

DBMiner: A System for Mining Knowledge in Large Relational Databases*

Jiawei Han Yongjian Fu Wei Wang Jenny Chiang Wan Gong Krzysztof Koperski Deyi Li
Yijun Lu Amynmohamed Rajan Nebojsa Stefanovic Betty Xia Osmar R. Zaiane

Data Mining Research Group, Database Systems Research Laboratory
School of Computing Science, Simon Fraser University, British Columbia, Canada V5A 1S6
E-mail: {han, yongjian, weiw, ychiang, wgong, koperski, dli, yijunl, arajan, nstefano, bxia, zaiane}@cs.sfu.ca
URL: <http://db.cs.sfu.ca/> (for research group) <http://db.cs.sfu.ca/DBMiner> (for system)

Abstract

A data mining system, DBMiner, has been developed for interactive mining of multiple-level knowledge in large relational databases. The system implements a wide spectrum of data mining functions, including generalization, characterization, association, classification, and prediction. By incorporating several interesting data mining techniques, including attribute-oriented induction, statistical analysis, progressive deepening for mining multiple-level knowledge, and meta-rule guided mining, the system provides a user-friendly, interactive data mining environment with good performance.

Introduction

With the upsurge of research and development activities on knowledge discovery in databases (Piatetsky-Shapiro & Frawley 1991; Fayyad *et al.* 1996), a data mining system, DBMiner, has been developed based on our studies of data mining techniques, and our experience in the development of an early system prototype, DBLearn. The system integrates data mining techniques with database technologies, and discovers various kinds of knowledge at multiple concept levels from large relational databases efficiently and effectively.

The system has the following distinct features:

1. It incorporates several interesting data mining techniques, including attribute-oriented induction (Han, Cai, & Cercone 1993; Han & Fu 1996), statistical analysis, progressive deepening for mining multiple-level rules (Han & Fu 1995; 1996), and meta-rule guided knowledge mining (Fu & Han 1995). It also implements a wide spectrum of data mining functions including generalization, characterization, association, classification, and prediction.

2. It performs interactive data mining at multiple concept levels on any user-specified set of data in a database using an SQL-like Data Mining Query Language, DMQL, or a graphical user interface. Users may interactively set and adjust various thresholds, control a data mining process, perform *roll-up* or *drill-down* at multiple concept levels, and generate different forms of outputs, including generalized relations, generalized feature tables, multiple forms of generalized rules, visual presentation of rules, charts, curves, etc.
3. Efficient implementation techniques have been explored using different data structures, including generalized relations and multiple-dimensional data cubes. The implementations have been integrated smoothly with relational database systems.
4. The data mining process may utilize user- or expert-defined set-grouping or schema-level concept hierarchies which can be specified flexibly, adjusted dynamically based on data distribution, and generated automatically for numerical attributes. Concept hierarchies are being taken as an integrated component of the system and are stored as a relation in the database.
5. Both UNIX and PC (Windows/NT) versions of the system adopt a client/server architecture. The latter may communicate with various commercial database systems for data mining using the ODBC technology.

The system has been tested on several large relational databases, including NSERC (Natural Science and Engineering Research Council of Canada) research grant information system, with satisfactory performance. Additional data mining functionalities are being designed and will be added incrementally to the system along with the progress of our research.

Architecture and Functionalities

The general architecture of DBMiner, shown in Figure 1, tightly integrates a relational database system, such as a Sybase SQL server, with a concept hierarchy module, and a set of knowledge discovery modules. The discovery modules of DBMiner, shown in Fig-

*Research was supported in part by the grant NSERC-OPG003723 from the Natural Sciences and Engineering Research Council of Canada, the grant NCE:IRIS/Precarn-HMI-5 from the Networks of Centres of Excellence of Canada, and grants from B.C. Advanced Systems Institute, MPR Teltech Ltd., and Hughes Research Laboratories.

