The Quest Data Mining System

Rakesh Agrawal and **Manish Mehta** and **John Shafer** and **Ramakrishnan Srikant***
IBM Almaden Research Center
San Jose, California 95120, U.S.A.

Andreas Arning and **Toni Bollinger**
IBM German Software Development Laboratory
Boeblingen, Germany

Introduction

The goal of the Quest project at the IBM Almaden Research center is to develop technology to enable a new breed of data-intensive decision-support applications. This paper is a capsule summary of the current functionality and architecture of the Quest data mining System.

Our overall approach has been to identify basic data mining operations that cut across applications and develop fast, scalable algorithms for their execution (Agrawal, Imielinski, & Swami 1993a). We wanted our algorithms to:

- *discover* patterns in very large databases, rather than simply verify that a pattern exists;
- have a *completeness* property that guarantees that all patterns of certain types have been discovered;
- have high performance and near-linear scaling on very large (multiple gigabytes) real-life databases.

We discuss the operations of discovering association rules, sequential patterns, time-series clustering, classification, and incremental mining. Due to space limitation, we only give highlights and point the reader to the relevant information for details. Unfortunately, for the same reason, we have not been able to include a discussion of the related work. Besides proceedings of the KDD, SIGMOD, VLDB, and Data Engineering Conferences, other excellent sources of information about the data mining systems and algorithms include (Piatetsky-Shapiro & Frawley 1991) (Fayyad *et al.* 1995). Further information about Quest can be obtained from `http://www.almaden.ibm.com/cs/quest`. IBM is making the Quest technology commercially available through the data mining product, IBM Intelligent Miner.

Association Rules

We introduced the problem of discovering *association rules* in (Agrawal, Imielinski, & Swami 1993b). Given a set of transactions, where each transaction is a set of literals (called items), an association rule is an expression of the form $X \Rightarrow Y$, where X and Y are sets of items. The intuitive meaning of such a rule is that transactions of the database which contain X tend to contain Y. An example of an association rule is: "30% of transactions that contain beer also contain diapers; 2% of all transactions contain both of these items". Here 30% is called the *confidence* of the rule, and 2% the *support* of the rule. The problem is to find all association rules that satisfy user-specified minimum support and minimum confidence constraints. Applications include discovering affinities for market basket analysis and cross-marketing, catalog design, loss-leader analysis, store layout, customer segmentation based on buying patterns, etc. See (Nearhos, Rothman, & Viveros 1996) for a case study of a successful application in health insurance.

Apriori Algorithm

The problem of mining association rules is decomposed into two subproblems (Agrawal, Imielinski, & Swami 1993b):

- Find all combinations of items that have transaction support above minimum support. Call those combinations *frequent* itemsets.
- Use the frequent itemsets to generate the desired rules. The general idea is that if, say, $ABCD$ and AB are frequent itemsets, then we can determine if the rule $AB \Rightarrow CD$ holds by computing the ratio $r =$ support($ABCD$)/support(AB). The rule holds only if $r \geq$ minimum confidence. Note that the rule will have minimum support because $ABCD$ is frequent.

The Apriori algorithm (Agrawal & Srikant 1994) used in Quest for finding all frequent itemsets is

[*] Current members of the Quest group.

```
procedure AprioriAlg()
begin
   $L_1$ := {frequent 1-itemsets};
   for ( $k := 2$; $L_{k-1} \neq \emptyset$; $k{+}{+}$ ) do {
      $C_k$ := apriori-gen($L_{k-1}$); // New candidates
      forall transactions $t$ in the dataset do {
         forall candidates $c \in C_k$ contained in $t$ do
            $c$.count++;
      }
      $L_k := \{c \in C_k \mid c.\text{count} \geq \text{min-support}\}$
   }
   Answer := $\bigcup_k L_k$;
end
```

Figure 1: Apriori Algorithm

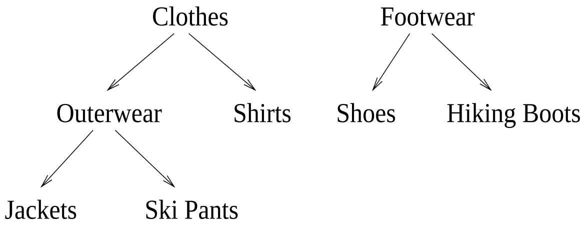


Figure 2: Example of a Taxonomy

given in Figure 1. It makes multiple passes over the database. In the first pass, the algorithm simply counts item occurrences to determine the frequent 1-itemsets (itemsets with 1 item). A subsequent pass, say pass k, consists of two phases. First, the frequent itemsets L_{k-1} (the set of all frequent $(k-1)$-itemsets) found in the $(k-1)$th pass are used to generate the candidate itemsets C_k, using the *apriori-gen()* function. This function first joins L_{k-1} with L_{k-1}, the joining condition being that the lexicographically ordered first $k-2$ items are the same. Next, it deletes all those itemsets from the join result that have some $(k-1)$-subset that is not in L_{k-1}, yielding C_k. For example, let L_3 be {{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}. After the join step, C_4 will be {{1 2 3 4}, {1 3 4 5} }. The prune step will delete the itemset {1 3 4 5} because the itemset {1 4 5} is not in L_3. We will then be left with only {1 2 3 4} in C_4.

