To appear in the *IEEE TKDE* special issue on *Learning & Discovery in Knowledge-Based Databases*, 1993.

Systems for Knowledge Discovery in Databases

Christopher J. Matheus Philip K. Chan Gregory Piatetsky-Shapiro

GTE Laboratories Incorporated
40 Sylvan Road, Waltham, MA 02254
matheus@gte.com, pkc@gte.com, gps0@gte.com

Abstract

The automated discovery of knowledge in databases is becoming increasingly important as the world's wealth of data continues to grow exponentially. Knowledge-discovery systems face challenging problems from real-world databases which tend to be dynamic, incomplete, redundant, noisy, sparse, and very large. This paper addresses these problems and describes some techniques for handling them. A model of an idealized knowledge-discovery system is presented as a reference for studying and designing new systems. This model is used in the comparison of three systems: CoverStory, EXPLORA, and the Knowledge Discovery Workbench. The deficiencies of existing systems relative to the model reveal several open problems for future research.

Contents

1. Introduction

Knowledge Discovery in Databases (KDD) is an active research area with promise for high payoffs in many business and scientific domains [Piatetsky-Shapiro and Frawley, 1991, Piatetsky-Shapiro, 1991b, Piatetsky-Shapiro, 1992b]. The corporate, governmental, and scientific communities are being overwhelmed with an influx of data that is routinely stored in on-line databases. Analyzing this data and extracting meaningful patterns in a timely fashion is intractable without computer assistance and powerful analytical tools. Standard computer-based statistical and analytical packages alone, however, are of limited benefit without the guidance of trained statisticians to apply them correctly and domain experts to filter and interpret the results. The grand challenge of knowledge discovery in databases is to automatically process large quantities of raw data, identify the most significant and meaningful patterns, and present these as knowledge appropriate for achieving the user's goals.

The realization of a general-purpose, fully-automated, knowledge-discovery system is far from reach. Much of the research in KDD has thus far focused on ways of manually applying traditional machine-learning and discovery methods to data stored in relational databases. With this approach, the user must provide significant guidance by, for example, selecting the portion of data to explore, identifying relevant fields, and specifying potential target concepts. Recently, attention has been shifting towards more fully automated approaches. Newer systems are beginning to use knowledge of the database domain to autonomously select relevant fields, to guide the application of various pattern-extraction algorithms, and to identify and filter the most meaningful results for presentation. In addition, new techniques, such as data-dependency analysis and deviation detection, are beginning to show promise as powerful components of KDD systems.

In this paper we analyze the problem of applying automated discovery to large, real-world databases. We begin with a review of databases, identifying some of their characteristics that make automated discovery challenging. We then propose a model of an idealized KDD system and outline its essential components. Finally, we compare three KDD systems: CoverStory^{tm} [Schmitz *et al.*, 1990], EXPLORA [Hoschka and Klosgen, 1991], and the Knowledge Discovery Workbench [Piatetsky-Shapiro and Matheus, 1991].

2. Database Issues

A *database* is an integrated collection of data maintained in one or more files, organized to facilitate the efficient storage, modification, and retrieval of related information [Date, 1977]. Although databases have been designed around various representational models, we will focus exclusively on the *relational* model because of its prevalence in large, real-world databases. (See [Dzeroski and Lavrac, 1993] in this issue for an application of discovery to *deductive databases*.)

In a relational database, data are organized into *tables* of fixed-length *records* as depicted in Figure 1. Each record is an ordered list of values, one value for each *field*. Values are either explicit data elements (e.g. numbers or strings) or logical pointers to records in other tables. Usually a separate table, called a *data dictionary*, contains information about each field's name, type, and possibly a range of permitted values. A *database management system*

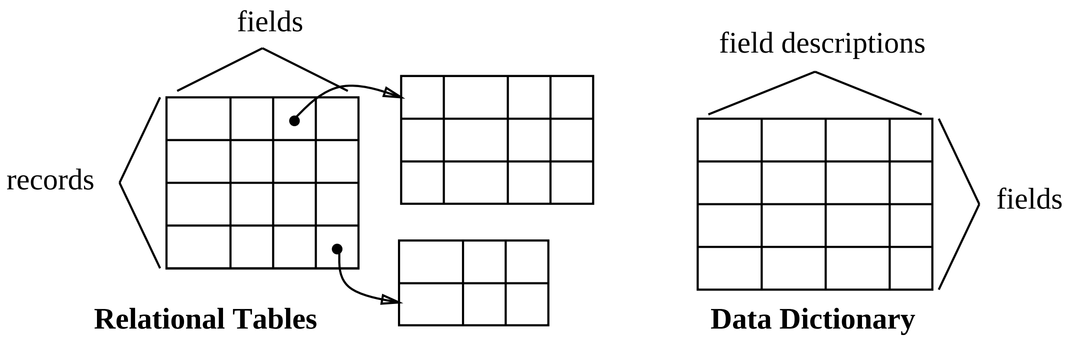


Figure 1: The components of a relational database.

(DBMS) is a collection of procedures for retrieving, storing, and manipulating data within a set of database tables.

In many cases, the separate tables of a relational database can be logically joined by constructing a *universal relation* (UR) [Ullman, 1982]. A UR is either computed and stored, or, if too large, logically represented through a UR interface. An external application using a UR interface can treat the database as a single, flat file (though perhaps inefficiently). As a result, existing machine-learning and discovery algorithms based on attribute-value representations can be readily applied to relational databases by treating each record in the UR as a single training instance. Structural or relational learning systems, e.g. FOIL [Quinlan, 1989a] and Subdue [Holder and Cook, 1993], do not need the UR, having been specifically designed to operate on relational data.

Although many machine-learning algorithms are readily applicable, real-world databases present additional difficulties due to the nature of their contents which tend to be dynamic, incomplete, redundant, noisy, sparse, and very large. Each of these issues has been addressed to some extent within machine learning, but few if any systems effectively address them all. Collectively handling these problems while producing useful knowledge is the challenge of KDD. In the rest of this section we look more closely at each of these issues.

Dynamic data: A fundamental characteristic of most databases is that their contents are ever changing. In an online system, precautions must be taken to ensure that these changes do not lead to erroneous discoveries. A common approach is to take snapshots of the data. This method is most appropriate when data is collected periodically, such as quarterly or yearly; its primary drawback is the additional storage requirement for each snapshot. Some DBMS's put timestamps on data as it is entered or changed, making it possible to perform consistency checks, although we are not aware of any discovery systems that do this. In the future, DBMS's may evolve with builtin discovery mechanisms to automatically handle aspects of this problem. A step in that direction can be found in mechanisms such as triggers or active rules (cf. the design of Postgres [Stonebraker, 1985]).

Noise and uncertainty: Erroneous data can be a significant problem in real-world databases, primarily because the error-prone, manual collection and entry of data is still commonplace. In some discovery systems, finding and correcting these data-entry errors is the main objective (see [Schlimmer, 1991]). More often, this uncertainty in the correctness of the data represents a problem that can necessitate larger data samples or stronger biases (e.g. more domain knowledge), and adds to the uncertainty of the final results.