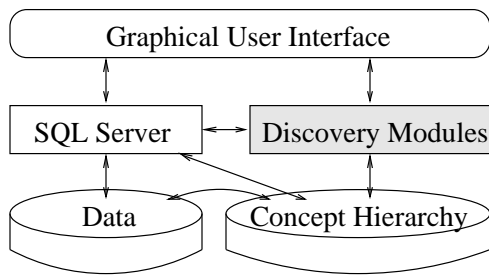


Figure 1: General architecture of DBMiner

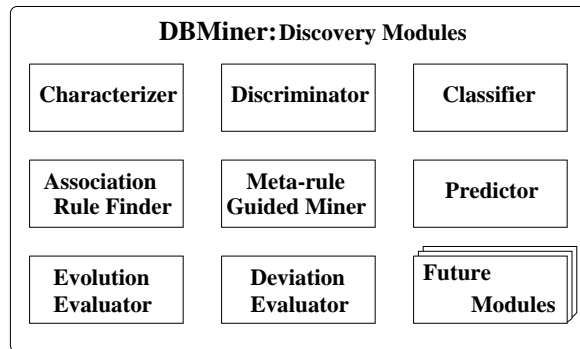


Figure 2: Knowledge discovery modules of DBMiner

Figure 2, include characterizer, discriminator, classifier, association rule finder, meta-rule guided miner, predictor, evolution evaluator, deviation evaluator, and some planned future modules.

The functionalities of the knowledge discovery modules are briefly described as follows:

- The **characterizer** generalizes a set of task-relevant data into a *generalized relation* which can then be used for extraction of different kinds of rules or be viewed at multiple concept levels from different angles. In particular, it derives a set of characteristic rules which summarizes the general characteristics of a set of user-specified data (called the *target class*). For example, the symptoms of a specific disease can be summarized by a characteristic rule.
- A **discriminator** discovers a set of **discriminant rules** which summarize the features that distinguish the class being examined (the *target class*) from other classes (called *contrasting classes*). For example, to distinguish one disease from others, a discriminant rule summarizes the symptoms that discriminate this disease from others.
- A **classifier** analyzes a set of training data (i.e., a set of objects whose class label is known) and constructs a model for each class based on the features in the data. A set of **classification rules** is generated by such a classification process, which can be used to classify future data and develop a better understanding of

each class in the database. For example, one may classify diseases and provide the symptoms which describe each class or subclass.

- An **association rule finder** discovers a set of association rules (in the form of " $A_1 \wedge \dots \wedge A_i \rightarrow B_1 \wedge \dots \wedge B_j$ ") at multiple concept levels from the relevant set(s) of data in a database. For example, one may discover a set of symptoms often occurring together with certain kinds of diseases and further study the reasons behind them.
- A **meta-rule guided miner** is a data mining mechanism which takes a user-specified meta-rule form, such as " $P(x, y) \wedge Q(y, z) \rightarrow R(x, z)$ " as a pattern to confine the search for desired rules. For example, one may specify the discovered rules to be in the form of " $major(s : student, x) \wedge P(s, y) \rightarrow gpa(s, z)$ " in order to find the relationships between a student's major and his/her gpa in a university database.
- A **predictor** predicts the possible values of some missing data or the value distribution of certain attributes in a set of objects. This involves finding the set of attributes relevant to the attribute of interest (by some statistical analysis) and predicting the value distribution based on the set of data similar to the selected object(s). For example, an employee's potential salary can be predicted based on the salary distribution of similar employees in the company.
- A **data evolution evaluator** evaluates the data evolution regularities for certain objects whose behavior changes over time. This may include characterization, classification, association, or clustering of time-related data. For example, one may find the general characteristics of the companies whose stock price has gone up over 20% last year or evaluate the trend or particular growth patterns of certain stocks.
- A **deviation evaluator** evaluates the deviation patterns for a set of task-relevant data in the database. For example, one may discover and evaluate a set of stocks whose behavior deviates from the trend of the majority of stocks during a certain period of time.

Another important function module of DBMiner is concept hierarchy which provides essential background knowledge for data generalization and multiple-level data mining. Concept hierarchies can be specified based on the relationships among database attributes (called *schema-level hierarchy*) or by set groupings (called *set-grouping hierarchy*) and be stored in the form of relations in the same database. Moreover, they can be adjusted dynamically based on the distribution of the set of data relevant to the data mining task. Also, hierarchies for numerical attributes can be constructed automatically based on data distribution analysis (Han & Fu 1994).