The algorithm now scans the database. For each transaction, it determines which of the candidates in C_k are contained in the transaction using a hash-tree data structure and increments the count of those candidates. At the end of the pass, C_k is examined to determine which of the candidates are frequent, yielding L_k. The algorithm terminates when L_k becomes empty.

Generalizations

Very often, taxonomies (*is-a* hierarchies) over the items are available. An example of a taxonomy is shown in Figure 2: this taxonomy says that Jacket *is-a* Outerwear, Ski Pants *is-a* Outerwear, Outerwear *is-a* Clothes, etc. Users are often interested in generating rules that span different levels of the taxonomy. For example, we may infer a rule that people who buy Outerwear tend to buy Hiking Boots from the fact that people bought Jackets with Hiking Boots and and Ski Pants with Hiking Boots. However, the support for the rule "Outerwear ⇒ Hiking Boots" may not be the sum of the supports for the rules "Jackets ⇒ Hiking Boots" and "Ski Pants ⇒ Hiking Boots" since some people may have bought Jackets, Ski Pants and Hiking Boots in the same transaction. Also, "Outerwear ⇒ Hiking Boots" may be a valid rule, while "Jackets ⇒ Hiking Boots" and "Clothes ⇒ Hiking Boots" may not. The former may not have minimum support, and the latter may not have minimum confidence. This generalization of association rules and the algorithm used in Quest for finding such rules are described in (Srikant & Agrawal 1995).

Another generalization of the problem of mining association rules is to discover rules in data containing both quantitative and categorical attributes. An example of such a "quantitative" association rule might be that "10% of married people between age 50 and 60 have at least 2 cars". We deal with quantitative attributes by fine-partitioning the values of the attribute and then combining adjacent partitions as necessary. We also have measures of partial completeness that quantify the information loss due to partitioning. This generalization and the algorithm for finding such rules used in Quest are presented in (Srikant & Agrawal 1996a).

One potential problem that users experience in applying association rules to real problems is that many uninteresting or redundant rules may be generated along with the interesting rules. In (Srikant & Agrawal 1995) (further generalized in (Srikant & Agrawal 1996a)), a "greater-than-expected-value" interest measure was introduced, which is used in Quest to prune redundant rules.

Sequential Patterns

We introduced the problem of discovering *sequential patterns* in (Agrawal & Srikant 1995). The input data is a set of sequences, called data-sequences. Each data-sequence is a list of transactions, where each transaction is a sets of items (literals). Typically there is a transaction-time associated with each transaction. A *sequential pattern* also consists of a list of sets of items. The problem is to find all sequential patterns with a user-specified minimum *support*, where the support of a sequential pattern is the percentage of data-sequences that contain the pattern.

For example, in the database of a book-club, each data-sequence may correspond to all book selections of a customer, and each transaction to the books selected by the customer in one order. A sequential pattern might be "5% of customers bought 'Foundation', then 'Foundation and Empire', and then 'Second Foundation'". The data-sequence corresponding to a customer who bought some other books in between these books still contains this sequential pattern; the data-sequence may also have other books in the same transaction as one of the books in the pattern. Elements of a sequential pattern can be sets of items, for example, "'Foundation' and 'Ringworld', followed by 'Foundation and Empire' and 'Ringworld Engineers', followed by 'Second Foundation'". However, all the items in an element of a sequential pattern must be present in a single transaction for the data-sequence to support the pattern.

This problem was initially motivated by applications in the retailing industry, including attached mailing, add-on sales, and customer satisfaction. But the results apply to many scientific and business domains. For instance, in the medical domain, a data-sequence may correspond to the symptoms or diseases of a patient, with a transaction corresponding to the symptoms exhibited or diseases diagnosed during a visit to the doctor. The patterns discovered using this data could be used in disease research to help identify symptoms/diseases that precede certain diseases.

Generalizations

The basic definition of sequential patterns was generalized in (Srikant & Agrawal 1996b) to incorporate following features:

- **Introduction of time constraints.** Users often want to specify maximum and/or minimum time gaps between adjacent elements of the sequential pattern. For example, a book club probably does not care if someone bought "Foundation", followed by "Foundation and Empire" three years later; they may want to specify that a customer should support a sequential pattern only if adjacent elements occur within a specified time interval, say three months.

