Victoria, Richmond, . . .
	Alberta	Edmonton, Calgary, Red Deer, . . .
	Saskatchewan	Regina, Saskatoon, Moose Jaw, . . .
	Any	British Columbia, Alberta, Saskatchewan, . . .
GPA	Excellent	80, 81, . . ., 100
	Above average	70, 61, . . ., 79
	Average	60, 61, . . ., 69
	Any	Excellent, Above average, Average, . . .



chy for the attribute. Other hierarchies can be selected explicitly by users.

Rules are one of the expected forms of the learning results. Many different rules can be discovered by database mining. A *characteristic rule* is an assertion which characterizes a concept satisfied by all or a majority number of the examples in the class undergoing learning (called the target class). For example, the symptoms of a specific disease can be summarized by a characteristic rule. A *discrimination rule* is an assertion that discriminates a concept of the class being learned (called the *target class*) from other classes (called *contrasting classes*). For example, to distinguish one disease from others, a discrimination rule should summarize the symptoms that discriminate this disease from others. Furthermore, *data evolution regularities* represent the characteristics of the changed data if it is a characteristic rule, or the features that discriminate the current data instances from the previous ones if it is a discrimination rule. If quantitative measurement is associated with a learned rule, the rule is called a *quantitative rule*. A quantitative rule is a rule associated with quantitative information, which assesses the representativeness of the rule in the database.

In learning a characteristic rule, relevant data are collected into one class, the target class, for generalization. In learning a discrimination rule, it is necessary to collect data into two classes, the target class and the contrasting class(es). The data in the contrasting class(es) imply that such data cannot be used to distinguish the target class from the contrasting ones, that is, they are used to exclude the properties shared by both classes.

Each tuple in a relation represents a logic formula in conjunctive normal form, and a data relation is characterized by a large set of disjunctions of such conjunctive forms. Thus, both the data for learning and the rules discovered can be represented in either relational form or first-order predicate calculus.

A relation which represents intermediate (or final) learning results is called an intermediate (or a final) generalized relation. In a generalized relation, some or all of its attribute values are generalized data, that is, nonleaf nodes in the concept hierarchies. Some learning-from-examples algorithms require the final learned rule to be in conjunctive normal form (22). This requirement is usually unrealistic for large databases, since the generalized data often contain different cases. However, a rule containing a large number of disjuncts indicates that it is in a complex form and further generalization should be performed. Therefore, the final generalized relation should be represented by either one tuple (a conjunctive rule) or a small number (usually 2 to 8) of tuples corresponding to a disjunctive rule with a small number of disjuncts. A system may allow a user to specify the preferred generalization threshold, a maximum number of disjuncts of the resulting formula.

Exceptional data often occur in a large relation. The use of statistical information can help learning-from-example handle exceptions and/or noisy data. A special attribute, *vote,* can be added to each generalized relation to register the number of tuples in the original relation, which are generalized to the current tuple in the generalized relation. The attribute vote carries database statistics and supports the pruning of scattered data and the generalization of the concepts that take a majority of votes. The final generalized rule will be the rule which either represents the characteristics of a majority number of facts in the database (called an approximate rule), or in a quantitative form (called a quantitative rule), indicating the quantitative measurement of each conjunct or disjunct in the rule.

The steps comprising the basic database mining process, illustrated in Fig. 2, depict an interactive session between a user and a database mining system. After selecting and preprocessing (by any of a number of mechanisms, as will become clear later) the task relevant data, the user is able to manipulate (generalize, specialize, etc.) the task relevant data until the knowledge sought is found and reported, if possible.

Attribute-Oriented Generalization

Attribute-oriented generalization is performed attribute by attribute using attribute removal and concept tree ascension

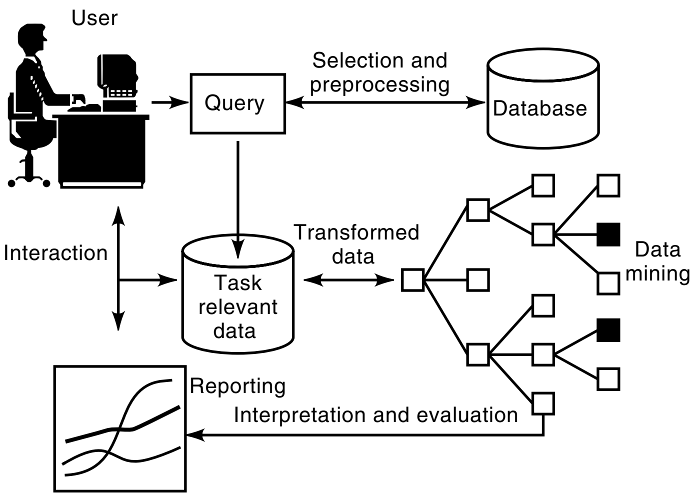


Figure 2. Overview of the database mining process.

as summarized below. In fact, seven strategies are utilized when performing attribute-oriented induction: (1) generalization on the smallest decomposable components; (2) attribute removal; (3) concept tree ascension; (4) "vote" propagation; (5) attribute threshold control; (6) generalization threshold control; and (7) rule transformation. See (17) for details. As a result, different tuples may be generalized to identical ones, and the final generalized relation may consist of a small number of distinct tuples, which can be transformed into a simple logical rule. Basic attribute-oriented induction is specified in Algorithm 1.

This basic attributed-oriented induction algorithm extracts a characteristic rule from an initial data relation. Since the generalized rule covers all of the positive examples in the database, it forms the necessary condition of the learning concept, that is, the rule is in the form: learning_class(x) → condition(x), where "condition(x)" is a formula containing "x". However, since data in other classes are not taken into consideration in the learning process, there could be data in other classes which also meet the specified condition. Therefore, "condition(x)" is necessary but may not be sufficient for "x" to be in the learning class.

Attribute-oriented generalization can also be applied to learning other knowledge rules, such as discrimination rules, data evolution regularities, and so on. Since a discrimination rule distinguishes the concepts of the target class from those of contrasting classes, the generalized condition in the target class that overlaps the condition in contrasting classes should be detected and removed from the description of discrimination rules. Therefore, a discrimination rule can be extracted by generalizing the data in both the target class and the contrasting class synchronously and by excluding the properties that overlap in both classes in the final generalized rule.

Algorithm 1. Attribute-oriented induction in relational databases.

Input: (i) A relational database, (ii) a concept hierarchy table, and (iii) the learning task, and optionally, (iv) the preferred concept hierarchies, and (v) the preferred form to express learning results.

Output: A {characteristic, discrimination, . . .} rule learned from the database.

Method: Attribute-oriented induction consists of the following 4 steps:

Step 1. Collection of the task-relevant data.
Step 2. Basic attribute-oriented induction.
Step 3. Simplification of the generalized relation, and
Step 4. Transformation of the final relation into a logical rule.

Notice that the basic attribute-oriented induction (Step 2) is performed as follows.

begin for each attribute A_i ($1 < i < n$, # of attributes) in the generalized relation **do**
while number_of_distinct_values_in_A_i > generalization_threshold **do**
begin
if no higher level concept in the concept hierarchy table for A_i

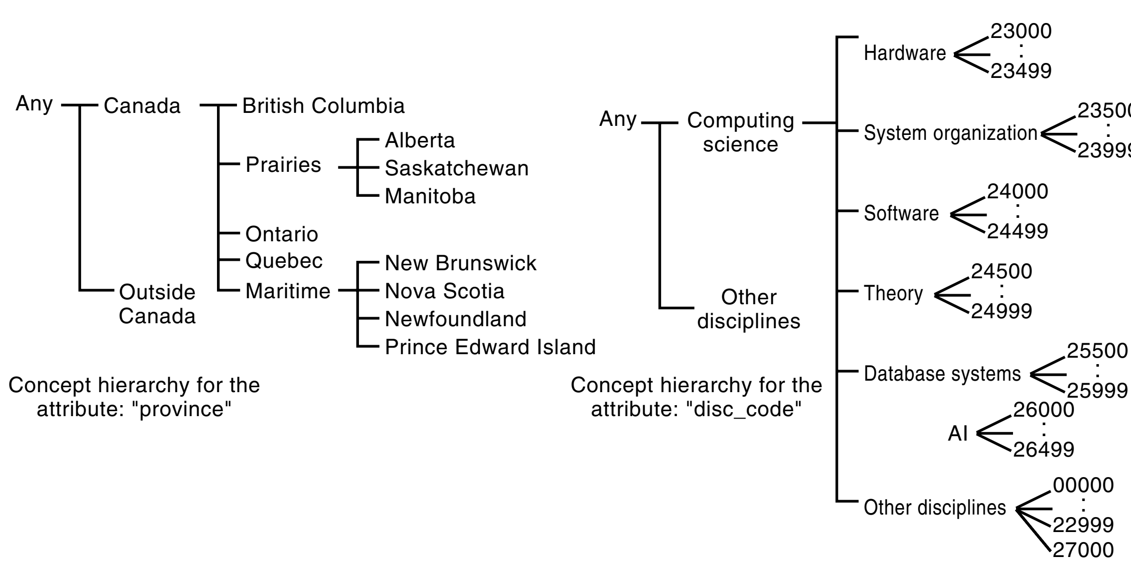


Figure 3. Sample concept hierarchies.

then remove A_i
else substitute for the values of A_i's by its corresponding minimal generalized concept;
merge identical tuples
end
while number_of_tuples_in_generalized_relation > generalization_threshold **do**
selectively generalized some attributes and merge identical tuples
end. {Attribute-oriented induction}

Some Examples of Database Mining Using Attribute-Oriented Generalization in DBLEARN

Consider the following example: Suppose that the learning task is to learn characteristic rules for graduate students relevant to the attributes Major, Birth_place, and GPA, using the conceptual hierarchy shown earlier and a threshold value of 3. The learning task is presented to DBLEARN as

```
in relation Student
  learn characteristic rule for
    Status = ''graduate''
  in relevance to Name, Major, Birth_place, GPA
```

Representation of the learning result takes the following form: each tuple in a relation is a logical formula in conjunctive normal form, and a data relation is characterized by a set of disjunctions of such conjunctive forms. The number of disjuncts, and thus the amount of generalization, is controlled by a user-specified threshold value. After applying the appropriate strategies (generalization on the smallest decomposable components, attribute removal, concept tree ascension, etc.), we might end up with (sample concept hierarchies are shown in Fig. 3):

```
∀x graduate(x) →
  {Birth_place(x) ∈ Canada ^ GPA(x) ∈
    excellent} [75%] |
  {Major(x) ∈ science ^ Birth_place(x) ∈
    foreign ^ GPA(x) ∈ good} [25%]
```

For another example, consider the query below, QUERY, which illustrates DBLEARN's discovery of the characteristics of computing science operating grants for artificial intelligence research by amounts, geographical area, and the percentages of grants awarded in a given discovery category and the percentage of funds awarded for the discovered category. It is not possible to structure a single SQL query to discover and report results of this query using the database illustrated in Fig. 4. The NSERC Grants Information System contains a database of information about the grants that are awarded by NSERC. The central table in the database had 10,087 tuples with 11 attributes when this example was created, a relatively small database. Concept hierarchy tables for the attributes "province" and "disc_code" for the NSERC database are shown above.

```
QUERY: learn characteristic rule for
  ''CS_Op_Grants''
  from Award A, Organization O, grant_type G
    where O.org_code = A.org_code and
      G.Grant_order = ''Operating Grants''
    and A.grant_code = G.grant_code and
      A.disc_code = ''Computer''
    in relevance to amount, province,
      prop(votes)*, prop(amount) [prop() is a
      built-in function which returns the number
      of original tuples covered by a
      generalized tuple in the final result and
      the proportion of the specified attribute
      respectively.]
    using table threshold 18
Result of query: an early DBLEARN version
  provided tabular output in response to QUERY
```

Amount	Geography Area	# of Grants	Prop. of Amount
0-20Ks	B.C.	7.4%	4.7%
0-20Ks	Prairies	8.3%	5.4%
0-20Ks	Quebec	13.8%	8.7%
0-20Ks	Ontario	24.5%	15.7%
0-20Ks	Maritime		
20Ks-40Ks	B.C.	5.3%	7.0%
20Ks-40Ks	Prairies	5.3%	6.6%
20Ks-40Ks	Quebec	5.1%	7.0%
20Ks-40Ks	Ontario	12.9%	16.0%
20Ks-40Ks	Maritime	1.0%	1.3%
40Ks-60Ks	B.C.	1.2%	3.1%
40Ks-60Ks	Prairies	0.2%	0.4%
40Ks-60Ks	Quebec	1.0%	2.5%
40Ks-60Ks	Ontario	5.1%	11.5%
60Ks-	B.C.	0.2%	0.6%
60Ks-	Prairies	0.4%	1.6%
60Ks-	Quebec	0.2%	0.6%
60Ks-	Ontario	1.2%	4.5%
Total:	$10,196,692	100%	100%

Several points are worth noting at this time:

- The general framework presented for knowledge discovery in databases (database mining) has been attribute-oriented generalization.
- Attribute-oriented generalization takes advantage of the organization of relational database systems.
- The concept tree ascending technique follows from version spaces, typical of learning from examples from machine learning paradigms.
- The version space method is tuple-oriented generalization; DBLEARN's method uses concept hierarchies of each attribute as a factored version space and performs generalization on individual attributes, significantly increasing processing efficiency.
- If there are p nodes in each concept tree and k concept trees (attributes) in the relation, the total size of the factored version space should be p^k—attribute-oriented generalization thus has a much smaller search space than tuple-oriented generalization.