DMQL and Interactive Data Mining

DBMiner offers both an SQL-like data mining query language, DMQL, and a graphical user interface for interactive mining of multiple-level knowledge.

Example 1. To characterize CS grants in the *NSERC96* database related to discipline code and amount category in terms of count% and amount%, the query is expressed in DMQL as follows,

```
use NSERC96
find characteristic rules for "CS_Discipline_Grants"
from award A, grant_type G
related to disc_code, amount, count(*), amount(*)%
where A.grant_code = G.grant_code
and A.disc_code = "Computer Science"
```

The query is processed as follows: The system collects the relevant set of data by processing a transformed relational query, generalizes the data by *attribute-oriented induction*, and then presents the outputs in different forms, including generalized relations, generalized feature tables, multiple (including visual) forms of generalized rules, pie/bar charts, curves, etc.

A user may interactively set and adjust various kinds of thresholds to control the data mining process. For example, one may adjust the generalization threshold for an attribute to allow more or less distinct values in this attribute. A user may also *roll-up* or *drill-down* the generalized data at multiple concept levels. □

A data mining query language such as DMQL facilitates the standardization of data mining functions, systematic development of data mining systems, and integration with standard relational database systems. Various kinds of graphical user interfaces can be developed based on such a data mining query language. Such interfaces have been implemented in DBMiner on three platforms: Windows/NT, UNIX, and Netscape. A graphical user interface facilitates interactive specification and modification of data mining queries, concept hierarchies, and various kinds of thresholds, selection and change of output forms, roll-up or drill-down, and dynamic control of a data mining process.

Implementation of DBMiner

Data structures: Generalized relation vs. multi-dimensional data cube

Data generalization is a core function of DBMiner. Two data structures, *generalized relation*, and *multi-dimensional data cube*, can be considered in the implementation of data generalization.

A *generalized relation* is a relation which consists of a set of (generalized) attributes (storing generalized values of the corresponding attributes in the original relation) and a set of "aggregate" (*measure*) attributes (storing the values resulted from executing aggregate functions, such as *count*, *sum*, etc.), and in which each

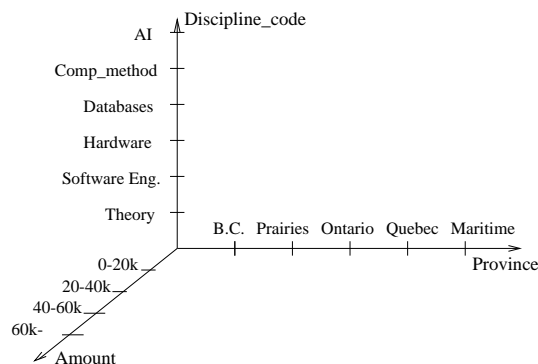


Figure 3: A multi-dimensional data cube

tuple is the result of generalization of a set of tuples in the original data relation. For example, a generalized relation *award* may store a set of tuples, such as "*award(AI, 20_40k, 37, 835900)*", which represents the generalized data for discipline code is "AI", the amount category is "20_40k", and such kind of data takes 37 in count and \$835,900 in (total) amount.

A *multi-dimensional data cube* is a multi-dimensional array structure, as shown in Figure 3, in which each dimension represents a generalized attribute and each cell stores the value of some aggregate attribute, such as *count*, *sum*, etc. For example, a multi-dimensional data cube *award* may have two dimensions: "discipline code" and "amount category". The value "AI" in the "discipline code" dimension and "20-40k" in the "amount category" dimension locate the corresponding values in the two aggregate attributes, *count* and *sum*, in the cube. Then the values, *count%* and *amount%*, can be derived easily.

In comparison with the generalized relation structure, a multi-dimensional data cube structure has the following advantages: First, it may often save storage space since only the measurement attribute values need to be stored in the cube and the generalized (dimensional) attribute values will serve only as dimensional indices to the cube; second, it leads to fast access to particular cells (or slices) of the cube using indexing structures; third, it usually costs less to produce a cube than a generalized relation in the process of generalization since the right cell in the cube can be located easily. However, if a multi-dimensional data cube structure is quite sparse, the storage space of a cube is largely wasted, and the generalized relation structure should be adopted to save the overall storage space.