Another type of uncertainty is in the discovered patterns. The patterns most useful to the end user often are valid over some, but not all of the data. For example, the pattern "customers with high-income are good credit risks" might be very useful even though it is not always true. Finding and representing these types of patterns requires probabilistic methods and representations, which are common in most systems.

Incomplete data: Data can be incomplete either through the absence of values in individual record fields, or through the complete absence of data fields necessary for certain discoveries. The missing-values problem is a familiar issue in machine learning [Quinlan, 1989b]. In relational databases, the problem occurs frequently because the relational model dictates that all records in a table must have the same fields, even if values are nonexistent for most records. Consider, for example, a hospital database with fields for a wide range of laboratory tests and procedures; only a few of these fields would be filled in for any given patient. This problem may be lessened in object-oriented databases which permit more flexible representations of data; but these have not yet gained wide acceptance, and when they do, they will undoubtedly create new kinds of problems for knowledge discovery.

Databases are seldom designed with discovery in mind; instead they are intended for some organizational activity, and discovery happens as an afterthought. This situation becomes problematic when the discovery, evaluation, or explanation of important patterns requires information not present in the database, i.e. the missing-field problem. Suppose, for example, a discovery system is employed to help identify and explain sales differences among regional divisions of a corporation. Without explicit information about regional demographics and market factors (items not usually present in a corporation's sales database), meaningful discoveries are unlikely. Recent work by Scheines and Spirtes [1992] suggests an approach for identifying where information is missing in a database by finding combinations of fields which are jointly influenced by latent (missing) fields.

Redundant information: Information often re-occurs in multiple places within a database. A common form of redundancy is a *functional dependency* in which a field is defined as a function of other fields, for example: `Profit` = `Sales` − `Expenses`. A weaker form of dependency occurs when a field is merely constrained by other information, e.g. `BeginDate` ≤ `EndDate`. The problem with redundant information is that it can be mistakenly discovered as knowledge, even though it is usually uninteresting to the end user [Piatetsky-Shapiro and Matheus, 1991]. To avoid this problem, a system needs to know the database's inherent dependencies. Some of this structure can be uncovered through the automatic discovery of dependencies as discussed in section 3.5.1, although the user must still confirm the validity of the patterns.

Sparse Data: The information in a database is often sparse in terms of the density of actual data records over the potential instance space [Zytkow and Baker, 1991]. Rare diseases, for example, occur so infrequently that few patient records in a clinical database are likely to refer to them. Less extremes situations are more common but just as challenging for empirical discovery algorithms. Assume for example that the top five percent of customers in a database generate eighty percent of total revenues. If this concept of "high revenue customers" is complex (i.e. involves several interacting factors), five percent of the records may be insufficient to accurately discover the concept with empirical techniques. One method for dealing with these situations is to take a "stratified" or selective sample that over emphasizes the events of interest [Buntine, 1991].

Data Volume: The rapid growth of data in the world's databases is one reason for the recent interest in KDD. The vastness of this data also creates one of KDD's greatest challenges. Exhaustive, empirical analysis is all but impossible on the megabytes, gigabytes, or even terabytes of data in many real-world databases. In these situations, a KDD system must be able to focus its analysis on samples of data by selecting specific fields and/or subsets of records. Because databases usually contain some fields that are redundant, irrelevant, or unimportant to a given discovery task, focusing on a subset of fields is now common practice [Piatetsky-Shapiro and Matheus, 1991, Hoschka and Klosgen, 1991]. Focusing further on a subset of records, which becomes necessary with larger databases, is achievable by random sampling methods, or by using selection constraints to limit attention to subclasses of records, e.g. selecting the top ten percent of customer records based on spending. Section 3.4 deals with this issue of focus in more detail.

Summary: The quality (or lack of) and vastness of the data in real-world databases represent the core problems for KDD. Overcoming the quality problem requires external domain knowledge to clean-up, refine, or fill in the data. The vastness of the data forces the use of techniques for focusing on specific portions of the data, which requires additional domain knowledge if it is to be done intelligently. A KDD system therefore must be able to represent and appropriately use domain knowledge in conjunction with the application of empirical discovery algorithms.

3. A KDD Model

This paper concerns systems that perform knowledge discovery on databases. Such systems must address the issues raised in the previous section. In this section we present a model of an idealized KDD system and describe how its components are intended to handle the specific requirements for discovery in real-world databases. We begin by providing a working definition of a KDD system.

Generally speaking, a discovery is a finding of something previously unknown. A *knowledge-discovery system*, then, is a system that finds knowledge that it previously did not have, i.e. it was not implicit in its algorithms or explicit in its representation of domain knowledge. For our purposes, a piece of knowledge is a relationship or pattern among data elements that is interesting and useful with respect to a specific domain and task [Frawley *et al.*, 1991]. When a knowledge-discovery system operates on the data in a real-world database, it becomes a Knowledge Discovery in Databases (KDD) system. More specifically:

> *A KDD system comprises a collection of components that together can efficiently identify and extract interesting and useful new patterns from data stored in real-world databases.*

Inherent in the meaning of discovery is the notion of autonomy, as connoted, for example, in the term "unsupervised learning" used to describe discovery algorithms in machine learning [Michalski *et al.*, 1983]. If we consider the discovery algorithms used in existing KDD systems, we find few that are totally autonomous, and those that are have limited

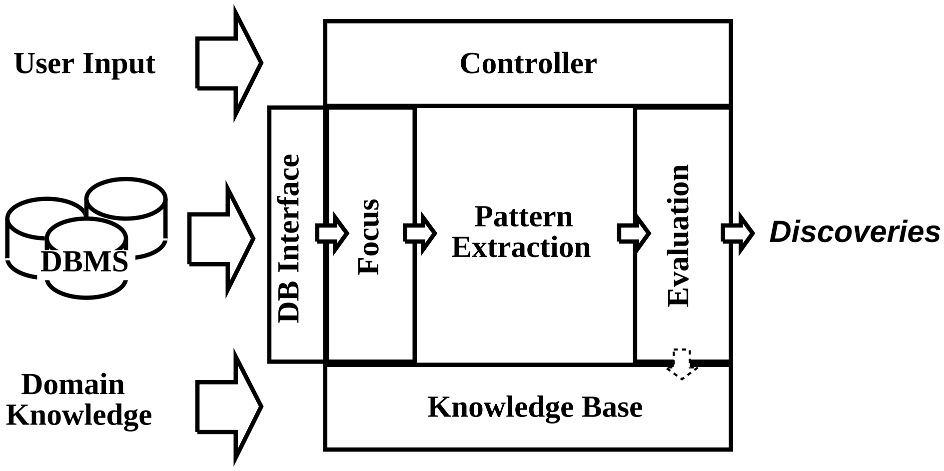


Figure 2: Model of a KDD System

applicability. In nearly all systems, human guidance is required to some degree. For this reason it is sometimes convenient to view the human as a part of the KDD system. One goal of KDD research is to reduce the amount of human direction required by discovery systems. When we discuss specific systems later, we will consider how each fairs in this respect.