DATA WAREHOUSES AND OLAP

Data Warehouses

To fully take advantage of database mining and decision support systems generally, the appropriate data should first be

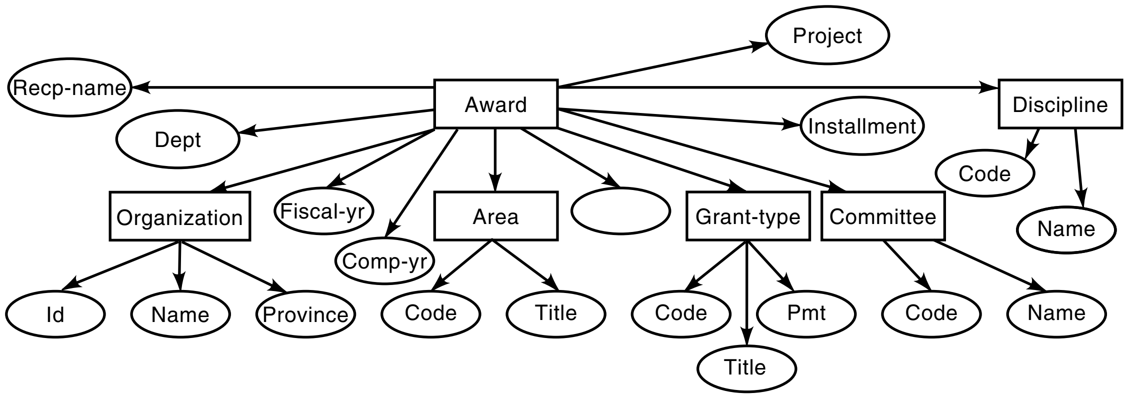


Figure 4. Canada's Natural Sciences and Engineering Research Council (NSERC) Grants Information System (database relations are enclosed in rectangular boxes; attributes are enclosed in ovals).

collected and stored in a *data warehouse.* A data warehouse is a relational database management system (RDMS) designed specifically to serve as a centralized data repository which can be queried. Data warehousing makes it possible to extract archived operational data and correct inconsistencies between different data formats. Data warehouses also can assimilate additional, sometimes expert information.

So what is the difference between a data warehouse and a database? Sometimes these two concepts become confused. Both databases and data warehouses store data so that applications can share and access the data. What, then, makes a data warehouse so different?

One important distinction is the peculiarity that a data warehouse holds *read-only* data. Inasmuch as databases contain operational data, many of the decision support applications that are associated with the data warehouse put too large a burden on the databases that run them. Nonetheless, this limited view of the data warehouse does not include all features normally associated with them. Two additional features generally associated with data warehouses include: information stored in the data warehouse derives from different disparate sources and that information is harmonized in the warehouse; and more than one different application program will make use of the same information. Considering this view, we begin to view the data warehouse as an environment and not an application, whereas a database is usually part of some application. In the data warehouse, the application is nonexistent and is supplanted by data mining programs.

Data warehouses generally share the following characteristics:

- *Topic-Oriented.* Data are organized topically rather than by application. For example, a medical laboratory using a data warehouse would organize their data by customer, insurance premium, and claims, instead of by different products.
- *Integrated.* Normally data from many separate applications are often inconsistently encoded. For example, in one application, gender might be coded as "m" and "f," in another by "M" and "F," and in another by 0 and 1. When data are moved from the source into the data warehouse, they assume a consistent encoding.
- *Time-Variant.* The data warehouse contains older data that are to be used for comparisons, trends, and forecasting. These data are not updated.
- *Nonvolatile.* Once data enter the data warehouse, they are not updated or modified; they are only copied and accessed.

It should prove instructive to consider briefly the processes involved in data warehousing. The first step is to insulate current operational data. The data warehouse retrieves data from a variety of heterogeneous operational databases, which is usually transformed and delivered to the data warehouse based on some selected data model. The model and definition of the source data is called *metadata* and is used to retrieve and understand the data in the data warehouse. Metadata contain the structure description of the data; the algorithm used for summarization; and the mapping from the operational environment to the data warehouse. Data cleansing is an important second step in the process, that is, the removal of certain aspects of operational data, which greatly increase query response times. Cleansing is normally a dynamic step in order to reconcile all types of queries. The data are now ready to be transferred to the data warehouse, typically a large database on a very high-performance computer, possibly special hardware. Data marts may then be created. Data marts essentially are small warehouses which provide possibly summarized subsets of the data warehouse. Often these data marts use multidimensional databases which significantly reduce query processing time.

Data are structured in a data warehouse to permit distinct, separate levels of summarization to be performed. Current detail data are normally voluminous and placed at the lowest level of granularity normally accessible via expensive and complex fast access disk storage. Legacy and older detail data, which are infrequently accessed, are normally stored separately on mass storage devices at a level consistent with current detail data. Data derived from low level detail data (lightly summarized data) are normally stored on disk. More highly summarized data are either kept outside of the data warehouse or are condensed and easily accessible. Metadata are stored in the data warehouse and provide the mapping as data are transformed for the operational database to the data warehouse and as a guide to summarization algorithms used between the detail data and summarized data.

In order not to be labeled unreliable, a data warehouse must meet certain performance requirements:

- *Load Performance and Processing.* Refers to the timely performance of periodic, incremental loading of voluminous new data into the data warehouse and efficiency considerations as to the data conversions, filtering, formatting, integrity checking, physical storage, indexing, and metadata updating.
- *Data Quality.* The warehouse must ensure consistency and referential integrity of data despite "dirty" sources and massive database sizes.
- *Query Performance.* Query processing efficiency must not decrease when using the data warehouse RDBMS.
- *Scalability.* The sizes of data warehouse are growing swiftly; data warehouses must support modular and parallel evolution, recovery mechanisms, and a layered storage hierarchy for handling growing masses of records to be stored.
- *Warehouse Maintenance.* Large scale and time-cyclic nature of the data warehouse demands administrative ease and flexibility of use. Such maintenance includes workload tracking and performance tuning to optimized for maximum performance.
- *Networking Capabilities.* Data warehouse systems should cooperate in a larger network of data warehouses.
- *Integrated Dimensional Analysis.* Multidimensional views support must be inherent in the data warehouse to permit fast, easy creation of precomputed summaries common in large data warehouses.

Specific additional criteria can certainly be added to this list, perhaps on specific data warehouse to data warehouse application or set of applications.

On-Line Analytic Processing (OLAP) and Multidimensional Analysis

Multidimensional analysis is a method of viewing aggregate measurement data (for example, sales, expenses, etc.) along a set of dimensions (e.g., product, brand, store, etc.). A multidimensional database (MDB) typically consists of dimensional information (usually similar to field names in a table, e.g., product), desired measurements which are aggregations for computation and display (e.g., average sales), and hierarchy information which impose structure along a set of dimensions (e.g., {province, region, country}) as a geographic hierarchy.

Effectively, MDBs can be considered to be a precomputation of the aggregation space that surrounds a relational system, with the addition of hierarchical meta-information. Removed from the hierarchies, the MDB contains no additional information than the relational database, but is designed for fast access to aggregate results by partially precomputing them.

The interface supporting the MDB must manage (traverse) the dimensions easily as the user requests aggregate information computations and integrate seamlessly with a query/reporting system.

On-line Analytical Processing (OLAP) is essentially fast analysis of shared multidimensional information. Because it is difficult to orient relational database management systems (RDBMS) for widespread use in the spectrum of database applications and because client/server architectures give organizations the opportunity to deploy specialized servers optimized to handle specific data management problems, something new had to be developed. Major classes of database applications are not serviced by RDBMSs. Oracle, for example, has built a media server for handling multimedia applications. Sybase uses an object-oriented DBMS designed to handle complex images and audio data. OLAP is another applications category in which database servers support common analytical operations including consolidation, drill-down, and "slicing and dicing." OLAP data servers can also go in the reverse direction and automatically display detail data which comprises consolidated (aggregated) data. This process is called *drill-downs.* The term OLAP was coined by E. F. Codd (23) and was defined by him as "the dynamic synthesis, analysis and consolidation of large volumes of multidimensional data."

Often multidimensional data and OLAP are used as synonyms. Nevertheless it is only when a multidimensional database can express complex calculations easily, promote intuitive navigation (via the conceptual hierarchies), and make fast responses to the user that a multidimensional database will be considered OLAP. To clarify this situation further, consider the following OLAP database composed of sales data aggregated by region, product type, and sales outlet. A typical OLAP query might wish to find all product sales in each region for each product type. An analyst interacting with the OLAP might further wish to find sales volume for each outlet within region/product classifications. Finally the analyst might want to perform year-to-year or quarter-to-quarter comparisons for each sales branch. If this process is to be carried out quickly and on-line, then it is an OLAP process.

OLAP applications and On-line Transaction Processing (OLTP) applications, which consist of a large number of relatively simple transactions, are quite different. OLTP servers handle production data accessed through simple queries while OLAP servers handle data accessed through an iterative analytical investigation. Both OLAP and OLTP both require special optimized servers for the two kinds of processing.

Before proceeding to more advanced techniques and example database mining systems that employ these advanced methods, it should be instructive to review the basic data mining process depicted in Fig. 2. Figure 5 illustrates how the more current database mining systems may operate. Obviously not all data mining programs employ all facets of the process as shown. However for conceptual purposes, we may consider the database mining process as starting with initial "raw" data through to the reporting of extracted discovered knowledge employing the following phases: *selection and preprocessing*—selecting or segmenting the data according to some criteria, for example, all students enrolled in courses, to determine subsets of data of interest and then "cleaning" the selected data to remove information unnecessary for the remainder of the process, for example, the gender of an expectant mother. Data may also be reconfigured to ensure a consistent format at this stage; *transformation*—the data are transferred to the next stage after possibly transforming it first so that additional information (overlays) may be added, such as the demographic information, and the data are made accessible; *data mining and/or OLAP*—discovering and extracting patterns from the data; and *interpretation and evalu-*

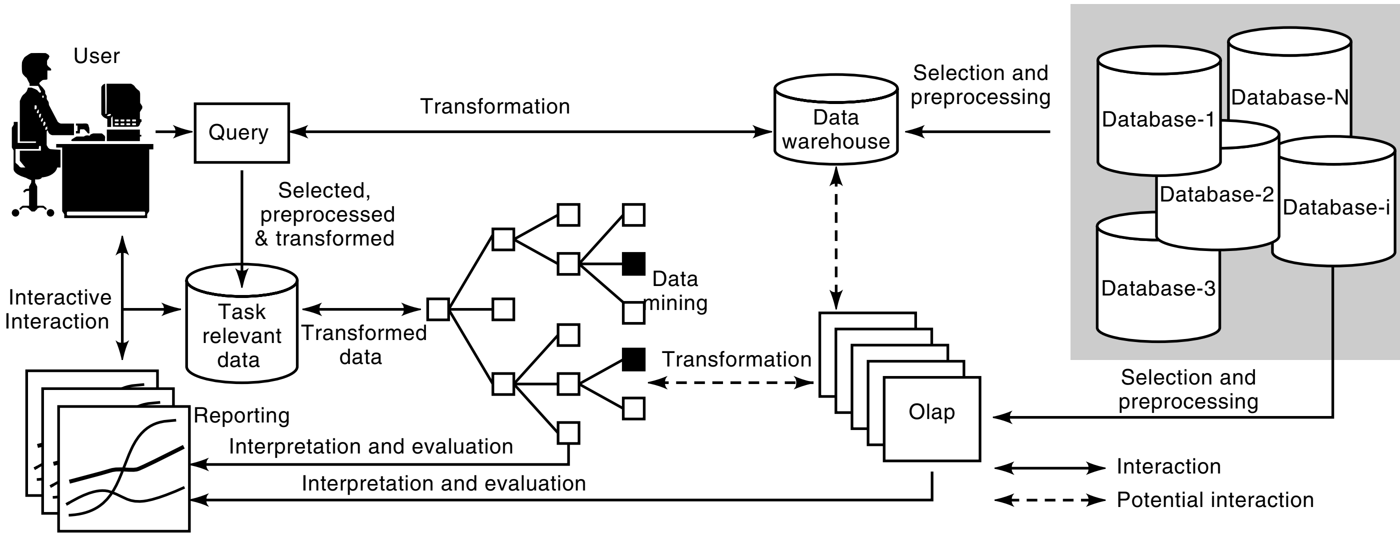


Figure 5. Revised overview of the data mining process with data warehouses and OLAP.

ation—identifying, evaluating, and finally interpreting the knowledge "mined" from the data and made accessible (presented) to the user. Sometimes results are returned in a manner to support decision making (prediction and classification), sometimes results summarize the data content or explain observations, sometimes the results confirm hypotheses tested by the user.

DATABASE MINING IN GREATER DETAIL

At this point it is desirable to look at the steps in database mining in greater detail. The following definitions and general principles are adhered to in this discussion:

- *Definition 1.* An attribute is generalizable if there are a large number of distinct values in the relation but there exists a concept hierarchy for the attribute (that is there are higher level concepts that subsume these attribute values). Otherwise it is nongeneralizable.
- *Definition 2.* An attribute in a relatively large relation is desirable for consideration for generalization if the number of distinct values it contains does not exceed a user-specified desirability threshold (~6 or less).

Table 2. An Animal World

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S
1	Tiger	Y	pt	fd	N	Claw	Meat	Y	N	Y
2	Cheetah	Y	pt	fd	N	Claw	Meat	Y	N	Y
3	Giraffe	Y	bt	sd	N	Hoof	Grass	Y	N	N
4	Zebra	Y	bt	sd	N	Hoof	Grass	Y	N	N
5	Ostrich	N	N	sd	Y	Claw	Grain	N	Y	N
6	Penguin	N	N	sd	Y	Web	Fish	N	N	N
7	Albatross	N	N	sd	Y	Claw	Grain	N	Y	Y
8	Eagle	N	N	fd	Y	Claw	Meat	N	Y	N
9	Viper	N	pt	fd	N	N	Meat	N	N	N

Abbreviations: H: hair, F: feather, T: teeth, S: swim, pt: pointed, bt: blunted, fd: forward, sd: side.