Both data structures have been explored in the DBMiner implementations: the generalized relation structure is adopted in version 1.0, and a multi-dimensional data cube structure in version 2.0. A more flexible implementation is to consider both structures, adopt the multi-dimensional data cube structure when the size of the data cube is reasonable, and switch to the

generalized relation structure (by dynamic allocation) otherwise (this can be estimated based on the number of dimensions being considered, and the attribute threshold of each dimension). Such an alternative will be considered in our future implementation.

Besides designing good data structures, efficient implementation of each discovery module has been explored, as discussed below.

Multiple-level characterization

Data characterization summarizes and characterizes a set of task-relevant data, usually based on generalization. For mining multiple-level knowledge, progressive deepening (*drill-down*) and progressive generalization (*roll-up*) techniques can be applied.

Progressive generalization starts with a conservative generalization process which first generalizes the data to slightly higher concept levels than the primitive data in the relation. Further generalizations can be performed on it progressively by selecting appropriate attributes for step-by-step generalization. Strong characteristic rules can be discovered at multiple abstraction levels by filtering (based on the corresponding thresholds at different levels of generalization) generalized tuples with weak support or weak confidence in the rule generation process.

Progressive deepening starts with a relatively high-level generalized relation, selectively and progressively specializes some of the generalized tuples or attributes to lower abstraction levels.

Conceptually, a top-down, progressive deepening process is preferable since it is natural to first find general data characteristics at a high concept level and then follow certain interesting paths to step down to specialized cases. However, from the implementation point of view, it is easier to perform generalization than specialization because generalization replaces low level tuples by high ones through ascension of a concept hierarchy. Since generalized tuples do not register the detailed original information, it is difficult to get such information back when specialization is required later.

Our technique which facilitates specializations on generalized relations is to save a “*minimally generalized relation/cube*” in the early stage of generalization. That is, each attribute in the relevant set of data is generalized to minimally generalized concepts (which can be done in one scan of the data relation) and then identical tuples in such a generalized relation/cube are merged together, which derives the minimally generalized relation. After that, both *progressive deepening* and *interactive up-and-down* can be performed with reasonable efficiency: If the data at the current abstraction level is to be further generalized, generalization can be performed directly on it; on the other hand, if it is to be specialized, the desired result can be derived by generalizing the minimally generalized rela-

tion/cube to appropriate level(s).

Discovery of discriminant rules

The *discriminator* of DBMiner finds a set of discriminant rules which distinguishes the general features of a target class from that of contrasting class(es) specified by a user. It is implemented as follows.

First, the set of relevant data in the database has been collected by query processing and is partitioned respectively into a *target* class and one or a set of *contrasting* class(es). Second, attribute-oriented induction is performed on the target class to extract a *prime target relation/cube*, where a *prime target relation* is a generalized relation in which each attribute contains no more than but close to the threshold value of the corresponding attribute. Then the concepts in the *contrasting* class(es) are generalized to the same level as those in the prime target relation/cube, forming the *prime contrasting relation/cube*. Finally, the information in these two classes is used to generate qualitative or quantitative discriminant rules.

Moreover, interactive drill-down and roll-up can be performed synchronously in both target class and contrasting class(es) in a similar way as that explained in the last subsection (characterization). These functions have been implemented in the discriminator.

Multiple-level association

Based on many studies on efficient mining of association rules (Agrawal & Srikant 1994; Srikant & Agrawal 1995; Han & Fu 1995), a multiple-level association rule finder has been implemented in DBMiner.

Different from mining association rules in transaction databases, a relational association rule miner may find two kinds of associations: *nested association* and *flat association*, as illustrated in the following example.

Example 2. Suppose the “course_taken” relation in a university database has the following schema:

course_taken = (*student_id*, *course*, *semester*, *grade*).

Nested association is the association between a data object and a set of attributes in a relation by viewing data in this set of attributes as a nested relation. For example, one may find the associations between students and their course performance by viewing “(*course*, *semester*, *grade*)” as a nested relation associated with *student_id*.

Flat association is the association among different attributes in a relation without viewing any attribute(s) as a nested relation. For example, one may find the relationships between *course* and *grade* in the *course_taken* relation such as “*the courses in computing science tend to have good grades*”, etc.