Our definition defines a system as a collection of components. While the components may differ among systems, certain basic functions are usually identifiable. We have incorporated these as components in a model of an idealized KDD system as shown in Figure 2. The models components include:

- **Controller:** controls the invocation and parameterization of the other components
- **Database Interface:** generates and processes database queries
- **Knowledge Base:** repository of domain specific information
- **Focus:** determines what portions of the data to analyze
- **Pattern Extraction:** a collection of pattern-extraction algorithms
- **Evaluation:** evaluates the interestingness and utility of extracted patterns

Information flows into the system from three sources: the user issues high-level commands to the controller, the DBMS provides the raw data, and domain knowledge from various sources is deposited into the system's knowledge base. Raw data is selected from the DBMS and then processed by the extraction algorithms which produce candidate patterns. These patterns are then evaluated and some are identified as interesting discoveries. In addition to presenting the results to the user[1], discovered patterns may be deposited into the system's knowledge base to support subsequent discoveries.

[1]The presentation of results is an important problem for KDD, since discoveries that cannot be effectively communicated are of no use. Unfortunately this topic is beyond the scope of this paper, and so it was not included in the model. Interested readers may refer to the following references [Tufte, 1983], [Roth and Mattis, 1991], and [Klosgen, 1991].

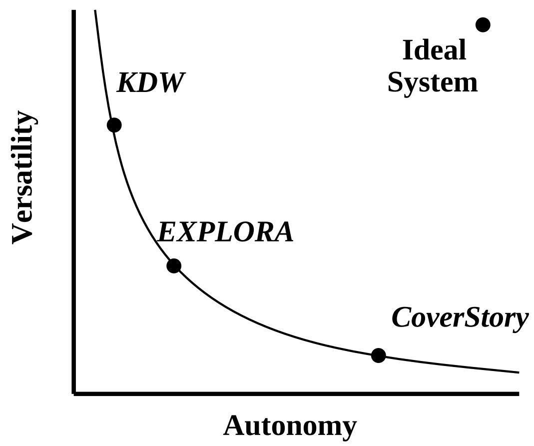


Figure 3: The tradeoff between autonomy and versatility in KDD systems.

Our model represents an abstraction of what occurs in KDD systems. In an actual system, it is not always possible to cleanly separate the individual components of the model. Nonetheless, we have found this abstraction useful in comparing existing approaches and in guiding the design of new systems. The model is particularly helpful for contrasting the tradeoffs between autonomy (i.e., the degree of freedom from human direction) and versatility (i.e., the range of domains and types of discoveries achievable). The line on the graph in Figure 3 depicts the characteristic tradeoff observed between autonomy and versatility in existing systems. The more autonomous a system is, the less versatile it tends to be, owing to its greater reliance on domain knowledge specific to a more focused task. The more versatile systems generally have a wider range of discovery techniques and are designed to work across broader domains, at the expense of greater reliance on user guidance. We have indicated on the graph the approximate locations of the three systems we will be analyzing later. The goal of KDD also appears on the graph as the point where both autonomy and versatility are maximized in the ideal system. The components in our model system are explored in the following subsections.

3.1. Domain Knowledge and User Input

A data dictionary contains information about the form of the contents of a database, specifying the field names, types, and perhaps simple value constraints. Additional information about the data's structure and inter-field constraints invariably exists outside the database in specifications, manuals, and domain experts. Further information about the specific analysis objectives may come from the end user. This collection of supplementary information, or *domain knowledge*, can assume many forms. A few examples include: lists of relevant fields on which to focus attention; definitions of new fields (e.g. `AGE` = `CURRENT_YEAR` − `YEAR_OF_BIRTH`); lists of useful classes or categories of fields or records (e.g. Revenue fields: `Profits`, `Expenses`, etc.); generalization hierarchies (e.g. `C` is-a `B` is-a `A`); functional or causal models (see Figure 4).

The primary purpose of domain knowledge is to bias the search for interesting patterns. This can be achieved by focusing attention on portions of the data, biasing the extraction algorithms, and assisting in pattern evaluation. The benefits can be greater efficiency and more relevant discoveries; reliance on domain knowledge, however, also can preclude the discovery of potentially useful patterns by leaving portions of the search space unexplored. A trade-off thus exists between efficiency and completeness.

Domain experts are the usual source of all but the simplest domain knowledge; the extraction and codification of this knowledge is a challenging problem in the development of a KDD system. Fortunately it is possible for some forms of domain knowledge to be discovered. An example of this is discussed in the section on data dependencies (section 3.5.1) which describes methods for automatically determining some of the interrelationships between data fields. Ideally, a KDD system would store all discoveries as new domain knowledge and use them to support and drive subsequent discoveries – this, however, remains an open problem.

3.2. Controlling Discovery

In our model system, autonomy comes from the controller. The controller's decisions are based on the information provided by the domain knowledge and user input. The controller interprets this information and uses it to direct the focus, extraction, and evaluation components. In some systems where the discovery task is well defined and static (e.g. CoverStory in section 4.1), the controller may simply execute a predefined sequence of operations. In more versatile systems, the controller may assume greater decision-making responsibility. In practice, many KDD systems require participation by the end user in making the majority of these decisions (cf. InLen [Kaufman *et al.*, 1991] and KDW [Piatetsky-Shapiro and Matheus, 1991]).

3.3. Interfacing to a DBMS

Data is extracted from a DBMS using queries. A typical query lists a set of fields to retrieve from a set of tables according to specified constraints. Many relational databases support a standard query language called SQL (structured query language). The following is an example of a simple SQL query that selects three fields from all claims records in which the "payment received" is lower than the "charge:"

```
select INSURANCE_CARRIER, PAYMENT_RECEIVED, CHARGE
from CLAIMS_TABLE
where PAYMENT_RECEIVED < CHARGE
```

The DBMS interface is where database queries are generated. This operation can be done without intelligence, although recent research (see [Han *et al.*, 1993] and [Shekhar *et al.*, 1993] in this issue) has shown how discovery techniques can improve the performance and results of database queries. In our model, the DBMS interface plays a subordinate but important role. It can be argued that a system that does "discovery on databases" *must* be able to access a database. From a practical perspective, a direct DBMS interface becomes critical with large databases, where the data to be analyzed cannot fit into working memory all at once and queries must be generated to access data upon demand. This issue has received little attention in the KDD literature, and many systems do not yet have builtin DBMS interfaces.

3.4. Focusing

The focusing component of a discovery system determines what data should be retrieved from the DBMS. This requires specifying which tables need to be accessed, which fields should be returned, and which or how many records should be retrieved. To do this operation, the focus component needs detailed information about the database table structures; it must know which fields are appropriate for the current task; if it is doing data sampling, it must have a way of randomly selecting the appropriate number of records; and, it must know the input required by the subsequent extraction algorithms to properly format the results.