- Nongeneralizable attributes should be removed from the generalization process; this removal corresponds to Michalski (13) *dropping conditions.*
- Generalizable attributes are generalized to higher level concepts by concept tree ascension which corresponds to Michalski (13) *climbing generalization trees.*

A problem of DBLEARN was that it tended to overgeneralize. We overcome that problem in four discrete steps as follows. Consider the following task relevant data as shown in Table 2:

A *prime relation Rp* for a set of data R stored in the relational table is an intermediate relation generalized from relation R by removing nongeneralizable attributes and generalizing each attribute to a *desirability level.*

Application of Algorithm 2 yields the prime relation table, Table 3, using the concept hierarchy of the animal world, Fig. 6.

Algorithm 2: Extraction of the prime relation from a set of data R

Input: (i) set of task relevant data R, (ii) set of concept hierarchies H_i where H_i is a hierarchy on the generalized attribute A_i, (iii) set of desirability thresholds T_i for each attribute A_i

Output: The Prime Relation R_p

Table 3. Prime Relation Table

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S	V
1	Cmammal	Y	pt	fd	N	Claw	Meat	Y	N	Y	2
2	Ungulate	Y	bt	sd	N	Hoof	Grass	Y	N	N	2
3	Nonfly	N	N	sd	Y	Claw	Grain	N	Y	N	1
4	Nonfly	N	N	sd	Y	Web	Fish	N	N	N	1
5	Flyingb	N	N	sd	Y	Claw	Grain	N	Y	Y	1
6	Flyingb	N	N	fd	Y	Claw	Meat	N	Y	N	1
7	Viper	N	pt	fd	N	N	Meat	N	N	N	1

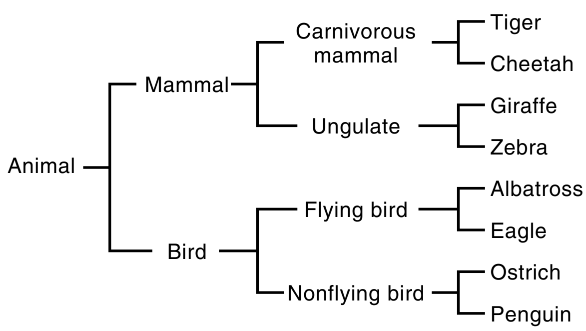


Figure 6. Concept hierarchy of the animal world.

Table 4. Generalized Relation

#	Animal	H	T	EY	F	Feet	Eat	M	Fly	S	V
1	Mammal	Y	pt	fd	N	Claw	Meat	Y	N	Y	2
2	Mammal	Y	bt	sd	N	Hoof	Grass	Y	N	N	2
3	Bird	N	N	sd	Y	Claw	Grain	N	Y	N	1
4	Bird	N	N	sd	Y	Web	Fish	N	N	N	1
5	Bird	N	N	sd	Y	Claw	Grain	N	Y	Y	1
6	Bird	N	N	fd	Y	Claw	Meat	N	Y	N	1
7	Other	N	pt	fd	N	N	Meat	N	N	N	1

Abbreviations: H: hair, F: feather, T: teeth, S: swim, pt: pointed, bt: blunted, fd: forward, sd: side.

Method:
Step 1. $R_t \leftarrow R$;
Step 2. **for** each attribute A_i ($1 \le i \le n$) or R_t **do** {
if A_i is nongeneralizable **then** remove A_i;
if A_i is not desirable but generalizable **then** generalize A_i to the desirable level;}
Step 3. $R_p \leftarrow R_t$

A prime relation is essentially an intermediate generalized relation with nondesirable attributes removed; it maintains the relationship among generalized data in different attributes for frequently sought data.

Algorithm 3: Feature Table T_A extraction for an attribute A from the generalized relation R'
Input: a generalized relation R' consists of (i) an attribute A with distinct values $a_1, \ldots, a_m$, where m is the number of distinct values for A, (ii) j other attributes $B_1, \ldots, B_j$, j is the number of attributes in the relation R' except A, (iii) a special attribute, *vote.*
Output: The Feature Table T_A
Method:
Step 1. The feature table T_A consists of $m + 1$ rows and $I + 1$ columns, where I is the total number of distinct values in all the attributes.
Step 2. Each slot in T_A (except the last row) is filled by the following:
for each row r **in** R' **do** {
for each attribute B_j **in** R' **do**
$T_A[r.A, r.B_j] \leftarrow T_A[r.A, r.B_j] + r.\text{vote}$;
$T_A[r.A, \text{vote}] \leftarrow T_A[r.A, \text{vote}] + r.\text{vote}$;}
Step 3. The last row p in T_A is filled by the following procedure:
for each column s **in** T_A **do** {
for each row t (except the last row p) **in** T_A **do**
$T_A[p, s] \leftarrow T_A[p, s] + T_A[t, s]$;

Storing prime relations for frequently sought datasets may facilitate extraction of different kinds of generalized rules. Further generalization may be performed on prime relations to derive characteristic or inheritance rules, if necessary. Based upon the domain of enquiry, different feature tables can be extracted as described in Algorithm 3. First the prime relation (Table 3) is futher generalized to the generalized relation (Table 4).

The feature table is then extracted from the generalized relation using Algorithm 3 based on the attribute “animal” and the result shown in Table 5. Different feature tables can be extracted from the generalized relation, depending upon the interest shown in different attributes. The feature table is useful for deriving relationships between the classification attribute and other high-level attributes.

We can now refine our original algorithm for attribute-oriented generalization and present Algorithm 4, which is used to discover characteristic and equality rules from a database. Other rules, for example, inheritance rules, can also be discovered using feature table extraction techniques.

Algorithm 4: Attribute oriented induction for discovering characteristic and equality rules w/a concept hierarchy.
Input: (i) the prime relation obtained by Algorithm 1, (ii) a concept hierarchy table, (iii) the threshold, N, for the total number of tuples in the final generalized relation.
Output: a set of characteristic rules and equality rules
Method:
Step 1. Generalize the prime relation further by performing attribute-oriented concept ascension
Step 2. Extract a feature table TA from the prime relation based upon a certain attribute A (Alg. 3)
Step 3. Assume that there are a total of I classes (distinct values for each attribute $A, A_1, \ldots, A_I$). Assume that there are J attributes: $C_1, \ldots, C_J$, for the data in the feature table. Use K_j to denote the number of distinct

Table 5. Feature Table for the Attribute Animal

	Hair		Teeth			Feather		Swim		
Animal	Y	N	P	B	N	Y	N	Y	N	Vote
Mammal	4	0	2	2	0	0	4	4	0	4
Bird	0	4	0	0	4	4	0	1	3	4
Others	0	1	1	0	0	0	1	0	1	1
Total	4	5	3	2	4	4	5	5	4	9

values for attribute J_j. According to the feature table, two probability values, $b_{i,j,k}$ and $c_{i,j,k}$, are associated with the k^{th} value ($k = 1, \ldots, K_j$) of the j^{th} attribute ($j = 1, \ldots, J$) in the i^{th} class ($i = 1, \ldots, I$). Notice that the number of tuples associated with the k^{th} value of the j^{th} attribute in the i^{th} class is denoted by $a_{i,j,k}$.
$b_{i,j,k} = a_{i,j,k}$/total & $c_{i,j,k} = a_{i,j,k}$/vote.
where $b_{i,j,k}$ represents the probability of $a_{i,j,k}$ in the entire database and $c_{i,j,k}$ denotes the probability of $a_{i,j,k}$ in the i^{th} class.

Step 4. Extract characteristic rules and equality rules based on the probability for each distinct value of every attribute in each class in the feature table T_A, as follows:
for each class **do** {
if $b_{i,j,k} = c_{i,j,k} = 1$ **then** the following rule is inferred $A_j = T_A[i, k, j] \leftrightarrow$ Class $= C_i$.
if $b_{i,j,k} = 1$ and $c_{i,j,k} < 1$ **then** the following rule is inferred $A_j = T_A[i, k, j] \rightarrow$ Class $= C_i$.
if $b_{i,j,k} < 1$ **and** $c_{i,j,k} = 1$ **then** include $A_j = T_A[\mathrm{i}, k, j]$ as a component for the corresponding characteristic rule for the i^{th} class.
if $b_{i,j,k} \neq 1$ **and** $c_{i,j,k} \neq 1$ **and** $b_{i,j,k} * c_{i,j,k} \leq r_{\text{frequency}}$ **then** ignore this value
else include the value as one of the characteristic values for this attribute.}

Step 5. Simplify the learned rules. [if the distinct data value set of an attribute covers the entire set of values for the attribute, remove this attribute and its associated values from the rule; etc.]

Step 6. Discover equality rules for different attributes based on the feature table. For each class C_i, for any two attributes j_1 and j_2 that relate the k_1^{th} value in the j_1^{th} attribute and the k_2^{th} value in the j_2^{th} attribute, if $a_{i,j1,k1} = a_{i,j2,k2} =$ vote, infer the following rule
$A_{j1} = T_A[i, j_1, k_1] \leftrightarrow A_{j2} = T_A[i, j_2, k_2]$.

An example is given below to illustrate this overall process utilizing Algorithm 4.

The first step consists of first further generalizing the prime relation (Table 3) given the *animal world* (Table 2) and the *concept hierarchy* for the attribute "animal"; to do so, we apply Algorithm 2 to Table 2, resulting in the *prime relation* Table 3. We then generalize Table 3 further to the *generalized relation* shown in Table 4.

In the second step, we extract the *feature table,* based on the attribute "animal," resulting in Table 5.

We examine the values in the feature table in the third step. For *Class = mammal* and *Hair = yes,* we have $a_{1,1,1} = 4$, $b_{1,1,1} = c_{1,1,1} = 1$, because *Class = mammal* appears 4 times and the total tuples for *Class = mammal* is 4. However *Hair = yes* appears only 4 times in the entire table, so a rule can be inferred *(Hair = yes) ↔ (Class = mammal).* Similarly, we obtain *(Milk = yes) ↔ (Class = mammal) and (Class = mammal) → (Feet = claw|hoof)* and *(Eats = meat|grass)* and for Class = bird *(Feather = yes) ↔ (Class = bird)* and *(Class = bird) → (Feet = claw|web)* and *(Eats = grain|fish|meat).*

In the fourth step we simplify the rules, count the number of values appearing as *characteristic* values for the attribute, and compare them with the total number of distinct values for the attribute. If the difference > threshold then the "not" operator is introduced to simplify the rule, for example, *(Hair = yes) ↔ (Class = mammal) and (Class = bird) → (Feet = claw|web)* and *(Eats = grain|fish|meat).* Since there are four distinct values {*meat, grass, grain, and fish*} for the attribute *Eats* and it takes 3 out of 4 of those values, we can use *(Eats ≠ grass)* instead of *(Eats = grain|fish|meat)* as a component of this rule. Thus *(Class = bird) → (Feet ≠ hoof) and (Eats ≠ grass).* Similarly *(Class = mammal) → not (Feet = web)* and *(Eats = meat|grass).*

We analyze the data between different attributes and find the relationship between them to infer *equality rules* in the last step, for example, for *(Hair = yes) ↔ (Feather = no), (Hair = yes) ↔ (Milk = yes),* and so on.

We now turn to some advanced techniques such as discretization, conceptual clustering, and rough sets analysis within the context of several recent advanced database mining systems.

SOME ADVANCED OPERATIONS AND ADVANCED DATABASE MINING SYSTEMS

Some more recent database mining systems amalgamate a number of novel techniques. One such amalgamation has been the marriage of rough sets theory (24,25) and database mining (26). Several interesting database mining programs have emerged from this marriage (27–31).

DB-Discover

The DB-Discover software package is useful for data access and summarization. As a data access tool, DB-Discover allows a data analyst to dynamically organize his or her data according to many different high level organizations without modifying the data itself. The analyst can then query the data, according to high-level concepts rather than according to specific data codes, even though these concepts are not present in the database. As a summarization tool, DB-Discover can generalize and summarize the data in large databases to many different levels so that useful patterns in the data become apparent. Since the database itself is not used for these operations, they can be done quickly. DB-Discover runs on a PC under OS/2 and on a Unix-based machine (IRIX and SunOS) with a graphical, X-windows interface, or with a text-based, command line interface. DB-Discover's client-server architecture allows connection to databases running on other platforms. The current DB-Discover version is Windows 95 by Microsoft.

DB-Discover consists of five components: a user-interface, a command module, a database access module, a concept hierarchy, and a learning module. DB-Discover is illustrated structurally in Fig. 7.

The user-interface of DBLEARN consisted of an interactive command line interface which implemented a superset of structured query language (SQL). Subsequently, DB-Discover incorporated a graphical user interface, which made the discovery program accessible by unskilled data miners via knowledge of the concept hierarchies rather than of the database schema.

The command module is the primary controller of communication between the DB-Discover modules. It provides one or two relations to be generalized to the learning module and provides the functions necessary to do so. The command han-

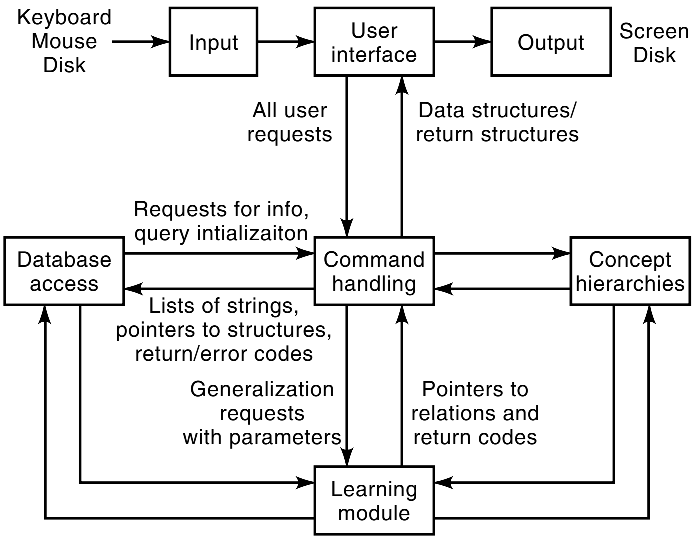


Figure 7. The architecture of DB-Discover.

dler guides the construction of the necessary query to extract desired relations and connects to the database access module to initialize the query and retrieve tuples for the learning module. The command module also directs loading of the concept hierarchies from the concept hierarchy module, provides access to them so the interface can display them, and then performs the translation from high-level concepts to low-level database attribute values.