Two associations require different data mining techniques.

For mining nested associations, a data relation can be transformed into a nested relation in which the tuples which share the same values in the unnested attributes are merged into one. For example, the *course_taken* relation can be folded into a nested relation with the schema,

$$\begin{aligned} \text{course_taken} &= (\text{student_id}, \text{course_history}) \\ \text{course_history} &= (\text{course}, \text{semester}, \text{grade}). \end{aligned}$$

By such transformation, it is easy to derive association rules like “90% senior CS students tend to take at least three CS courses at 300-level or up in each semester”. Since the nested tuples (or values) can be viewed as data items in the same transaction, the methods for mining association rules in transaction databases, such as (Han & Fu 1995), can be applied to such transformed relations in relational databases.

Multi-dimensional data cube structure facilitates efficient mining of multi-level flat association rules. A count cell of a cube stores the number of occurrences of the corresponding multi-dimensional data values, whereas a dimension count cell stores the sum of counts in the whole dimension. With this structure, it is straightforward to calculate the measurements such as *support* and *confidence* of association rules. A set of such cubes, ranging from the least generalized cube to rather high level cubes, facilitate mining of association rules at multiple concept levels. \square

Meta-rule guided mining

Since there are many ways to derive association rules in relational databases, it is preferable to have users to specify some interesting constraints to guide a data mining process. Such constraints can be specified in a *meta-rule* (or *meta-pattern*) form (Shen *et al.* 1996), which confines the search to specific forms of rules. For example, a meta-rule “ $P(x, y) \rightarrow Q(x, y, z)$ ”, where P and Q are predicate variables matching different properties in a database, can be used as a rule-form constraint in the search.

In principle, a meta-rule can be used to guide the mining of many kinds of rules. Since the association rules are in the form similar to logic rules, we have first studied meta-rule guided mining of association rules in relational databases (Fu & Han 1995). Different from the study by (Shen *et al.* 1996) where a meta-predicate may match any relation predicates, deductive predicates, attributes, etc., we confine the search to those predicates corresponding to the attributes in one relation. One such example is illustrated as follows.

Example 3. A meta-rule guided data mining query can be specified in DMQL as follows for mining a specific form of rules related to a set of attributes: “*major, gpa, status, birth_place, address*” in relation *student* for those born in Canada in a *university* database.

find association rules in the form of

$$\text{major}(s : \text{student}, x) \wedge Q(s, y) \rightarrow R(s, z)$$

related to major, gpa, status, birth_place, address
 from student
 where birth_place = “Canada”

Multi-level association rules can be discovered in such a database, as illustrated below:

$$\begin{aligned} \text{major}(s, \text{“Science”}) \wedge \text{gpa}(s, \text{“Excellent”}) &\rightarrow \\ \text{status}(s, \text{“Graduate”}) &(60\%) \\ \text{major}(s, \text{“Physics”}) \wedge \text{status}(s, \text{“M.Sc”}) &\rightarrow \\ \text{gpa}(s, \text{“3.8-4.0”}) &(76\%) \end{aligned}$$

The mining of such multi-level rules can be implemented in a similar way as mining multiple-level association rules in a multi-dimensional data cube. \square

Classification

Data classification is to develop a description or model for each class in a database, based on the features present in a set of class-labeled training data.

There have been many data classification methods studied, including decision-tree methods, such as ID-3 and C4.5 (Quinlan 1993), statistical methods, neural networks, rough sets, etc. Recently, some database-oriented classification methods have also been investigated (Mehta, Agrawal, & Rissanen 1996).

Our classifier adopts a generalization-based decision-tree induction method which integrates attribute-oriented induction with a decision-tree induction technique, by first performing attribute-oriented induction on the set of training data to generalize attribute values in the training set, and then performing decision tree induction on the generalized data.

Since a generalized tuple comes from the generalization of a number of original tuples, the *count* information is associated with each generalized tuple and plays an important role in classification. To handle noise and exceptional data and facilitate statistical analysis, two thresholds, *classification threshold* and *exception threshold*, are introduced. The former helps justification of the classification at a node when a significant set of the examples belong to the same class; whereas the latter helps ignore a node in classification if it contains only a negligible number of examples.