Identifying relevant fields is the most common focusing technique. This can occur as an explicit list of relevant fields in the domain knowledge, or as the result of an extraction algorithm requesting specific fields on demand as the need arises. Limiting the number of fields alone may not sufficiently reduce the size of the data set, in which case a subset of records must be selected. Record sampling can be done randomly with the intent of taking a large enough sample to statistically justify the results.[2] Alternatively, a logical predicate can be used to select a small subset of records sharing some common characteristics. For example, the predicate `HIGH_CREDIT(X)` could be used to select the top N% of customer records based on their credit ratings. A novel approach to field- and record-based focusing is in the use of "abstracts" as described in [Dhar and Tuzhilin, 1993] in this issue.

3.5. Extracting Patterns

The term *pattern* refers to any relation among elements of a database, i.e. the records, fields, and values. Simple examples of patterns include: $\texttt{ADMISSION\_DATE}_x < \texttt{RELEASE\_DATE}_x$; *if* $\texttt{REGION}_x =$ *"west" then* $\texttt{SALES}_x >$ average(`SALES`). When such patterns are probabilistic, uncertainty measures may appear as annotations. More complex patterns can be built up from these simple structures into intricate networks of relationships among multiple fields and values, cf. dependency networks.

The algorithms used to extract these patterns form the core of any discovery system. A wide variety of machine-learning and statistical-analysis algorithms have been incorporated into KDD systems. Rather than review all of these, we will consider four generic tasks: dependency analysis, class identification, class description, and deviation detection.

3.5.1. Dependency Analysis

Data dependencies represent an important class of discoverable knowledge. A dependency exists between two items if the value of one item can be used to predict the value of the other: $A \rightarrow B$. An item in a dependency can be a field or a relation among fields. The exact dependency function may or may not be known, and when it is, it may be probabilistic rather than certain, e.g. $A \stackrel{.95}{\rightarrow} B$. A collection of related dependencies defines a dependency graph, such as the one depicted in Figure 4. The notion of dependency graphs generalizes the basic statistical measures of correlation coefficients and linear regression by addressing both numeric and discrete variables and by organizing all dependencies in a single structure.

[2] Even when nothing is known about the distribution, non-parametric statistics [Dixon and Massey, 1979] can be used to provide an upper-bound on the error in estimating numeric value distributions, regardless the size of the original database.

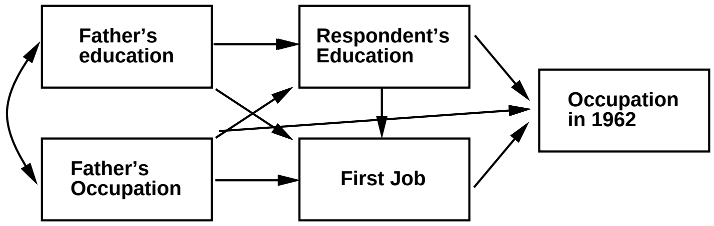


Figure 4: A data-dependency graph derived by TETRAD [Glymour *et al.* 1987] depicting the American Occupational Structure according to a 1962 survey of 20,000 people.

Exact or *functional dependencies* have been the subject of database research since the 1970's. Several algorithms now exist for using functional dependencies to create normalized databases that minimize redundancies and facilitate updates [Ullman, 1982]. An asymptotically optimal algorithm also exists for finding the minimal set of functional dependencies in a database [Mannila and Raiha, 1987].

In recent years, research by Pearl [1988, 1991], Glymour *et al.* [1987], and others has resulted in major advances in the area of discovering *dependency* or *causal graphs*. Because standard statistical techniques cannot distinguish causation from covariation, data precedence information or assumptions are needed to establish the direction of influence. The proposed methods uncover dependencies between field pairs by analyzing their covariance with respect to subsets of other variables. An alternative, Bayesian approach is taken by Cooper and Herskovits [1991] in deriving the most likely dependency graph for a set of data.

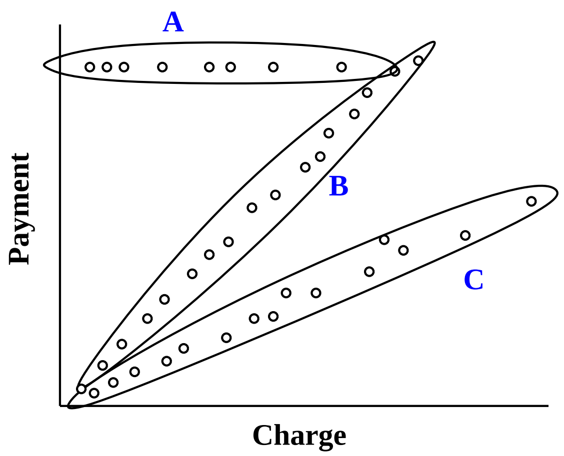
Data dependencies have numerous applications. They are used for database normalization and design, and for query optimization. Dependency graphs are often the objective of social, economic, and psychological studies. They have also been used to look for exceptions in data [Schlimmer, 1991] and for minimizing the number of fields a decision tree requires [Almuallim and Dietterich, 1991].

In a discovery system, the results of dependency analysis may sometimes be of direct interest to the end-user, for example by revealing unknown dependencies among fields. Often, however, strong dependencies reflect inherent domain structure rather than anything new or interesting. Automatically detecting dependencies can be a useful way of discovering this knowledge for use by other pattern-extraction algorithms. One such use is in explaining plausible causes for discovered changes: a change in a field can be traced through a dependency network to find changes in other variables which may explain the observed change.

3.5.2. Class Identification

Records can be grouped into meaningful subclasses. The identification of such classes may be of direct interest to the user, or they may provide useful information for other extraction algorithms. For example, a class might serve as the target concept required by a supervised-learning algorithm, e.g. a decision-tree inducer. Similarly, in deviation detection (section 3.5.4), subclasses can be used as the basis against which deviations are judged.

Classes can come from several sources. Any existing field can be used to define a set of subclasses: the field `Region` might define subclasses of `North`, `South`, `East`, and `West`. Domain knowledge may contain additional classes defined by domain experts to describe



Linear Clusters along two dimensions:

A = fixed payment
B = payment equals charge
C = payment 50% of charge

Characteristic Description of A:

Insur. Carrier = Medicare
Nom. Length of Stay = 4

Discriminating Description of Classes A, B, and C:

If Insurance is Medicare, Class = A
else if Insur. Group is 8, Class = B
else Class = C

Figure 5: An example of clustering and description algorithms. The graph shows three linear clusters found in actual hospital admissions data. The clusters are defined by the relation between fields standing for payment and charges. A characteristic description is given for class A and a discriminating description is given for all classes.

more complex relations between fields and records. Alternatively, *clustering algorithms* can be used to discover classes automatically. An example of this appears in Figure 5 which shows three classes that were identified using a linear clustering algorithm in the Knowledge Discovery Workbench.

There are numerous clustering algorithms ranging from the traditional methods of pattern recognition [Duda and Hart, 1973] and mathematical taxonomy, [Dunn and Everitt, 1982] to the more recent conceptual clustering techniques developed in machine learning [Fisher *et al.*, 1991]. Although useful under the right conditions and with the proper biases, these methods do not always equal the human ability to identify useful clusters, especially when dimensionality is low and visualization is possible. This has prompted the development of interactive clustering algorithms that combine the computer's computational powers with a human's knowledge and visual skills [Smith *et al.*, 1990].