The original DBLEARN prototype suffered from relatively poor performance, albeit serving well as an adequate proof of concept for knowledge discovery in databases. The primary causes of the performance difficulties are detailed in (32) and include: excessive storage requirements, inefficient data representations, and inefficient data retrieval. DB-Discover addressed these problems resulting in a 1000-fold speedup of process and additionally added a graphical user interface which permitted users access to discovered data via concept hierarchies, as illustrated in Fig. 8 which shows how to structure a query using DB-Discover's graphical user interface.

DB-Discover also permits the user to manipulate the retrieved task-relevant data by further generalizing (or specializing) across various attributes flexibly and easily without additional SQL queries. This can be accomplished most easily by manipulating the concept hierarchies; see Fig. 9.

Information Reduction and Attribute Reduction Using Rough Sets

Throughout this section we will make use of the information presented in Table 6 by way of illustration. Table 6 illustrates a collection of Japanese and American cars and our objective is to discover knowledge that can tell us factors that affect the gasoline mileage of a car. We partition the table into two disjoint subsets, the *condition attributes C* ("make_model," type of fuel system "fuel," engine displacement "disp," "weight," number of cylinders "cyl," "power," presence of turbocharge "turbo," compression ratio "comp," and transmission "trans") and the *decision attribute D* ("mileage").

An attribute-oriented generalization algorithm similar to DBLEARN and DB-Discover is first applied constrained by two thresholds: the *attribute* threshold and the *proportion* threshold, using the concept hierarchy shown in Table 7. If the attribute is generalizable, it should be generalized to a higher level concept. The generalized car information system illustrated in Table 8 is the result of applying Algorithm 5 to Table 6 with all thresholds set to 2 and $p = 0.84$.

Algorithm 5 differs from previous attribute oriented algorithms in the use of the ratio d_i/t_i to choose the next attribute for generalization. Rather than selecting the attributes in arbitrary order, we select the attribute that has the most values in proportion to the threshold, thus directing the algorithm to areas where the most improvement is possible.

Algorithm 5. Extracts a generalized information system from a relation (EGIS).

Input: (i) A set of task-relevant data R, a relation or arity n with a set of attributes $A_i (1 \leq i \leq n)$; (ii) a set H of concept hierarchies where each $H_i \in H$ is a hierarchy on the generalized attribute A_i, if available; (iii) t_i is a threshold for attribute A_i, and d_i is the number of dis-

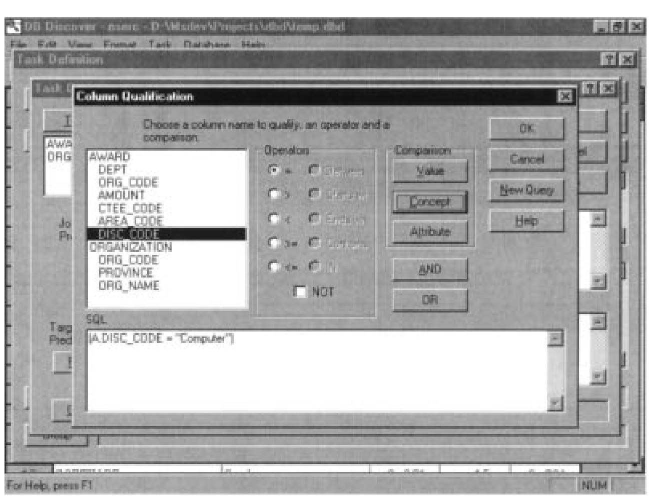


Figure 8. How to structure a query using DB-Discover.

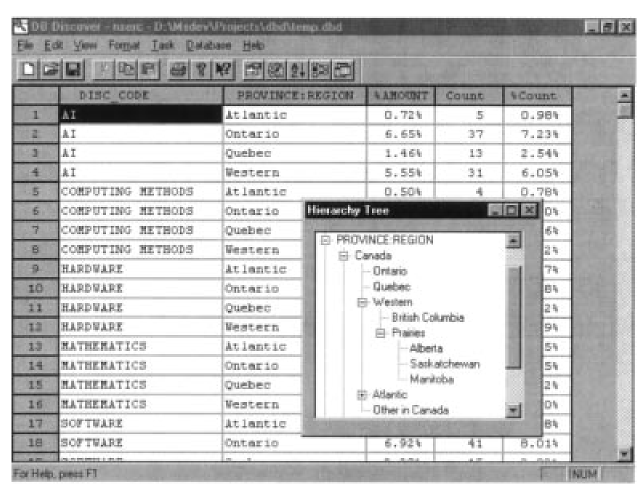


Figure 9. Further manipulating the retrieved task-relevant data using DB-Discover.

Table 6. Collection of "Cars" Information

Make_model	Fuel	Disp	Weight	Cyl	Power	Turbo	Comp	Trans	Mileage
Ford Escort	EFI	Medium	876	6	High	Yes	High	Auto	Medium
Dodge Shadow	EFI	Medium	1100	6	High	No	Medium	Manu	Medium
Ford Festiva	EFI	Medium	1589	6	High	No	High	Manu	Medium
Chevrolet Corvette	EFI	Medium	987	6	High	No	Medium	Manu	Medium
Dodge Stealth	EFI	Medium	1096	6	High	No	High	Manu	Medium
Ford Probe	EFI	Medium	867	6	High	No	Medium	Manu	Medium
Ford Mustang	EFI	Medium	1197	6	High	No	High	Manu	Medium
Dodge Daytona	EFI	Medium	798	6	High	Yes	High	Manu	High
Chrysler LeBaron	EFI	Medium	1056	4	Medium	No	Medium	Manu	Medium
Dodge Sprite	EFI	Medium	1557	6	High	No	Medium	Manu	Low
Honda Civic	2-BBL	Small	786	4	Low	No	High	Manu	High
Ford Escort	2-BBL	Small	1098	4	Low	No	High	Manu	Medium
Ford Tempo	2-BBL	Small	1187	4	Medium	No	High	Auto	Medium
Toyoto Corolla	EFI	Small	1023	4	Low	No	High	Manu	High
Mazda 323	EFI	Medium	698	4	Medium	No	Medium	Manu	High
Dodge Daytona	EFI	Medium	1123	4	Medium	No	Medium	Manu	Medium
Honda Prelude	EFI	Small	1094	4	High	Yes	High	Manu	High
Toyoto Paseo	2-BBL	Small	1023	4	Low	No	Medium	Manu	High
Chevrolet Corsica	EFI	Medium	980	4	High	Yes	Medium	Manu	Medium
Chevrolet Beretta	EFI	Medium	1600	6	High	No	Medium	Auto	Low
Chevrolet Cavalier	EFI	Medium	1002	6	High	No	Medium	Auto	Medium
Chrysler LeBaron	EFI	Medium	1098	4	High	No	Medium	Auto	Medium
Mazda 626	EFI	Small	1039	4	Medium	No	High	Manu	High
Chevrolet Corsica	EFI	Small	980	4	Medium	No	High	Manu	High
Chevrolet Lumina	EFI	Small	1000	4	Medium	No	High	Manu	High

tinct values of attribute A_i; and (iv) p defined by user is a proportional value ($0 < p \leq 1$).
Output: The generalized information system R'.
MAXTUPLES $\leftarrow p \times |R|$; $R' \leftarrow R$;
while $|R'| \geq$ MAXTUPLES and $\exists d_i > t_i$ **do**
select an attribute $A_i \in A$ such that d_i/t_i is maximal
if A_i is generalizable
then ascend tree H_i 1 level & make appropriate substitutions in R'
else remove attribute A_i from R'
endif
remove duplicates from R'; recalculate d_i for each attribute
endwhile

Table 7. Concept Hierarchy for Table 6

Attribute	Concept	Values
Make_model	Honda	Civic, Acura, . . ., Accord
	Toyota	Tercel, . . ., Camry
	Mazda	Mazda_323, Mazda_626, . . ., Mazda 939
	Japan (car)	Honda, Toyoto, . . ., Mazda
	Ford	Escort, Probe, . . ., Taurus
	Chevrolet	Corvette, Camaro, . . ., Corsica
	Dodge	Stealth, Daytona, . . ., Dynasty
	USA (car)	Ford, Dodge, . . ., Chevrolet
	any (make-model)	Japan (car), . . ., USA (car)
	Light	0, . . ., 800
	Heavy	801, . . ., 1200
	Medium	1201, . . ., 1600
	Any (weight)	Light, medium, heavy

Often it is difficult to know exactly which features are relevant and/or important for the learning task. Usually all features believed to be useful are collected into the database; hence databases normally contain some attributes that are unimportant, irrelevant, or even undesirable for a given learning task. The need to focus attention on a subset of relevant attributes is now receiving a great deal of attention in the database mining community (33,34). Pawlak (24) introduced *rough sets* theory, which provides the necessary tools to analyze a set of attributes globally. Using rough set theory, the minimal attribute set or *reduct* of the attribute in the generalized relation can be computed and each reduct can be used instead of the entire attribute set without losing any essential information. By removing these attributes, which are not in the reduct, the generalized relation can be further reduced. To reduce the generalized relation further, two fundamental concepts play an important role—the *reduct* and the *core.* Intuitively, a reduct of the generalized relation is its essential part, that part which is sufficient to define all basic concepts in the class under consideration. The core is, in a certain sense, the reduct's most important part. Reducing the generalized relation entails removal of irrelevant or superfluous attributes in such a way that the set of elementary categories in the generalized relation are preserved. This procedure enables us to eliminate all unnecessary data from the generalized relation, preserving only that part of the data which is most useful for decision-making.

Objects can be grouped to represent a certain relationship among a set of attributes C, in a generalized information system. Each relationship among the set of attributes C correspond to a classification of objects on the generalized information system into disjoint *equivalence* classes, where objects belonging to the same classification have the same attribute values for every attribute in C. An equivalence relation $U \times$

Table 8. Generalized Cars Information System

Make_model	Fuel	Disp	Weight	Cyl	Power	Turbo	Comp	Trans	Mileage
USA	EFI	Medium	Medium	6	High	Yes	High	Auto	Medium
USA	EFI	Medium	Medium	6	High	No	Medium	Manu	Medium
USA	EFI	Medium	Heavy	6	High	No	High	Manu	Medium
USA	EFI	Medium	Medium	6	High	No	High	Manu	Medium
USA	EFI	Medium	Light	6	High	Yes	High	Manu	High
USA	EFI	Medium	Medium	4	Medium	No	Medium	Manu	Medium
USA	EFI	Medium	Heavy	6	High	No	Medium	Manu	Low
Japan	2-BBL	Small	Light	4	Low	No	High	Manu	High
USA	2-BBL	Small	Medium	4	Low	No	High	Manu	Medium
USA	2-BBL	Small	Medium	4	Medium	No	High	Auto	Medium
Japan	EFI	Small	Medium	4	Low	No	High	Manu	High
Japan	EFI	Medium	Light	4	Medium	No	Medium	Manu	High
Japan	EFI	Small	Medium	4	High	Yes	High	Manu	High
Japan	2-BBL	Small	Medium	4	Low	No	Medium	Manu	High
USA	EFI	Medium	Medium	4	High	Yes	Medium	Manu	Medium
USA	EFI	Medium	Heavy	6	High	No	Medium	Auto	Low
USA	EFI	Medium	Medium	6	High	No	Medium	Auto	Medium
USA	EFI	Medium	Medium	4	High	No	Medium	Auto	Medium
Japan	EFI	Small	Medium	4	Medium	No	High	Manu	High
USA	EFI	Small	Medium	4	Medium	No	High	Manu	High

$U \supseteq R(C)$ represents the classification corresponding to the set of attributes in C. Pawlak (25) calls the pair $AS = (U, R(C))$ an *approximation space.*

Before discussing attribute reduction, it is informative first to perform a dependency analysis of attributes. Let $R^*(C) = \{X_1, X_2, \ldots, X_n\}$ be the collection of equivalence classes of the relation $R(C)$, where an element X_i is a group of objects having the same values for all attributes in C, and let $R^*(D) = \{Y_1, Y_2, \ldots, Y_m\}$ be a collection of equivalence classes of the relation $R(D)$, where each element is a group of objects having the same values for all attributes in D and creates a concept class on the universe U. The lower approximation in the approximation space AS, denoted as $LOW(C, D)$ is defined as the union of those equivalent classes of the relation $R(C)$ that are completely contained by one of the equivalence classes of relation $R(D)$, that is,

$$LOW(C, D) = \cup_{YiR^*(D)} \{X \in R^*(C) : Y_i \supseteq X\}$$

The upper approximation in the approximation space AS, denoted as $UPP(C, D)$, is defined as the union of those equivalence classes of $R(C)$ which are partially contained by one of the equivalence classes of $R(D)$, that is,

$$UPP(C, D) = \cup_{YiR^*(D)} \{X \in R^*(C) : Y_i \cap X \neq 0\}$$

The lower approximation $LOW(C, D)$ characterizes objects that can be classified into one of the concepts without any *uncertainty* based only on the classification information (35). The upper approximation $UPP(C, D)$ is a set of objects which can *possibly* be classified into one of the concepts with some ambiguous measurements. By definition,

$$U \supseteq UPP(C, D) \supseteq LOW(C, D)$$

The degree of dependency $K(C, D)$ in the relationship between the groups of attributes C and D can be defined as

$$K(C, D) = card\ (LOW(C, D))/card(U)$$

where *card* yields set cardinality. The dependency between two sets of attributes C and D indicates the extent to which values of attributes in D depend on values of attributes in C. By definition, $0 \le K(C, D) \le 1$ because $U \supseteq LOW(C, D)$. If $K(C, D)$ is equal to 1, the dependency is considered to be fully functional. $K(C, D)$ is equal to 0 when none of the values of attributes in D can be uniquely determined from the values of attributes in C.