There are several alternatives for doing generalization before classification: A data set can be generalized to either a minimally generalized concept level, an intermediate concept level, or a rather high concept level. Too low a concept level may result in scattered classes, bushy classification trees, and difficulty at concise semantic interpretation; whereas too high a level may result in the loss of classification accuracy.

Currently, we are testing several alternatives at integration of generalization and classification in databases, such as (1) generalize data to some medium concept levels; (2) generalize data to intermediate concept level(s), and then perform node merge and split

for better class representation and classification accuracy; and (3) perform multi-level classification and select a desired level by a comparison of the classification quality at different levels. Since all three classification processes are performed in relatively small, compressed, generalized relations, it is expected to result in efficient classification algorithms in large databases.

Prediction

A predictor predicts data values or value distributions on the attributes of interest based on similar groups of data in the database. For example, one may predict the amount of research grants that an applicant may receive based on the data about the similar groups of researchers.

The power of data prediction should be confined to the ranges of numerical data or the nominal data generalizable to only a small number of categories. It is unlikely to give reasonable prediction on one's name or social insurance number based on other persons' data.

For successful prediction, the factors (or attributes) which strongly influence the values of the attributes of interest should be identified first. This can be done by the analysis of data relevance or correlations by statistical methods, decision-tree classification techniques, or simply be based on expert judgement. To analyze attribute correlation, our predictor constructs a contingency table followed by association coefficient calculation based on χ^2 -test by the analysis of minimally generalized data in databases. The attribute correlation associated with each attribute of interest is precomputed and stored in a special relation in the database.

When a prediction query is submitted, the set of data relevant to the requested prediction is collected, where the relevance is based on the attribute correlations derived by the query-independent analysis. The set of data which matches or is close to the query condition can be viewed as similar group(s) of data. If this set is big enough (i.e., sufficient evidence exists), its value distribution on the attribute of interest can be taken as predicted value distribution. Otherwise, the set should be appropriately enlarged by generalization on less relevant attributes to certain high concept level to collect enough evidence for trustable prediction.

Further Development of DBMiner

The DBMiner system is currently being extended in several directions, as illustrated below.

- Further enhancement of the power and efficiency of data mining in relational database systems, including the improvement of system performance and rule discovery quality for the existing functional modules, and the development of techniques for mining new kinds of rules, especially on time-related data.

- Integration, maintenance and application of discovered knowledge, including incremental update of discovered rules, removal of redundant or less interesting rules, merging of discovered rules into a knowledge-base, intelligent query answering using discovered knowledge, and the construction of multiple layered databases.
- Extension of data mining technique towards advanced and/or special purpose database systems, including extended-relational, object-oriented, text, spatial, temporal, and heterogeneous databases. Currently, two such data mining systems, GeoMiner and WebMiner, for mining knowledge in spatial databases and the Internet information-base respectively, are being under design and construction.

References

- Agrawal, R., and Srikant, R. 1994. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, 487-499.
- Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R. 1996. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press.
- Fu, Y., and Han, J. 1995. Meta-rule-guided mining of association rules in relational databases. In *Proc. 1st Int'l Workshop on Integration of Knowledge Discovery with Deductive and Object-Oriented Databases*, 39-46.
- Han, J., and Fu, Y. 1994. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, 157-168.
- Han, J., and Fu, Y. 1995. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, 420-431.
- Han, J., and Fu, Y. 1996. Exploration of the power of attribute-oriented induction in data mining. In Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press. 399-421.
- Han, J.; Cai, Y.; and Cercone, N. 1993. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering* 5:29-40.
- Mehta, M.; Agrawal, R.; and Rissanen, J. 1996. SLIQ: A fast scalable classifier for data mining. In *Proc. 1996 Int. Conference on Extending Database Technology (EDBT'96)*.
- Piatetsky-Shapiro, G., and Frawley, W. J. 1991. *Knowledge Discovery in Databases*. AAAI/MIT Press.
- Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.
- Shen, W.; Ong, K.; Mitbender, B.; and Zaniolo, C. 1996. Metaqueries for data mining. In Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds., *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press. 375-398.
- Srikant, R., and Agrawal, R. 1995. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases*, 407-419.