3.5.3. Concept Description

Users may sometimes be interested in the individual records in a class, but more typically they want an abstract or intentional description that summarizes interesting qualities about the class. There are two broad types of intentional descriptions: characteristic and discriminating. A characteristic description describes what the records in a class share in common among themselves. A discriminating description describes how two or more classes differ. Examples of these two types of concept descriptions are shown in Figure 5.

Characteristic descriptions represent patterns in the data that best describe or summarizes one class regardless of the characteristics of other classes. Locating these descriptions involves identifying commonalities among records of the same class. Typical examples of characteristic algorithms can be found in LCHR [Cai *et al.*, 1990], the summary module of the KDW [Piatetsky-Shapiro and Matheus, 1991], and the Char operator in INLEN [Kaufman *et al.*, 1991].

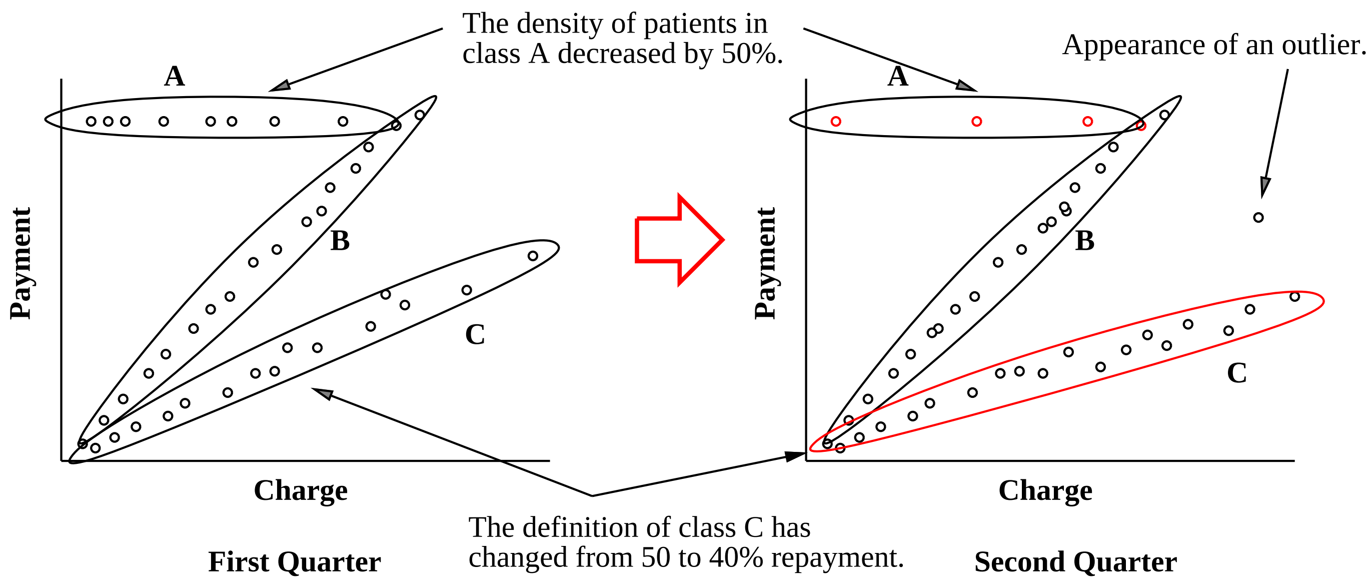


Figure 6: An example of deviations or changes between two database snapshots: the population of Class A has significantly decreased; the description of Class C has changed in terms of the slope of its line; an outlier has also appeared.

Langley [1987] provides a general theory of learning discriminating descriptions. The general approach involves a systematic search for minimal descriptions that can distinguish between members of different classes. Often these descriptions take the form of production rules (see [Agrawal *et al.*, 1993] in this issue), decision trees, or decision lists. Many empirical learning algorithms, such as decision-tree inducers [Quinlan, 1986], neural networks [Rummelhart and McClelland, 1986], and genetic algorithms [Holland *et al.*, 1986] are designed to produce discriminating descriptions. This class of algorithms is central to much of the research on KDD systems.

3.5.4. Deviation Detection

A general heuristic for finding interesting patterns is to look for deviations, particularly extremes [Lenat, 1977]. Deviations cover a wide variety of potentially interesting patterns: anomalous instances that do not fit into standard classes; outliers that appear at the fringe of other patterns; classes that exhibit average values significantly different from their parent or sibling classes; changes in a value or set of values from one time period to the next; discrepancies between observed values and expected values predicted by a model. Examples of some of these deviations are depicted in Figure 6

The common denominator among the methods for finding these types of patterns is the search for significant differences between an observation and a reference. The observation is usually a field value or a summarization of one or more field values, taken either across individual records or across sets of records. The reference might be another observation (such as when one quarter's observation is compared with another quarter's), a value provided by some outside domain knowledge (e.g. a national norm), or a value calculated by a model applied to the data (e.g., the result of a linear regression).

A fundamental feature of deviation analysis is that it can effectively filter out a large number of patterns that are less likely to be interesting. If we consider the reference to be a representation of the expectations of the user, then a pattern is potentially interesting in so far as the observation deviates from the expectations. The major challenge with this approach is determining when a deviation is "significant" enough to be of interest. A system can rank all observed deviations according to their magnitudes, and then leave it to the user to decide where to draw the line. The significance of a deviation, however, often involves more than just a statistical measure of variance. Users typically employ information outside the database to judge the interestingness of a pattern, and this information may change over time. Incorporating this sort of information into the system usually requires knowledge extraction from the user. Alternatively, this information might be discoverable through observation of the user's rankings of patterns over time.

3.6. Evaluation

Databases are replete with patterns, but few of them are of much interest. A pattern is *interesting* to the degree that it is accurate, novel, and useful with respect to the end-user's knowledge and objectives (see [Frawley *et al.*, 1991] for a more detailed discussion of what makes a pattern interesting). The evaluation component of our model determines the relative degree of interest of extracted patterns, and decides which to present and in what order. In actual systems, the evaluation component is often subsumed by the pattern-extraction algorithm(s) which are assumed to produce only significant results.

Statistical significance is usually a key factor in determining interestingness. If a pattern cannot be shown to be valid with some degree of certainty, it is not of much use, and thus not very interesting. For some patterns, the percentage of coverage or degree of accuracy may provide a sufficient measure. If patterns are based on samples from the database, a confidence measure may also be required to state how likely the patterns are to hold over the entire population. Such information is also desirable when the patterns are to be used to make predictions on data outside the database, e.g. identifying potential new customers based on profiles from an existing customer database.

Statistical significance alone is often insufficient to determine a pattern's degree of interest. A "five percent increase in sales of widgets in the western region," for example, could be more interesting than a "50% increase in the eastern region." In this case it could be that the western region has a much large sales volume, and thus its increase translates into greater income growth. Here the "impact" of the first pattern increases its relative interest. The specific factors that influence the impact and interestingness of a pattern will vary for different databases and tasks, thus requiring outside domain knowledge. Pattern templates (see EXPLORA in section 4.2) also use domain knowledge for a form of evaluation by ensuring the generation of useful results, or by serving to filter out less-desirable patterns.