In actual applications, databases usually contain incomplete and ambiguous information. The original rough sets technique does not use information in the boundary area $UPP(C, D) - LOW(C, D)$ of an approximation space AS. In some situations, this leads to information loss and the inability to take advantage of statistical information. Extensions to rough sets theory to rectify this situation can be found in (36,37). Essentially these extensions draw some elementary sets belonging to the boundary area into the lower approximation; we can easily modify our approach by changing slightly the computation of the degree of dependency. The decision rules obtained in this fashion are characterized by an uncertainty factor which is, in fact, probabilistic that an object matching the condition part of the rule belongs to the concept.

We say that an attribute $a \in C$ is *superfluous* in C, with respect to D if $K(C, D) = K(C\text{-}\{a\}, D)$; otherwise a is *indispensable* in C, with respect to D. If we remove an indispensable attribute, we decrease the degree of dependency, that is, $K(C\text{-}\{a\}, D) < K(C, D)$, if a is indispensable. Furthermore, we call a subset B of a set of attributes C a *reduct* of C, with respect to D, if and only if: (1) $K(B, C) = K(C, D)$; and (2) $K(B, D) \neq K(B, \neg\{a\}, D)$, for any $a \in B$. A reduct is a minimal sufficient subset of a set of attributes which preserves the degree of dependency with respect to another set and which has the same ability to discern concepts as when the full set of attributes is used (24). The first condition ensures that the reduct preserves the degree of dependency with respect to D and the second condition ensures that the reduct is a minimal subset and that any further removal will change the degree of dependency.

A given information system can have more than one reduct and each reduct can be used to represent the original infor-

Table 9. Significance Values

Attribute Name	χ^2
Weight	17.54
Make_model	12.86
Disp	7.08
Cyl	5.94
Power	5.68
Tran	4.53
Comp	3.84
Fuel	0.63
Turbo	0.63

mation system. In (38) they computed all reducts for small information systems and then chose one to use. Unfortunately, finding all reducts of an information system is NP-hard (39) and, for many applications such as ours, is also unnecessary. We are interested in finding one "good" reduct. Table 9 illustrates the significance values for the attributes in Table 6. Higher significance value for an attribute indicates greater interaction with decision attributes in D. The computation of a "good" reduct depends on the optimality criterion associated with attributes. Alternatively/additionally, we can assign significance values to attributes and base the selection of those values. The chi-square statistic, traditionally used to measure the association between two attributes in a contingency table, compares the observed frequencies with the frequencies that one would expect if there were no association between the attributes (40).

The following greedy algorithm, Algorithm 6, constructs a reduct for a generalized information system U.

Algorithm 6. Computes a reduct (GENRED).

Input: (i) A generalized information system U; (ii) a set of attributes C over the information system U; and (iii) the degree of dependency $K(C, D)$ in the information system U;
Output: A reduct, that is, a set of attributes SM.
Compute the significance value for each attribute $a \in C$;
Sort the set of attributes C based on significance values;
$SM \leftarrow 0$;
while $K(SM, D) \neq K(C, D)$
  do /*create subset SM of attr's C by adding attr's */
  select an attr a with the highest significance value in C; $SM \leftarrow a \cup SM$;
  compute degree of dependency $K(SM, D)$ in the information system U
endwhile
$N \leftarrow |SM|$;
for $i = 0$ to $N - 1$
  do /*create a reduct of attr's SM by dropping condition attr's */
  remove the i^{th} attribute a_i from the set SM;
  compute the degree of dependency $K(SM, D)$ in the information system U
  if $K(SM, D) \neq K(C, D)$ **then** $SM \leftarrow SM \cup a_i$
  endif
endfor

Algorithm 6 assigns a significance value based on an evaluation function to each attribute and sorts the attributes based on their significance values. A forward selection method is then employed to create a smaller subset of attributes with the same discriminating power as the original attributes. At the end of this phase, the attribute set SM contains the "good" performing attribute subset found thus far. Finally, to compute a reduct, a backward elimination method removes attributes, one by one, from the set SM. The lower the significance value is, the earlier the attribute is processed. The degree of dependency is calculated at each step based on the remaining attributes in SM; if the degree of dependency is changed the attribute is restored to the set SM, otherwise it is permanently removed. Attributes remaining in the set SM for the reduct, other attributes may be removed. Table 10 illustrates a reduct for the generalized car information system presented in Table 8. The forward selection process collects the attributes with higher significance values one by one. For Table 8 this process stops with the collected set SM = {weight, make_model, disp, cyl, power, tran, comp}, which has the same degree of dependency as the original set. The backward elimination step deletes redundant attributes from SM, resulting in the set SM = {weight, make_model, power, tran, comp} as a reduct from which further deletion would reduce the degree of dependency.

For n objects (tuples) with a attributes, the time complexity of our algorithm is $O(an + a \log a)$ in the worst case, because computing the degree of dependency using a hashing technique is $O(n)$, computing attribute significance values is $O(an)$, sorting the attributes based on significance values is $O(a \log a)$, creating the smaller subset of attributes using a hash technique is $O(an)$, and creating the reduct is $O(an)$.

Before introducing GRG, we first introduce an earlier version entitled DBROUGH, which inspired many of the ideas from this discussion.

Rough Sets Approach to Attribute-Oriented Generalization: DBROUGH

DBROUGH is a direct descendant of DBLEARN; its architecture is shown in Fig. 10. The system takes SQL-like database

Table 10. Reduct of the Generalized Car Information System (Table 8)

Make_model	Weight	Power	Comp	Tran	Mileage
USA	Medium	High	High	Auto	Medium
USA	Medium	High	Medium	Manu	Medium
USA	Heavy	High	High	Manu	Medium
USA	Medium	High	High	Manu	Medium
USA	Light	High	High	Manu	High
USA	Medium	Medium	Medium	Manu	Medium
USA	Heavy	High	Medium	Manu	Low
Japan	Light	Low	High	Manu	High
USA	Medium	Low	High	Manu	Medium
USA	Medium	Medium	High	Auto	Medium
Japan	Medium	Low	High	Manu	High
Japan	Light	Medium	Medium	Manu	High
Japan	Medium	High	High	Manu	High
Japan	Medium	Low	Medium	Manu	High
USA	Heavy	High	Medium	Auto	Low
USA	Medium	High	Medium	Auto	Medium
Japan	Medium	Medium	High	Manu	High
USA	Medium	Medium	High	Manu	High

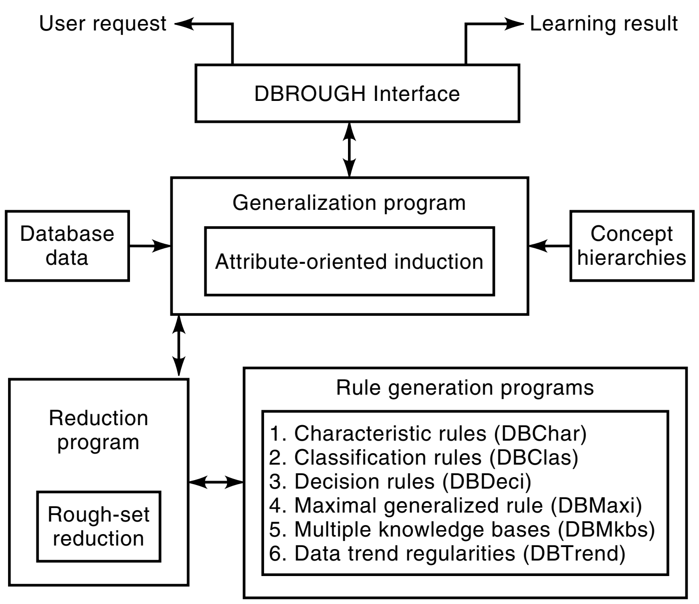


Figure 10. The architecture of DBROUGH.

learning requests and applies different algorithms to discover rules. Again background knowledge is stored in concept hierarchies, which, in this case, can be adjusted dynamically according to database statistics and specific learning requests.

DBROUGH can execute the following procedures to produce results:

- *DBChar:* find the characteristic rule for the target class;
- *DBClass:* find the characteristic rules of the target class with other classes;
- *DBDeci:* find the decision rules for the decision attributes;
- *DBMaxi:* find all the maximal generalized rules or the best *k* maximal generalized rules;
- *DBTrend:* find the data trend regularities for the target class; and
- *DBMkbs:* find different knowledge bases for the target class.

Perhaps the best way to illustrate DBROUGH is by example as well. Details are provided in (41) on system operation, including the syntax of its extended SQL language. Our example illustrates use of the procedure DBChar; specification of the learning task to DBROUGH is as follows:

```
learn characteristic rule for ''CS_Op_Grants''
  from Award A, Organization O, grant_type G
  where O.org_code = A.org_code and
    G.Grant_order = ''Operating''
  and A.grant_code = G.grant_code and
    A.disc_code = ''Computer''
  in relevance to amount, province,
    prop(votes)*, prop(amount)
    using table threshold 18
    using hierarchy disc, amount, prov,
      grant_type go
```

The results returned from DBROUGH are almost identical to those shown earlier in response to a similar request of DBLEARN, as expected. Another example illustrates the diversity of DBROUGH:

```
learn discrimination rule for
  ''Ontario_CS_Grants''
  where O.province = ''Ontario''
  in contrast to ''Newfoundland_CS_Grants''
    where O.province = ''Newfoundland''
    from award A, organization O, grant_type G
      where A.grant_code = G.grant_code and
        A.org_code = O.org_code and A.disc_code
        = ''Computer''
      in relevance to disc_code, amount,
        grant_order go
```

Notice that both attribute and table threshold values are defaulted. All of the concept hierarchy information required is stored in a default file *concept.* The classification rule for “Ont_Grants” versus “Newfoundland_Grants” is:

```
∀x Ont_Grants(x) ←
  {disc_code = ''Computer'' & grant_order =
    ''Operating'' & amount = (''20-40K, 40-
    60K'')} [34.4%] |
  {disc_code = ''Computer'' & grant_order =
    ''Other'' & amount = (''40K- , 40-60K'')}
    [4.74%] |
  {disc_code = ''Computer'' & grant_order =
    ''Strategic, Operating'' & amount = (''40K-
    '')} [5.53%] |
  {disc_code = ''Computer'' & grant_order =
    ''Strategic'' & amount = (''40-60K'')}
    [0.004%]
```

The final reduced relation is illustrated in Table 11.

DBROUGH is the first system to apply attribute oriented generalization to remove undesirable attributes and generalize the primitive data to a desirable level, much like the DBLEARN family, and then perform a data-reduction process based on rough set theory to compute the minimal attribute set (reduct) for use in further reducing the generalized relation. Although the realization of a general purpose, fully automated knowledge discovery system is still in the future, DBROUGH and its successor, GRG (42,43) (still under development), are promising to lead us to such a realization.

Induction of Decision Rules: GRG

Decision rules preserve logical properties of data. They are easy to understand. Decision rules are a common way to rep-

Table 11. Final Reduced Relation

Disc_code	Grant_order	Amount	Votes
Computer	Operating_grants	20–40K	62
Computer	Operating_grants	40–60K	25
Computer	Other	60K–	7
Computer	Other	40–60K	5
Computer	Strategic_grants	60K–	8
Computer	Operating_grants	60K–	6
Computer	Strategic_grants	40–60K	1

resent knowledge in rue-based expert systems and have become popular in inductive learning system. A *rule* is a combination of values of some condition attributes such that the set of all objects matching it is contained in the set of objects labeled with the same class (and such that there exists at least one such object). A rule r is denoted as an implication

$$r: (a_{i1} = V_{i1}) \,\&\, (a_{i2} = V_{i2}) \,\&\, \ldots \,\&\, (a_{in} = V_{in}) \rightarrow (d = V_d)$$

where a_{i1}, a_{i2}, . . ., and a_{in} are the condition attributes and d is the decision attribute. The set of attribute-value pairs occurring on the left-hand side of the rule r is referred to as the *condition part,* denoted $cond(r)$, and the right-hand side is the *decision part,* $dec(r)$, so that the rule can be expressed as $cond(r) \rightarrow dec(r)$. Including more condition attributes in $cond(r)$ makes the rule more specific. Decision rules obtained directly from the reduced relation (information system) are the specific rules that only match one equivalence class. These rules can be generalized by removing one or several conditions from the condition part.

Our aim is to produce rules in the learning process that are maximally general rules by removing the maximum number of condition attributes values without decreasing classification accuracy of the rule. Computing such rules in especially important in data mining applications, since they represent the most general patterns existing in the data. A reduced information system can be considered as a set of specific decision rules, each rule of which corresponds to an equivalence class of $R^*(RED)$, which is the set of equivalence classes generated by the subset of $C \supseteq RED$ of condition attributes C, where the subset RED is a reduct of C. Before describing our rule generation algorithm, Algorithm 7, which computes a set of maximally generalized rules, we introduce two propositions: *rule redundancy* and *rule inconsistency.*

Rule redundancy:

1. If r_i and r_j are valid rules where $cond(r_i) = cond(r_j)$ and $dec(r_i) = dec(r_j)$, then r_i and r_j are *logically equivalent rules.*
2. If r_i and r_j are valid rules where $cond(r_j) \supset cond(r_i)$ and $dec(r_i) = dec(r_j)$, then r_j is *logically included* in r_i.

Rule inconsistency:

1. If r_i and r_j are valid rules where $cond(r_j) \supseteq cond(r_i)$ and $dec(r_i) \neq dec(r_j)$, then r_i and r_j are *decision inconsistent.*

Algorithm 7. Computes a set of maximally generalized rules (GENRULES).

Input: A non-empty set of specific decision rule *RULE*
Output: A non-empty set of maximally general rule *MRULE*
MRULE ← 0; $N \leftarrow |RULE|$ /* N is the number of rules in *RULE* */
for $i = 0$ to $N - 1$ **do**
  $r \leftarrow r_i$
  $M \leftarrow |r|$ /* M is the number of condition attributes in rule r */
  compute the significance value SIG for each condition of the rule r
  sort the set of conditions of the rule based on the significance values
  for $j = 0$ to $M - 1$ **do**
    remove the j^{th} condition attribute a_i in rule r
    if r inconsistent with any rule $r_n \in RULE$ **then**
      restore the dropping condition a_j
    endif
  endfor
  remove any rule $r' \in MRULE$ that is logically included in the rule r
  if rule r is not logically included in a rule $r' \in MRULE$ **then**
  $MRULE \leftarrow r \cup MRULE$
  endif
endfor

To obtain a set of maximally general rules, Algorithm 7 tells us to consider each rule in the set of specific decision rules for dropping conditions until we are left with a set of maximally general rules. The order in which we process the attributes determines which maximally general rule is generated. Thus a maximally general rule may not turn out to be the best with respect to the conciseness or the coverage of the rule. Given a rule with m conditions, we could evaluate all $2^m - 1$ possible subsets of conditions on the database and select the best rule but this is, in general, impractical.

For a near optimal solution, each condition of the rule is assigned a significance value by an evaluation function before the dropping conditions process is started. The significance value indicates the relevance of this condition for this particular case. Higher significance values indicate more relevance. The process of dropping conditions should first drop the conditions with lower significance values, as described in (44). Their evaluation function for a condition c_i of a rule is defined as

$$\mathrm{SIG}(c_i) = \mathrm{P}(c_i)(\mathrm{P}(D|c_i) - \mathrm{P}(D))$$

where $\mathrm{P}(c_i)$ is the probability of occurrence of the condition c_i or the proportion of objects in the universe matching to this condition; $(\mathrm{P}(D|c_i)$ is the conditional probability of the occurrence of the concept D conditioned on the occurrence of the condition c_i; $\mathrm{P}(D)$ is the proportion of the concept D in the database. For example, the specific rule (the seventh entry in Table 10) can be translated as

1. **if** (make_model = USA) & (weight = heavy) & (power = high) & (comp = medium) & (tran = manu)
2. **then** (mileage = low)

By definition we have

1. SIG(tran = manu) = −0.03
2. SIG(make_model = USA) = 0.04
3. SIG(power = high) = 0.06
4. SIG(comp = medium) = 0.07
5. SIG(weight = heavy) = 0.093

Thus we drop conditions of the rule in the sequence given above. No inconsistency results from dropping the first three conditions. After dropping the fourth condition “comp,” the new rule "if (weight = heavy) then (mileage = low)" is inconsistent with the specific rule derived from the third entry in Table 10, thus the condition "comp" is replaced. The fifth condition "weight" also cannot be dropped because of inconsistency. Thus the maximally generalized rule for the specific rule derived from the seventh entry in Table 10 is

if (weight = heavy) & (comp = medium) then (mileage = low)

Suppose there are n' tuples (decision rules) with a' attributes in the reduced information system. The computation of significance values of one rule requires computation $O(a'n')$ and the process of dropping conditions on one rule requires $O(a'n')$. Thus finding a maximally general rule for one decision rule requires $O(2a'n')$ time and finding maximally general rules for n' decision rules requires $O(2a'n'^2)$ time. Eliminating redundant rules requires $O(n'^2)$ time and the complexity of Algorithm 6 is $O((2a' + 1)n'^2) = O(a'n'^2)$.

Table 12 shows the set of maximally general rules corresponding to the values in Table 10 where "—" indicates "don't care." Rules in Table 10 are more concise than the original data in Table 6 and they provide information at a more abstract level. Nevertheless they are guaranteed to give decisions about mileage consistent with the original data. The column "supp" is the number of tuples in the original database that support the generalized rule. This measure provides confidence because, if the tuples in the original database distribute evenly over all possible discrete values for an attribute, then it is impossible to obtain a meaningful set of rules. Higher values for "supp" indicate greater confirmation of the rule.

This article could have been written in a variety of different ways stressing a variety of different approaches and techniques. Particular language choices may have biased other terminology; examples may have been more or less illustrative. Database mining is relatively new but the theory and foundations of many of the techniques underlying database mining are not so new. Thus, the research community has not taken long to realize and take advantage by quickly filling in details, amalgamating relevant results from a wide variety of related research paradigms, especially machine learning research.

The deliberate choice of introducing database mining in increasingly greater detail repeatedly, followed by examples to match, drawn from actual advanced prototype systems, is for reader edification. At this time, we deviate from our main discussion to present a sampling of commercial software offerings currently available, preceding some speculations about directions or considerations for future database mining endeavours.

SELECTION OF COMMERCIALLY AVAILABLE DATABASE MINING SYSTEMS

By the time you read this section, it may be obsolete. New database mining and associated products are entering the market daily. This section is intended to identify a small number of representative "database mining" products that are commercially available in 1997. The descriptions of a company's software are an edited version of their World Wide Web site information.

Angoss International—KnowledgeSEEKER

Angoss International is the developer of KnowledgeSEEKER, the leading data mining software, and of SmartWare, a complete cross-platform product suite. With KnowledgeSEEKER, Angoss provides its customers with one of the most important business software tools today. KnowledgeSEEKER benefits from a simple, intuitive GUI that is very easy to learn. It is faster and easier to use and interpret than both traditional statistical models and new technologies such as neural networks. Analysis results are rapidly displayed in the form of a clear and interactive decision tree. In just a few minutes it examines all the relationships between the fields in your data, eliminating trial-and-error guesswork by homing in on strong statistical relationships. All fields and combinations of field values within your data are examined. Those best describing your specific decision-making problem are ranked in order of importance. Both the sensitivity of the correlation finding and the volume of the information displayed are easily user-defined. KnowledgeSEEKER runs on many operating systems and configurations.

Data Distilleries B.V.—Data Surveyor

Data Surveyor is a client/server system, consisting of the following. Data Surveyor CLIENT, a graphical user interface, allows the user to formulate mining questions, inspect data and results, and interactively guide the mining process. Data Surveyor INTERNET REPORTING FACILITIES, an extensive intranet, is geared for data mining with Data Surveyor. This intranet contains: background information on data mining; application areas; all Data Surveyor documentation on line; visual objects like 3-D maps included in-line in the reports; concise management summaries; and extensive reports

Table 12. Set of Maximally General Rules

Make_model	Weight	Power	Comp	Tran	Mileage	Supp
—	Heavy	—	Medium	—	Low	2
USA	Medium	High	—	—	Medium	9
USA	Medium	—	Medium	—	Medium	8
—	Medium	—	—	Auto	Medium	4
USA	—	Light	—	—	Medium	1
—	Heavy	—	High	—	Medium	1
—	—	Medium	High	Manu	High	3
Japan	—	—	—	—	High	6
—	Light	—	—	—	High	3

automatically generated by Data Surveyor. Key features include a *user friendly interface* (a graphical interface), *easy to interpret results* (by end users, such as marketers and actuaries), *interactive data mining* (allowing you to guide the discovery process by using your own expert knowledge), and 3-D visualization.

Information Discovery, Inc.—IDIS: The Information Discovery System

IDIS: the Information Discovery System(R) is a data mining program that uncovers knowledge in large databases, automatically looking at data patterns to find unexpected influence factors. IDIS decides what to look at, generates hypotheses, discovers hidden and unexpected patterns, rules of knowledge, graphs, and anomalities that are waiting to be uncovered. The results are displayed within an easy-to-use hypermedia environment. Moreover, this data mining software generates fully readable English text reports that tell the untold story of your database. You do not need to know a query language to use IDIS. Just specify your database name and say, "Go find something interesting and useful for me!" IDIS is the leading data mining software. It has found more rules, in more databases, in more application areas than any other program ever!

IBM—IBM Intelligent Miner

IBM Intelligent Miner is a heavy-duty data miner that enables users to identify hidden data correlations by performing predictive modeling, database segmentation, link analysis, and deviation detection using a variety of data mining techniques. Yet Intelligent Miner is easy to use because it's designed for decision makers as well as data analysts. The IBM Intelligent Miner tool kit consists of powerful algorithms and processing techniques that enable application developers to analyze data stored in databases. Through deviation detection, for example, a financial services company could quickly and easily detect possible fraudulent usages of credit cards by examining deviations in the credit-card usage patterns of its customers. Using predictive modeling, a retailer could forecast changes in customer buying patterns and keep abreast of comparisons of purchases over the Internet or through mail-order with those through in-store buying. Through association discovery, a supermarket chain could determine which products are most frequently sold in conjunction with other products, and stock these store items on shelves accordingly, to maximize sales opportunities. An insurance company could use customer segmentation data to create target-marketing campaigns, or to cross-sell services among existing customers. Sequential Pattern analyses could help medical researchers identify common symptoms leading to particular illnesses.

IBM has also developed three customizable, cross-industry data mining applications. These applications include: *Customer Segmentation*—an effort to better understand customer behavior used for target marketing, cross-selling, customers retention, propensity to purchase, and consumer vulnerability analysis; *Item Set Analysis* (market basket analysis)—aims to understand customer buying behavior, and to predict their future behavior by identifying affinities among their choice of products and services; and *Fraud Detection*—identifies deviations from established usage norms in order to flag suspicious transactions, which may be indicative of fraudulent activity.

NeoVista Solutions, Inc.—The NeoVista Decision Series

NeoVista specializes in advanced pattern recognition techniques, having deployed highly parallel computing solutions in some of the most demanding, defense-related environments in the world, where accuracy and speed are essential. Now, NeoVista's Decision Series software suite brings the same advanced technology to commercial data mining, allowing Global 2000 organizations to better understand the patterns of their business. The NeoVista Decision Series allows the construction and deployment of knowledge discovery solutions for targeted business applications. These solutions can be deployed on scalable, parallel platforms that are available from accepted, standard hardware providers and operate against data resident in popular databases or in legacy systems. The data mining solutions from NeoVista thereby augment existing decision support environments by integrating with installed, standards based systems.

The Decision Series suite consists of a set of advanced, easy to use, knowledge discovery tools that interface with a data access and transformation fabric called DecisionAccess. DecisionAccess performs automatic translation of data between relational databases and pattern discovery tools, and takes care of the all-important sampling, conditioning, and encoding of data.

The pattern discovery tools of the Decision Series suite are: *DecisionNet,* an advanced neural network tool that learns to recognize patterns from training examples; *DecisionCL,* used to find groups of items that are similar (the groups can be fixed in advance, supervised clustering, or determined by the system, unsupervised clustering); *DecisionGA,* a genetic algorithm used to breed potential cases based on a loosely constructed model where the cases can converge to the best example based on a wide variety of measures; and *DecisionAR,* an association rule system used to determine the likelihood that events will occur together at one instant in time, or that they will follow each other in a logical progression over time.

Pilot Software—Pilot Discovery Server

Pilot Software, a subsidiary of Cognizant Corporation (NYSE: CZT), develops and markets decision support software designed to improve business knowledge through the flexible analysis of market and customer data. More than 100,000 users in industries such as retail, financial services, packaged goods, telecommunications, and healthcare have rapidly deployed Pilot Software's OLAP, data mining, and Web publishing products.

Pilot Discovery Server is the industry's first data mining product designed for sales and marketing professionals. Using vital customer metrics such as profitability, life-time value, or new product return on investment, Pilot Discovery Server drives a focused market segmentation and proactive analysis of customer behavior or evaluates the profitability of future marketing efforts. Working directly with and residing in a relational data warehouse, Pilot Discovery Server issues standard SQL queries for database analysis. This unique patent-pending relational database integration delivers highly visual and easily understood, explainable results.

Thinking Machines Corporation—Darwin®

Thinking Machines Corporation is the leading provider of knowledge discovery software and services. Darwin®, TMC's high-end data mining software suites enables users to extract meaningful information from large databases—information that reveals hidden patterns, trends, and correlations—and allows them to make predictions that solve business problems. Darwin's® power of prediction enables businesses to: increase return on investment; expand market share; improve the effectiveness and efficiency of marketing programs; and maximize the quality of their customer service. In short, companies that employ Darwin® data mining enjoy a clear competitive advantage over those that do not.

WHAT TO EXPECT IN THE FUTURE

Database mining is quickly gaining acceptability and marketability. The Gartner Group estimates that the use of database mining in marketing applications will increase from less than 5 percent to more than 80 percent in ten years. The META Group estimates that the database mining market will grow from $300 million in 1997 to $800 million by the year 2000.

When database mining functionality is seamlessly integrated with OLAP and existing data warehouse and business intelligence software, then the real promise of database mining will be realized. Database mining will then transcend its enabling status in favor of engaging in actual business solutions. Thus organizations will come to understand their core business much more and make business decisions based on a better understanding as a result of database mining analysis.

The core technology for database mining already exists and has been described in previous sections. What is new is the systematic application of these techniques to the databases and data warehouses that many organizations have built in recent years.

What Is a Database (Information Store)?

Databases have been with us for a long time. Many refer to databases and do not mean what we traditionally think of as databases, for example, Environment Canada has over 1000 "databases" at their disposal, only 20 percent of which are in electronic form. Databases are collections. Since the advent of the computer, the early databases were created using a variety of structures, leading to network database models, codasyl database models, and so on. Relational databases have been around since the late 1960s, but only began to be heavily employed in business, industry, and organizations in the 1980s. We know a great deal about how to build relational databases, how to optimize queries to take advantage of their structure, how to represent information using the relational model, and so forth. We can store increasingly large amounts of data in these repositories and we do.

We have built interesting products using databases, from English interfaces (natural language access to databases) which permit a person to pose an ad hoc query to a database to database mining interfaces which permit a person to uncover hidden information implicit in the database.