4. Discovery Systems

We now consider three systems that perform knowledge discovery on databases: CoverStory, EXPLORA, and the Knowledge Discovery Workbench. These systems were selected because they exhibit some of the capabilities and limitations of current technology. In the following sections we briefly describe each system, compare them to the model, and discuss how they tradeoff autonomy versus versatility.

4.1. CoverStory

CoverStory^{tm} is a commercial system developed by Information Resources, Inc. in cooperation with John D.C. Little of MIT Sloan School of Management [Schmitz *et al.*, 1990]. Its purpose is to analyze supermarket scanner data and produce a marketing report summarizing the "key events." The developers of CoverStory interviewed market analysts and uncovered their interest in tracking changes in regional sales across aggregate product lines, particular product components, and competitors' products. This task involves identifying and ranking significant changes across time and geographic regions, and providing plausible explanations for the changes. For this latter process, the developers used model-building experience to select the marketing variables within the data that most strongly influence sales volume; these were store displays, feature ads, distribution, price cuts, and price. The relative impact of each of these factors was determined from marketing research data.

CoverStory performs a top-down analysis of the raw scanner data beginning with aggregate products lines, decomposing them into product components, and finally comparing these to competitor products. At each stage the system ranks the products according to volume change, selects the top few to report, and identifies the causal factors having the highest scores as defined by the equation:

$$\text{Score} = \text{Percent-Change} * \text{Factor-Weight} * \text{Market-Weight}$$

The Percent-Change here is the percent of change in product volume, Factor-Weight is a constant weight indicating the relative importance of a marketing factor (derived from the original marketing analysis), and Market-Weight takes into account the market size (heuristically set to the square root of market size). After analyzing all market changes, CoverStory produces a report using natural-language templates to generate text such as:

> "Sizzle's share in Total United States was 71.3 in the C&B Juice/Drink category for the twelve weeks ending 5/21/89. This is an increase of 1.2 points from a year earlier but down .5 points from last period. ...
>
> Sizzle 64oz's share increase is partly due to 11.3 pts rise in % ACV with Display versus a year ago. ..."

Commercially, CoveryStory has proven successful – more than a dozen systems had been installed by 1993. Much of this success lies in its focus on a particular, well-defined need and in its presentation of results in a very usable, human-oriented form. SPOTLIGHTS, a similar system for filtering and analyzing gigabytes of data from packaged goods scanners, has since been introduced by A.C. Nielson [Anand and Kahn, 1992].

Comparison to the Model:

All of the components of our model KDD system are exhibited in CoverStory. The system gets its data directly from a scanner database. Its domain knowledge is built into the linear model of causal relationships and the top-down analysis algorithm. The system's controller follows the simple four-step algorithm outlined above. The causal model provides focus by identifying a small set of relevant features. The extraction algorithm falls into the class of deviation-detection methods: it identifies where the data deviates most from previous periods, other regions, or competitors performance, and then attempts to explain the deviations by identifying the factors that most strongly influence the results. Evaluation of the results relies on the ranking of changes according to the percent of change and the causal factor scores.

CoverStory is fully automated once the initial domain knowledge for a particular distributor has been entered. In turn, it is fairly limited in its applicability, being tied closely to the scanner data and marketing models. Its high degree of autonomy and relatively low versatility place CoverStory near the right extreme of the tradeoff curve in Figure 3.

4.2. EXPLORA

EXPLORA, developed by Hoschka and Klosgen [1991], is "an integrated system for conceptually analyzing data and searching for interesting relationships." It operates by performing a graph search through a network of pattern templates (also called *statement types*) searching for "facts." A fact is a data instantiation of a pattern template that satisfies statistical criteria specified in an associated *verification method. Redundancy rules* use taxonomic information to reduce the search, and *generalization* and *selection criteria* condense the resulting set of discovered facts. Using the interactive browser, an end-user can take the ordered set of facts and generate a customized, final report. The user may also intervene throughout the discovery process to create new statement types, modify verification methods, and generally guide the search path.

The pattern templates, or *statement types*, assume three forms: *rule searcher*, *change detector*, and *trend detector*. The rule searcher type describes patterns between subpopulations based on the following template:

> *Target group* **shows** *outstanding behavior* **within** *population* **for** *subpopulation* **in** *year*.

An instantiation of this type might look like:

> *Persons having initiated an insurance policy at company A* are *highly over-represented* within the *clients of A* for *high-income people in the South* in *1987*.

Change-detector and trend-detector statement types are similar, except they support slots for time periods instead of population groups, and the values for the "outstanding behavior" slot differ. For a specific application, taxonomies of objects for each template slot (i.e. the target groups, outstanding behaviors, populations, time periods, and time ranges) must be provided as domain knowledge. These taxonomies define the search space in which EXPLORA looks for facts.

A statistical *verification method* is associated with each statement type, and is used to determine when a statement instantiation constitutes a fact within the data. The standard verification method for the rule searcher type uses the statistical measure $q = (p - p_0)/s$, where p is the percentage of *target group* in *population*, p_0 is the percentage of *subpopulation* in *population*, and s is the estimated standard deviation. The value of the measure, q, is then compared to a threshold to select and reject instantiations as facts. During interactive discover, the user has the option to adjust verification methods to fine tune the results.

After the system completes its search through the network of patterns, *inductive generalization rules* are applied to the set of discovered facts to integrate related statements. One form of generalization, for example, collects patterns with similar observations; another identifies regularities among similar statements across time periods. Application-dependent *selection rules* provide a final filter of the facts before the user turns them into a final report using the browser tool.

Comparison to the Model:

It is unclear from the literature whether EXPLORA has a direct DBMS interface. Its knowledge base is well defined, comprising generic statement types, verification methods, filtering rules, and applications-specific object taxonomies. EXPLORA's controller combines human guidance with a heuristic search through the space of instantiated statement types. Focus is provided by the instantiations of statement types which specify the fields to access from specific subsets of records. EXPLORA's extraction algorithm is fundamentally a deviation detector that identifies significant differences between populations or across time periods. Evaluation is based primarily upon statistical measures with additional, application-specific constraints optionally provided by the user.

EXPLORA is specifically designed to work with data that changes "regularly and often." Within this context, the statement types are generic enough for most deviation-detection problems, but they need to be supplemented with object taxonomies specific to each application. Once these are defined, the search algorithm can be turned loose to operate autonomously, although human guidance is required to fine tune the results for the best performance. This combination of moderate autonomy with moderate versatility places EXPLORA roughly in the middle of the tradeoff curve in Figure 3.