In the last decade or so new systems have tempted database designers with alternative models for data representation, storage, and retrieval. The terms used by these designers read like a list of buzzwords no designer would be without, for example, functional, applicative, object-oriented, distributed, concurrent, parallel, sequential, inference, heuristic, associative, procedural, connectionist, declarative, nonmonotonic, holographic, and the like. The machines such systems operate on are considered hardware, software, firmware, wetware, neural-ware, and so forth. Where will all of this lead?

Naturally, I would like to think that it is the Internet, a nonarchival (for the present time) source of unstructured data, that holds the key to future development. Database mining require three primitives to operate attribute oriented generalization effectively. Gathering task-relevant knowledge in the Internet environment is a huge challenge, but the development of generic, adaptable conceptual hierarchies will not be far behind. The future should prove to be exciting for database miners, but it will not be an easy future.

Machine Learning and Hybrid Architectures: Helpful?

Research into machine learning (ML) has evolved rapidly over the past two decades. ML researchers have embraced a variety of machine learning techniques in their efforts to improve the quality of learning programs. The relatively recent development of hybrid representations for ML systems has resulted in several interesting approaches, which combine rule induction (RI) methods with case-based reasoning (CBR) techniques to engender performance improvements over more traditional one-representation architectures.

RI systems learn general domain-specific knowledge from a set of training data and represent the knowledge in comprehensible form as IF-THEN rules. RI systems also often succeed in identifying small sets of highly predictive features, and make effective use of statistical measures to eliminate data noise. Example RI systems include C4.5 (45), AQ15 (46), and CN2 (47); Clark (48) provides an overview of RI techniques and strategies for noise abatement. Despite their successes, RI systems do not represent continuous functions well.

CBR is used in learning and problem-solving systems to solve new problems by recalling and reusing specific knowledge obtained from past experience. Common retrieval schemes are variations of the *nearest neighbor* method, in which similarity metrics are used to identify cases *nearest* to the current case. An overview of the foundational issues related to CBR is presented in (16). CBR can learn nonlinearly separable categories of continuous functions and CBR is incremental by nature, unlike most inductive learning methods, which have difficulty extending or refining their rule set during the problem-solving stage. CBR, however, does have limitations: it does not yield concise representations of concepts that can be understood easily by humans and CBR systems are usually sensitive to noise.

The complementary properties of CBR techniques and RI techniques can be advantageously combined to solve some problems to which only one technique fails to provide a satisfactory solution. Generally the combination involves CBR systems using rule-based reasoning for support. CBR systems can also be used in a support role or integrated with rule-based reasoning in some balanced fashion.

CBR processing can be augmented with rule-based techniques when general domain knowledge is needed. For example, adaptation tasks in the CBR processing cycle are usually performed by rule-based systems, where the rules capture a theory of case adaptation and the necessary aspect of the domain theory to carry out the changes (49). CASEY (50) is one such system where case adaptation is performed by rule-based reasoning in which solutions to new problems built from old solutions use the condition-part to index differences and a transformational operator at the action part of the rule. Rules can also be used to guide the search-and-matching processes in retrieval tasks of a CBR system. Rules regarding the problem domain may serve to organize the case base and, when applied, focus the search space to more relevant cases. Rules may also be used in similarity assessment by determining weights for attributes. INRECA (51) serves as an example in which a decision tree is built on the database of cases, weights of the attributes, with respect to the subclasses discovered in the three, are computed, and class-specific similarity functions are defined based on these weights. Rule-based reasoning can help in case retrieval by justifying a candidate set of cases as plausible matches. For example, knowledge-based pattern matching (rule-based reasoning) is used in PROTOS (52), to confirm expectations about a new case.

CBR can also serve a supporting role. Unlike rules, cases in a case base contain specific knowledge about a domain. When general domain knowledge is not accessible, the specific knowledge inherent in cases can provide valuable information to solve problems. Because CBR can elicit domain knowledge through its analysis of cases, CBR can aid systems with tasks where general domain knowledge is not available but needed.

Several RI systems have employed CBR to make use of the information inherent in training cases support their induction process. CABARET (53) uses CBR to aid a cooperating inductive decision-tree based learning algorithm with training set selection, branching feature selection, deliberate bias selection, and specification of inductive policy. CBR is used to form categories of a training set which include *most-on-point* cases, *best* cases, *near miss* cases, *trumping* cases, and *conflict* cases. These case taxonomies allow the learning system to consider the various roles cases play, in addition to classification, say, as positive or negative examples. For feature selection, CABARET takes advantage of CBR-provided domain knowledge as well as information-theoretic methods to select branching attributes for growing decision trees. RISE (54) induces rules in a specific-to-general fashion, starting with a rule set that is the training set of examples. RISE examines each rule in turn, uses CBR to find the nearest example of the same class that it does not already cover, and attempts to minimally generalize the rule to cover the class.

More balanced combination techniques use CBR and rule-based techniques to support each other in a learning and problem-solving environment, neither of which is in a purely support role. Example systems include INRECA (51), and a hybrid system by Golding et al. (55). INRECA performs classification by first generating and trying a decision tree, generated from the case base, to navigate the search for a matched concept or a similar case. The generalized knowledge is also used to improve retrieval by determining attribute weights (degree of attribute importance for similarity case measurement) with respect to the subclasses discovered by the decision tree. If INRECA can answer a given query at this point, no further action is required, otherwise their hybrid approach applies CBR when the query lies outside the region of the induced concept.

Golding et al.'s system focuses on hybrid representations of a concept. A concept is represented by two parts: a generalized abstract description in the form of rules and a set of exceptions in the form of exemplars. Since rules represent broad domain trends and cases usefully "fill in" rule exceptions, a hybrid approach is supported. Both rules and exemplars are used to match the new case during problem-solving. Golding et al.'s system applies rules to the target problem to approximate the answer. However, if the problem is judged to be compellingly similar to a known exception to the rules in any aspect of its behavior, then the aspect is modified after the exception rather than the rule.

INRECA's advantage lies in its incremental learning of decision trees. Over time, more and more generalized concepts can be induced based on the increasing case base. Thus INRECA evolves from a more or less pure CBR system to a system based on inductively learned knowledge. INRECA does not address uncertainty, that is, when a new case is in the boundary region of two or more concepts and thus is covered by rules that belong to different concepts.

Advantages of CBR techniques (incremental learning computational cost is small, nonlinear separable categories and continuous functions can be learned, etc.) are offset by limitations of CBR (concise representations of easily reasoned with and easily understood concepts remain elusive, high noise sensitivity remains, etc.). RI systems, symbolic in nature, have not succeeded in representing continuous functions, except by transforming the domain of a continuous decision variable into numeric ranges. However RI systems often succeed in identifying small sets of highly predictive features and make effective use of statistical measures to combat "noise."

An (56) proposes a new hybrid method which integrates RI and CBR techniques. The ELEM2-CBR employs *relevance weighting* to access similarities between cases, making use of RI results to assign weights to each attribute-value pair of the query case. Cases in the case-base can then be ranked according to their probability of relevance to the new case. ELEM2-CBR performs classification and numeric prediction under a mixed paradigm of rule-based and case-based reasoning. After performing RI, induced rules are applied in case retrieval to determine weight settings for features and to detect noise in the training set for removal before CBR is conducted. During classification, rules are applied to make decisions; conflicts observed between matched rules are resolved by performing CBR.

ELEM2-CBR employs weighting and case ranking methods and can perform both classification and numeric prediction. Given a set of training data, ELEM2-CBR performs RI using ELEM2 to generate a set of classification rules for both tasks. ELEM2's classification is performed over a set of training data after RI and misclassified cases are removed from the case-base, and thus before CBR is performed.

If the task is to predict numeric values, problem solving in ELEM2-CBR is basically a CBR process. Relevance weighting and case ranking methods are employed in case retrieval. Rules generated by ELEM2 are used to determine parameters in the weighting function. After ranking cases in the case-base according to relevance to the new case, several of the most relevant cases are selected and their solutions adapted to the new case.

SUMMARY

Database mining is an exciting and interesting initiative. We have attempted to capture that excitement and interest, while being informative by gradually weaving together a database mining story in three successive, increasingly complex stages, from basic database mining as most people would understand the concept to a second stage, where move advanced operations and techniques are employed. In the first stage we used illustrative examples from DBLEARN, one of the early database mining programs. In the second stage we used examples from DB-Discover and DBROUGH, two contemporary systems currently at the advanced prototype stages; they are still evolving. Finally, the last stage shows a particular, relatively new, advanced representation which appears to be very promising for information and attribute reduction based on rough sets theory. This stage is characterized by examples from both DBROUGH and GRG, a DB-Discover type of prototype system with enhancements from rough sets theory and improved systems and interface design. We added some speculations, a representative sampling of brief commercial product reports, and smoothed the way for more advanced, more general machine learning techniques to be incorporated into database mining in the future.

A wide variety of organizations should countenance database mining to take full advantage of the investment they have made and are currently making in building databases and data warehouses. Alternatively, for database mining systems to fully reach their potential, they must seamlessly interface with data warehouses and other business systems.

The complexity of modern organizations, the overflow of information available to them, and the decisions that they need to make dictate the use of automated computer-based tools to help to manage this melange effectively. Thus database mining should be an essential component of their decision making support and the variety of database mining operations must continue to improve. While not every database mining application will require the use of all operations discussed above, the cooperative use of several database mining operations is probably appropriate in many actual situations, since it is likely that a single data mining technique is not sufficient for addressing every problem within a particular operation. Database mining is sure to continue to evolve and, with increasingly larger storage units and faster, more versatile computers available, the challenges of database mining are also sure to continue.

ACKNOWLEDGMENTS

The authors are members of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada, the Natural Sciences and Engineering Research Council, and the participation of PRECARN Associates Inc. They are grateful to Canadian Cable Labs Fund for their financial assistance. They also thank their students and former students for their contributions: Yandong Cai, who started it all with DBLEARN; Colin Carter, who implemented and contributed much to the development of DB-Discover; Xiaohua Xu, who designed and built DBROUGH; Ning Shan, who continues to contribute to the design of GRG; and Aijun An, who designed and implemented ELEM2-CBR. Other students and members of the IRIS laboratory have made contributions.

GLOSSARY

This glossary was adapted from the World Wide Web sites of *Pilot Software* (www.pilotsw.com/r_and_t/whtpaper/datamine/dmglos.htm), *Creative Data, Incorporated* (www.std.com/CreativeData/credata/termin.html), and *N. Cercone.*

Analytical model. A structure and process for analyzing a dataset; for example, a decision tree is a model for the classification of a dataset.

Artificial neural networks. Nonlinear predictive models that learn through training and resemble biological neural networks in structure.

Bitmapped indexing. A family of advanced indexing algorithms that optimize RDBMS query performance by maximizing the search capability of the index per unit of memory and per CPU instruction. Properly implemented, bitmapped indices eliminate all table scans in query and join processing.

Business model. An object-oriented model that captures the kinds of things in a business or a business area and the relationships associated with those things (and sometimes associated business rules, too). Note that a business model exists independently of any data or database. A data warehouse should be designed to match the underlying business models or else no tools will fully unlock the data in the warehouse.

CART. Classification and regression trees; a decision tree technique used for classification of a dataset. Provides a set of rules that you can apply to a new (unclassified) dataset to predict which records will have a given outcome. Segments a dataset by creating two-way splits. Requires less data preparation than CHAID.

CHAID. Chi-square automatic interaction detection; a decision tree technique used for classification of a dataset. Provides a set of rules that one can apply to a new (unclassified) dataset to predict which records will have a given outcome. Segments a dataset by using chi-square tests to create multiway splits. Preceded, and requires more data preparation than, CART.

Classification. The process of dividing a dataset into mutually exclusive groups such that the members of each group are as "close" as possible to one another, and different groups are as "far" as possible from one another, where distance is measured with respect to specific variable(s) one is trying to predict. For example, a typical classification problem is to divide a database of companies into groups that are as homogeneous as possible, with respect to a creditworthiness variable with values "Good" and "Bad."

Clustering. The process of dividing a dataset into mutually exclusive groups such that the members of each group are as "close" as possible to one another, and different groups are as "far" as possible from one another, where distance is measured with respect to all available variables.

Corporate data. All the databases of the company. This includes legacy systems, old and new transaction systems, general business systems, client/server databases, data warehouses, and data marts.

Data cleansing. The process of ensuring that all values in a dataset are consistent and correctly recorded.

Data dictionary. A collection of metadata. Many kinds of products in the data warehouse arena use a data dictionary, including database management systems, modeling tools, middleware, and query tools.

Data mart. A subset of a data warehouse that focuses on one or more specific subject areas. The data usually is extracted from the data warehouse and further denormalized and indexed to support intense usage by targeted customers.

Database mining. The extraction of hidden predictive information from large databases; techniques for finding patterns and trends in large data sets. See also *data visualization.*

Data model. The road map to the data in a database. This includes the source of tables and columns, the meanings of the keys, and the relationships between the tables.

Data navigation. The process of viewing different dimensions, slices, and levels of detail of a multidimensional database. See OLAP.

Data visualization. The visual interpretation of complex relationships in multidimensional data; designed to make particular kinds of visualization easy.

Data warehouse. A system for storing and delivering massive quantities of data; typically a data warehouse is fed from one or more transaction databases. The data need to be cleaned and restructured to support queries, summaries, and analyses.

Decision support. Data access targeted to provide the information needed by business decision makers. Examples include pricing, purchasing, human resources, management, and manufacturing.

Decision support system (DSS). Database(s), warehouse(s), and/or mart(s) in conjunction with reporting and analysis software optimized to support timely business decision making.

Decision tree. A tree-shaped structure that represents a set of decisions. These decisions generate rules for the classification of a dataset. See CART and CHAID.