4.3. Knowledge Discovery Workbench

The Knowledge Discovery Workbench (KDW) is a collection of tools for the interactive analysis of large databases [Piatetsky-Shapiro and Matheus, 1991]. Its components have evolved through three versions (KDW, KDW II, and KDW++) all of which provide a graphical user interface to a suite of tools for accessing database tables, creating new fields, defining a focus, plotting data and results, applying discovery algorithms, and handling domain knowledge. The current version of the system is embedded with an extensible command interpreter based on *tcl* [Ousterhout, 1990] which enables the user to interactively control the discovery process or call up intelligent scripts to automate discovery tasks. The following extraction algorithms have been incorporated into one or more versions of the KDW: *clustering* for identifying simple linearly-related classes (see Figure 5); *classification* for finding rules using a decision-tree algorithm; *summarization* for characterizing classes of records; *deviation detection* for identifying significant differences between classes of records; *dependency analysis* for finding and displaying probabilistic dependencies . The details of most of these algorithms can be found in [Piatetsky-Shapiro, 1991a], [Piatetsky-Shapiro and Matheus, 1991], and [Piatetsky-Shapiro, 1992a]. InLen, a system developed by [Kaufman *et al.*, 1991], has a similar design to that of the KDW.

The KDW itself is intended to be versatile and domain independent. As such, it requires considerable guidance from the user who must decide what data to access, how to focus the analysis, which discovery algorithms to apply, and how to evaluate and interpret the results. This "workbench" design is ideal for exploratory analysis by a user knowledgable in both the data and the operation of the discovery tools. We are, however, also interested in making knowledge discovery more accessible to less skilled users through the development of customized applications. These efforts require extensive interaction and exploration of the data with the end user to identify what tools are appropriate and what domain knowledge is needed. The KDW is serving as an invaluable tool for this knowledge-engineering process, assisting in the exploration of the data, the building of models, and identification of structure in the database. The command interpreter built into the KDW also facilitates system development by making it possible to quickly write scripts that appropriately combine the tools needed to perform a sequence of analysis.

Comparison to the Model:

The KDW has direct access to a DBMS through its SQL-based query interface. Its knowledge base contains information specific to a database regarding important field groups, record groups, functional dependencies, and SQL-query statements. Most of this domain knowledge is used to provide focus by guiding the access of information from the database. Control in the KDW is provided exclusively by the user, who may define scripts to automate frequently repeated operations. The pattern-extraction algorithms range from clustering to classification to deviation detection. Each of these provides significance measures for their results, although final evaluation is left to the user.

Heavy reliance on the user places the KDW at the lower end of the autonomy scale. In turn, its range of tools and generic applicability rates it high on versatility. Together these traits put the KDW at the upper left-hand extreme of the tradeoff curve in Figure 3.

5. Conclusions

A KDD system is a collection of components that enables the complete process of knowledge discovery, from the accessing of data in a DBMS, to the focusing and application of pattern-extraction algorithms, to the evaluation and interpretation of results. An ideal system would handle all of these autonomously while being applicable across many domains. The systems we have considered are far from this ideal, being constrained by the versatility/autonomy tradeoff depicted in Figure 3. We are thus led to ask, what will it take to push these types of systems closer to the ideal?

We have argued that autonomy requires domain knowledge, whereas versatility implies domain independence. Although these two seem irreconcilable, there is much that can be done to improve the situation. First, some domain knowledge can be extracted from databases automatically. Data dependencies are a good example of the type of structure that can be identified and used to guide further analysis. In this single area alone we need 1) better methods for performing dependency analysis across all types of data, 2) tools for presenting the results for evaluation and modification by the user, and 3) discovery algorithms that can make fuller use of dependency networks. Second, we need better methods for extracting knowledge from the user, regardless of the database domain. This will require powerful interactive tools for systematically gathering knowledge from users. It will also likely require generic representations of knowledge for specific task in the discovery process, such as focusing and evaluation. A fuller analysis of the uses and representational requirements of domain knowledge within KDD would be a valuable study. Third, systems that will work in multiple domains will need to learn from their experiences working with the data and the user, which also means they will have to be able to store and reuse their own discoveries. This calls for a common representation of domain knowledge and discovered knowledge, placing even greater demands on a system's representational expressiveness. In short, we need improved methods for representing, acquiring, and using domain knowledge within KDD.

Improvements can also be made in the pattern-extraction algorithms. The growing size of databases begs for more efficient algorithms that can analyze larger portions of data. Faster processors and larger memories may help existing algorithms, but they will likely yield to new distributed discovery algorithms as parallel processing proliferates. New ways of combining domain knowledge with empirical techniques will also be important. The deviation-detection methods used by several existing systems, for example, can be significantly enhanced by going beyond the empirical patterns and attempting to explain observed differences based on knowledge of the structural dependencies of the data. This type of technique will become increasingly important as users of KDD systems begin to ask not only what the patterns are, but also why they are occurring in the data.

While our idealized KDD system is years away, interest in KDD is growing and research efforts are intensifying – the collection of papers in this issue are indicative of this direction. With the world's data continuing to grow exponentially, discovery systems may soon become the only viable solution to understanding what it all means.

Acknowledgments:

We would like to thank Bud Frawley, Jan Zytkow, and the journal referees for their helpful comments and suggestions on early versions of this paper. We are also indebted to Shri Goyal for his support and encouragement.

References

[Agrawal *et al.*, 1993] Rakesh Agrawal, Tomasz Imielinksi, and Arun Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, to appear, 1993.

[Almuallim and Dietterich, 1991] H. Almuallim and T. G. Dietterich. Learning with many irrelevant features. In *Proc. AAAI 91*, pages 547–552, 1991.

[Anand and Kahn, 1992] T. Anand and G. Kahn. SPOTLIGHT: A data explanation system. In *Proc. Eighth IEEE Conf. Appl. AI*, 1992.

[Buntine, 1991] Wray Buntine. Stratifying samples to improve learning. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 305–314. AAAI/MIT Press, Cambridge, MA, 1991.

[Cai *et al.*, 1990] Y. Cai, N. Cercone, and J. Han. Learning characteristic rules from relational databases. In Gardin and G. Mauri, editors, *Computational Intelligence II*, pages 187–196. Elsevier, New York, NY, 1990.

[Cooper and Herskovits, 1991] G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. Technical Report KSL-91-02, Knowledge Systems Laboratory, Standford University, Stanford, CA, 1991.

[Date, 1977] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley, Reading, MA, 1977.

[Dhar and Tuzhilin, 1993] Vasant Dhar and Alexander Tuzhilin. Abstract-driven pattern discovery in databases. *IEEE Transactions on Knowledge and Data Engineering*, to appear, 1993.

[Dixon and Massey, 1979] W. J. Dixon and F. J. Massey. *Introduction to Statistical Analysis.* McGraw-Hill, 1979.

[Duda and Hart, 1973] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, 1973.

[Dunn and Everitt, 1982] G. Dunn and B. S. Everitt. *An Introduction to Mathematical Taxonomy.* Cambridge University Press, Cambridge, MA, 1982.

[Dzeroski and Lavrac, 1993] Saso Dzeroski and Nada Lavrac. Inductive learning in deductive databases. *IEEE Transactions on Knowledge and Data Engineering*, to appear, 1993.

[Fisher *et al.*, 1991] Doug Fisher, Michael Pazzani, and Pat Langley, editors. *Concept Formation: Knowledge and Experience in Unsupervised Learning.* Morgan Kaufmann Publishers, Inc., 1991.

[Frawley *et al.*, 1991] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. In *Knowledge Discovery in Databases*, pages 1–27. AAAI/MIT Press, Cambridge, MA, 1991. Reprinted in *AI Magazine, Vol. 13, No. 3, 1992.*

[Glymour *et al.*, 1987] C. Glymour, R. Scheines, P. Spirtes, and K. Kelly. *Discovering Causal Structure.* Academic Press, 1987.

[Han *et al.*, 1993] Jiawei Han, Yue Hwang, and Nick Cercone. Intelligent query answering using discvoered knowledge. *IEEE Transactions on Knowledge and Data Engineering*, to appear, 1993.

[Holder and Cook, 1993] Lawrence B. Holder and Diane J. Cook. Discovery of inexact concepts from structural data. *IEEE Transactions on Knowledge and Data Engineering*, to appear, 1993.

[Holland *et al.*, 1986] John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge, MA, 1986.

[Hoschka and Klosgen, 1991] P. Hoschka and W. Klosgen. A support system for interpreting statistical data. In G. Piatetsky-Shapiro and W. Frawley, editors, *Knowledge Discovery in Databases*, chapter 19, pages 325–345. AAAI/MIT Press, Cambridge, MA, 1991.

[Kaufman *et al.*, 1991] Kenneth A. Kaufman, Ryszard S. Michalski, and Larry Kerschberg. Mining for knowledge in databases: Goals and general description of the INLEN system. In *Knowledge Discovery in Databases*, chapter 26. AAAI/MIT Press, Cambridge, MA, 1991.

[Klosgen, 1991] Willi Klosgen. Visualization and adaptivity in the statistics interpreter EXPLORA. In *Workshop Notes from the Ninth National Conference on Artificial Intelligence: Knowledge Discovery in Databases*, pages 25–34, Anaheim, CA, July 1991. American Association for Artificial Intelligence.

[Langley, 1987] P. Langley. A general theory of discrimination learning. In D. Klahr, P. Langley, and R. Neches, editors, *Production System Models of Learning and Development*, pages 99–161. MIT Press, Cambridge, MA, 1987.

[Lenat, 1977] D. B. Lenat. On automatic scientific theory formation: A case study using the AM program. In *Machine Intelligence, 9*, pages 251–286. Halsted Press, New York, 1977.

[Mannila and Raiha, 1987] H. Mannila and K.-J. Raiha. Dependency inference. In *Proceedings of the Thirteenth International Conference on Very Large Data Bases (VLDB'87)*, pages 155–158, 1987.

[Michalski *et al.*, 1983] Ryszard S. Michalski, Jaime G. Carbonell, and Thomas M. Mitchell. *Machine Learning: An Artificial Intelligence Approach.* Tioga Press, Palo Alto, 1983.

[Ousterhout, 1990] John K. Ousterhout. TCL: An embeddable command language. In *Proceedings of the 1990 Winter USENIX Conference*, pages 133–146, 1990.

[Pearl and Verma, 1991] J. Pearl and T. S. Verma. A theory of inferred causation. In *Proceedings of Second Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 441–452, San Mateo, CA, 1991. Morgan Kaufmann.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[Piatetsky-Shapiro and Frawley, 1991] G. Piatetsky-Shapiro and W. J. Frawley, editors. *Knowledge Discovery in Databases.* AAAI/MIT Press, Cambridge, MA, 1991.

[Piatetsky-Shapiro and Matheus, 1991] Gregory Piatetsky-Shapiro and Christopher J. Matheus. Knowledge Discovery Workbench: An exploratory environment for discovery in business databases. In *Workshop Notes from the Ninth National Conference on Artificial Intelligence: Knowledge Discovery in Databases*, pages 11–24, Anaheim, CA, July 1991.

[Piatetsky-Shapiro, 1991a] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–248. AAAI/MIT Press, Cambridge, MA, 1991.

[Piatetsky-Shapiro, 1991b] Gregory Piatetsky-Shapiro, editor. *Workshop Notes from the Ninth National Conference on Artificial Intelligence: Knowledge Discovery in Databases*, Anaheim, CA, July 1991.

[Piatetsky-Shapiro, 1992a] G. Piatetsky-Shapiro. Probabilistic data dependencies. In *Proc. Mach. Discovery Work. (Ninth Mach. Learn. Conf.)*, Aberdeen, Scotland, 1992. (to appear).

[Piatetsky-Shapiro, 1992b] Gregory Piatetsky-Shapiro, editor. Special issue on: Knowledge Discovery in Data- and Knowledge Bases, *International Journal of Intelligent Systems, 7(7)*, 1992.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[Quinlan, 1989a] J. Ross Quinlan. Learning relations: Comparison of a symbolic and a connectionist approach. Technical Report TR-346, Basser Department of Computer Science, University of Sydney, Sydney, Australia, May 1989.

[Quinlan, 1989b] J.R. Quinlan. Unknown attribute values in induction. In A. M. Segre, editor, *Proceedings of the Sixth International Machine Learning Workshop*, pages 164–168. Morgan Kaufmann Publishers, June 1989.

[Roth and Mattis, 1991] Stevn F. Roth and Joe Mattis. Automating the presentation of information. In *IEEE Conference on Artificial Intelligence Applications*, Miami Beech, FL, 1991.

[Rummelhart and McClelland, 1986] Donald E. Rummelhart and Jay L. McClelland. *Parallel Distributed Processing, Vol. 1.* MIT Press, Cambridge, MA, 1986.

[Scheines and Spirtes, 1992] R. Scheines and P. Spirtes. Finding latent variable models in large data bases. *International Journal of Intelligent Systems*, 1992. forthcoming.

[Schlimmer, 1991] J. Schlimmer. Learning determinations and checking databases. In *Proc. Knowledge Discovery in Databases (AAAI 91)*, pages 64–76, 1991.

[Schmitz *et al.*, 1990] J. Schmitz, G. Armstrong, and J. D. C. Little. CoverStory – automated news finding in marketing. In *DSS Transactions*, pages 46–54, Providence, RI., 1990. Institute of Management Sciences.

[Shekhar *et al.*, 1993] Shashi Shekhar, Babak Hamidzadeh, Ashim Kohli, and Mark Coyle. Learning transformation rules for semantic query optimization: A data-driven approach. *IEEE Transactions on Knowledge and Data Engineering*, to appear, 1993.

[Smith *et al.*, 1990] S. Smith, D. Bergeron, and G. Grinstein. Stereophonic and surface sound generation for exploratory data analysis. In *Conference of the Special Interest Group in Computer and Human Interaction*, Seattle WA, April 1990.

[Stonebraker, 1985] M. Stonebraker. Triggers and inference in data base systems. In *Proc. Islamoora Conference on Expert Data Bases*, 1985.

[Tufte, 1983] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, 1983.

[Ullman, 1982] J. D. Ullman. *Principles of Database Systems.* Computer Science Press, Rockville, MD, 1982.

[Zytkow and Baker, 1991] Jan M. Zytkow and John Baker. Interactive mining of regularities in databases. In *Knowledge Discovery in Databases.* AAAI/MIT Press, Cambridge, MA, 1991.