Deduction. A technique that infers information that is a logical consequence of the information in the database. RDBMSs offer methods of expressing deductive relationships between data. Expert systems and case based reasoning systems are examples of deductive data mining tools.

Dimension. In a flat or relational database, each field in a record represents a dimension. In a multidimensional database, a dimension is a set of similar entities; for example, a multidimensional sales database might include the dimensions Product, Time, and City.

Exploratory data analysis. The use of graphical and descriptive statistical techniques to learn about the structure of a dataset.

Genetic algorithms. Optimization techniques that use processes such as genetic combination, mutation, and natural selection in a design based on the concepts of natural evolution.

Induction. A technique to infer generalizations from the information in the database. General statements about properties of objects can be viewed as inferred knowledge. Neural networks and rule induction systems are examples of inductive data mining tools.

Linear model. An analytical model that assumes linear relationships in the coefficients of the variables studied.

Linear regression. A statistical technique used to find the best-fitting linear relationship between a target (dependent) variable and its predictors (independent variables).

Logistic regression. A linear regression that predicts the proportions of a categorical target variable, such as type of customer, in a population.

Metadata. Literally, "data about data." More usefully, descriptions of what kind of information is stored where, how it is encoded, how it is related to other information, where it comes from, and how it is related to your business. A hot topic right now is standardizing metadata across products from different vendors.

Middleware. Hardware and software used to connect clients and servers, to move and structure data, and/or to presummarize data for use by queries and reports.

Multidimensional database. A database designed for online analytical processing. Structured as a multidimensional hypercube with one axis per dimension.

Nearest neighbor. A technique that classifies each record in a dataset based on a combination of the classes of the k record(s) most similar to it in a historical dataset (where k is greater than or equal to 1). Sometimes called a *k-nearest neighbor technique.*

Nonlinear model. An analytical model that does not assume linear relationships in the coefficients of the variables being studied.

Object oriented analysis (OOA). A process of abstracting a problem by identifying the kinds of entities in the problem domain, the is-a relationships between the kinds (kinds are known as classes, is-a relationships as subtype/supertype, subclass/superclass, or less commonly, specialization/generalization), and the has-a relationships between the classes. Also identified for each class are its attributes (e.g., class Person has attribute Hair Color) and its conventional relationships to other classes (e.g., class Order has a relationship Customer to class Customer.)

Object oriented design (OOD). A design methodology that uses Object Oriented Analysis to promote object reusability and interface clarity.

On-line analytical processing (OLAP). A common use of a data warehouse that involves real time access and analysis of multidimensional data such as order information; refers to array-oriented database applications that allow users to view, navigate through, manipulate, and analyze multidimensional databases.

OLTP. On-line transaction processing. Refers to a database which is built for on-line transaction processing, generally regarded as unsuitable for data warehousing; *OLTP* systems have been designed to answer "simple aggregations" such as "what is the current account balance for this customer?"

Parallel processing. The coordinated use of multiple processors to perform computational tasks. Parallel processing can occur on a multiprocessor computer or on a network of workstations or PCs.

Predictive model. A structure and process for predicting the values of specified variables in a dataset.

Prospective data analysis. Data analysis that predicts future trends, behaviors, or events based on historical data.

Query. A specific request for information from a database.

Relational on-line analytic processing (ROLAP). OLAP based on conventional relational databases rather than specialized multidimensional databases.

Replication. A standard technique in data warehousing. For performance and reliability, several independent copies are often created of each data warehouse. Even data marts can require replication on multiple servers to meet performance and reliability standards.

Replicator. Any of a class of product that supports replication. Often these tools use special load and unload database procedures and have scripting languages that support automation.

Retrospective data analysis. Data analysis that provides insights into trends, behaviors, or events that have already occurred.

Rule induction. The extraction of useful if-then rules from data based on statistical significance.

Time series analysis. The analysis of a sequence of measurements made at specified time intervals. Time is usually the dominating dimension of the data.

BIBLIOGRAPHY

1. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, Knowledge discovery and data mining: Towards a unifying approach, *Proc. 2nd Int. Conf. Knowledge Discovery Data Mining,* Menlo Park, CA: AAAI Press, 1996, pp. 82–88.
2. P. Langley and H. A. Simon, Applications of machine learning and rule induction, *Commun. ACM,* **38** (11): 54–64, 1995.
3. G. Pietetsky-Shapiro and W. J. Frawley, *Knowledge Discovery in Databases,* Cambridge, MA: AAAI/MIT Press, 1991.
4. B. Morton, Defining data mining, *DBMS Data Warehouse Supplement, DBMS Online,* August, 1996.
5. W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, Knowledge discovery in databases: An overview, in G. Piatetsky-Shapiro and W. J. Frawley (eds.), *Knowledge Discovery in Databases,* Cambridge, MA: AAAI/MIT Press, 1991, pp. 1–27.
6. R. Agrawal and R. Srikant, Fast algorithms for mining association rules, *Proc. 20th Int. Conf. Very Large Databases,* Santiago, Chile, 1994.
7. R. Hull and R. King, Semantic database modeling: Survey, applications, and research issues, *ACM Comput. Surveys,* **19**: 201–260, 1987.
8. A. Sheth and J. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surveys,* **22**: 183–236, 1990.
9. W. Kim and F. Lochovsky, *Object-Oriented Languages, Applications, and Databases,* Reading, MA: Addison-Wesley, 1989.
10. X. Hu, N. Cercone, and J. Xie, Learning data trend regularities from databases in a dynamic environment, *Knowledge Discovery in Databases KDD-94,* Seattle, 1994, pp. 323–334.
11. J. Han et al., Discovery of data evolution regularities in large databases. *J. Comput. Softw. Eng.,* special issue on *Methodologies Tools Intell. Inf. Syst.,* **3** (1): 41–69, 1995.
12. J. R. Quinlan, *C4.5: Programs for Machine Learning,* Morgan Kaufmann, San Mateo, CA, 1993.
13. R. S. Michalski et al., The multi-purpose incremental learning system AQ15 and its testing applications to three medical domains, *Proc. AAAI-86,* 1986, pp. 1041–1044.
14. P. Clark and R. Boswell, Rule induction with CN2: Some recent improvements. *Proc. Eur. Working Session Learning,* Porto, Portugal, 1991, pp. 151–163.
15. P. Clark, Machine learning: Techniques and recent developments. In A. R. Mirzai (ed.), *Artificial Intelligence: Concepts and Applications in Engineering.* Chapman and Hall, London: 1990.
16. A. Aamodt and E. Plaza, Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intell. Commun.,* **7** (1): 39–59, 1994.
17. Y. Cai, N. Cercone, and J. Han, Attribute-oriented induction in relational databases, in G. Piatetsky-Shapiro and W. J. Frawley (eds.), *Knowledge Discovery in Databases,* Cambridge, MA: AAAI/MIT Press, 1991, pp. 213–228.
18. J. Han, Y. Cai, and N. Cercone, Knowledge discovery in databases: An attribute-oriented approach, *Proc. 18th VLDB Conf.,* Vancouver, Canada, 1992, pp. 547–559.
19. J. Han, Y. Cai, and N. Cercone, Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowl. Data Eng.,* **5**: 29–40, 1993.
20. Y. Cai, N. Cercone, and J. Han, Learning in relational databases: An attribute oriented approach. *Computat. Intell.,* **7** (3): 119–132, 1991.
21. T. Dietterich and R. Michalski, A comparative review of selected methods for learning from examples, in R. S. Michalski et al. (eds.), *Machine Learning: An Artificial Intelligence Approach, vol. 1,* San Mateo, CA: Morgan Kaufmann, 1983, pp. 41–82.
22. T. G. Dietterich and R. S. Michalski, A theory and methodology of inductive learning, in R. S. Michalski et al. (eds.), *Machine Learning: An Artificial Intelligence Approach, vol. 1,* San Mateo, CA: Morgan Kaufmann, 1983, pp. 43–82.
23. E. F. Codd, S. B. Codd, and C. T. Salley, *Providing OLAP (Online Analytical Processing) to user-analysts: an IT mandate,* San Jose, CA: Codd and Date, 1993.
24. Z. Pawlak, Rough sets, *Inf. Comput. Sci.,* **11** (5): 341–356, 1982.
25. Z. Pawlak, *Rough sets, Theoretical Aspects of Reasoning About Data,* Norwell, MA: Kluwer, 1991.
26. T. Y. Lin and N. Cercone (eds.), *Applications of Rough Sets and Data Mining,* Norwell, MA: Kluwer, 1997.
27. X. Hu and N. Cercone, Learning in relational databases: A rough set approach, *Comput. Intell.,* **11** (2): 323–338, 1995.
28. T. Y. Lin, Fuzzy controllers: An integrated approach based on fuzzy logic, rough sets, and evolutionary computing, in T. Y. Lin and N. Cercone (eds.), *Applications of Rough Sets and Data Mining,* Norwell, MA: Kluwer, 1997, pp. 123–138.
29. X. Hu and N. Cercone, Rough sets similarity-based learning from databases, *Knowledge Discovery in Databases KDD-95,* Montreal, 1995.
30. A. Skowron and L. Polkowski, Synthesis of decision systems from data tables, in T. Y. Lin and N. Cercone (eds.), *Applications of Rough Sets and Data Mining,* Norwell, MA: Kluwer, 1997, pp. 259–300.
31. N. Shan et al., Using rough sets as tools for knowledge discovery from large relational databases, *Knowledge Discovery in Databases KDD-95,* Montreal, 1995, pp. 263–268.
32. C. Carter and H. Hamilton, Performance improvement in the implementation of DBLEARN, TR CS-94-05, University of Regina, Canada, 1994.
33. C. Matheus, P. Chan, and G. Piatetsky-Shapiro, Systems for knowledge discovery in databases, *IEEE Trans. Knowl. Data Eng.,* **5**: 903–913, 1993.
34. K. Kira and L. Rendell, The feature selection problem: traditional methods and a new algorithm, *AAAI-92,* Cambridge, MA: MIT Press, 1992, pp. 129–134.

35. Y. Xiang, M. Wong, and N. Cercone, Quantification of uncertainty in classification rules discovered from databases, *Computat. Intell.,* **11** (2): 427–441, 1995.

36. N. Shan, H. Hamilton, and N. Cercone, GRG: Knowledge and discovery using information generalization, information reduction, and rule generation, *Int. J. Artificial Intell. Tools* **5** (1&2): 99–112, 1996.

37. W. Ziarko, Variable precision rough set model, *Comput. Syst. Sci.,* **46** (1): 39–59, 1993.

38. X. Hu, N. Cercone, and J. Han, An attribute-oriented rough set approach for knowledge discovery in databases, *Int. Workshop Rough Sets Knowl. Discovery (RSKD-93),* Banff, 1993, pp. 79–94.

39. M. Wong and W. Ziarko, On optimal decision rules in decision tables, *Bulletin Polish Acad. Sci.,* **33** (11&12): 693–696, 1985.

40. W. Press et al., *Numerical recipes in C: The art of scientific computing,* Cambridge, MA: Cambridge University Press, 1988.

41. X. Hu and N. Cercone, Mining knowledge rules from databases: A rough set approach, *12th Int. Conf. Data Eng.,* New Orleans, 1995.

42. N. Shan et al., Discretization of continuous valued attributes in attribute-value systems, *Fifth Rough Sets, Fuzzy Sets, and Machine Discovery RFSD'96,* Tokyo, 1996, pp. 74–81.

43. N. Shan, H. Hamilton, and N. Cercone, The GRG knowledge discovery system: Design principles and architectural overview, *9th Eur. Conf. Mach. Learning ECML-97,* Prague, 1997.

44. W. Ziarko and N. Shan, Knowledge discovery as a search for classification, *Workshop Rough Sets Database Mining, 23rd Annu. Comput. Sci., CSC'95,* 1995.

45. J. R. Quinlan, *C4.5: Programs for Machine Learning,* San Mateo, CA: Morgan Kaufmann, 1993.

46. R. S. Michalski et al., The multi-purpose incremental learning system AQ15 and its testing application to three medical domains, *Proc. AAAI-86,* 1986, pp. 1041–1044.

47. P. Clark and R. Boswell, Rule induction with CN2: Some recent improvements, *Proc. Eur. Working Session Learning,* Porto, Portugal, 1991, pp. 151–163.

48. P. Clark, Machine learning: Techniques and recent developments, in A. R. Mirzai (ed.), *Artificial Intelligence: Concepts and Applications in Engineering,* London: Chapman and Hall, 1990.

49. D. B. Leake, CBR in context: The present and future, in D. B. Leake (ed.), *Case-Based Reasoning: Experiences, Lessons, and Future Directions,* Menlo Park, CA: AAAI Press, 1996.

50. P. Koton, Using experience in learning and problem solving, *Ph.D. Dissertation,* Laboratory of Computer Science, Massachusetts Institute of Technology, MIT/LCS/TR-441, 1989.

51. K. Althoff, S. Wess, and R. Traphoner, INRECA—A seamless integration of induction and case-based reasoning for decision support tasks, *Proc. 8th Workshop German Special Interest Group Mach. Learning,* 1995.

52. E. R. Bareiss and C. C. Wier, Protos: An exemplar-based learning apprentice. *Proc. 4th Int. Workshop Mach. Learning.* Irvine, CA, 1987.

53. D. B. Skalak and E. L. Rissland, Inductive learning in a mixed paradigm setting, *AAAI-90,* 1990, pp. 840–847.

54. P. Domingos, Rule induction and instance-based learning: A unified approach. *IJCAI-95,* Montreal, Canada, 1995, pp. 1226–1232.

55. A. R. Golding and P. S. Rosenbloom, Improving rule-based systems through case-based reasoning, *AAAI-91,* 1991, pp. 22–27.

56. A. An, *Integrated analysis tools for enhanced problem solving, Ph.D. Thesis,* Dept. of Computer Science, University of Regina, Regina, Canada, 1997.

NICK CERCONE
HOWARD HAMILTON
University of Waterloo