- **Flexible definition of a transaction.** For many applications, it is immaterial if items in an element of a sequential pattern were present in two different transactions, as long as the transaction-times of those transactions are within some small time window. That is, each element of the pattern can be contained in the union of the items bought in a set of transactions, as long as the difference between the maximum and minimum transaction-times is less than the size of a sliding time window. For example, if the book-club specifies a time window of a week, a customer who ordered the "Foundation" on Monday, "Ringworld" on Saturday, and then "Foundation and Empire" and "Ringworld Engineers" in a single order a few weeks later would still support the pattern "'Foundation' and 'Ringworld', followed by 'Foundation and Empire' and 'Ringworld Engineers'".

In addition, if there were taxonomies (*is-a* hierarchies) over the items in the data, the sequential patterns could now include items across different levels of the taxonomy.

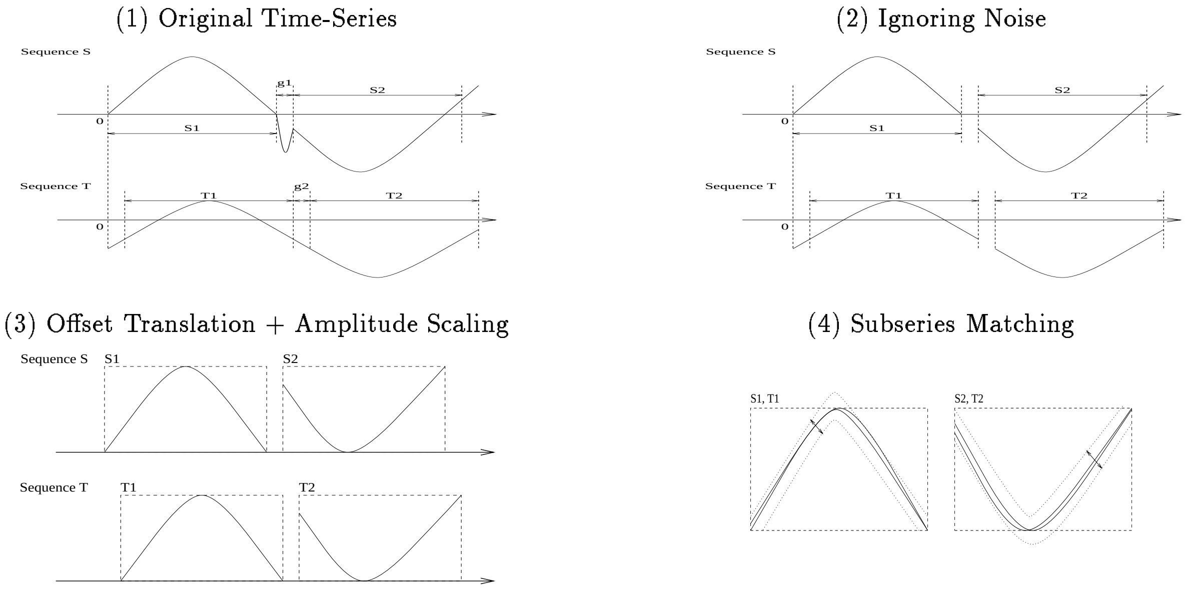
See (Srikant & Agrawal 1996b) for a description of the GSP algorithm used in Quest for finding such generalized sequential patterns.

Time-Series Clustering

Time-series data constitute a large portion of data stored in computers. The capability to find time-series (or portions thereof) that are "similar" to a given time-series or to be able to find groups of similar time-series has several applications. Examples include identifying companies with similar pattern of growth, finding products with similar selling patterns, discovering stocks with similar price movements, determining portions of seismic waves that are not similar to spot geological irregularities, etc.

We introduced a model of time-series similarity in (Agrawal *et al.* 1995a). In this model, two time-series are considered to be similar if they have enough non-overlapping time-ordered pieces (subseries) that are similar. The amplitude of one of the two time-series is allowed to be scaled by any suitable amount and its offset adjusted appropriately before matching the other series. Two subsequences are considered similar if one lies within an envelope of ϵ width around the other, ignoring outliers. The matching subseries need not be aligned along the time axis. Figure 3 captures the intuition underlying our similarity model.

The matching system used in Quest is described in (Agrawal *et al.* 1995a). It consists of three main parts: (i) "atomic" subseries matching, (ii) long subseries matching, and (iii) series matching. The basic idea is to create a fast, indexable data structure using small, atomic subseries that represents all the series up to amplitude scaling and offset, and find atomic matches by doing a self-join on this structure. The initial prototype used the $R+$-tree for this representation. A faster data structure described in (Shim, Srikant, & Agrawal


(1) Original Time-Series

(2) Ignoring Noise

(3) Offset Translation + Amplitude Scaling

(4) Subseries Matching

Figure 3: Illustration of Time-series matching

1996) is now used in its place. The second stage employs a fast algorithm for stitching atomic matches to form long subseries matches, allowing non-matching gaps to exist between the atomic matches. The third stage linearly orders the subseries matches found in the second stage to determine if enough similar pieces exist in the two time-series. In every stage, the system allows for the flexibility of user/system-defined matching parameters without sacrificing efficiency.

Classification

Classification is a well recognized data mining operation and it has been studied extensively in statistics and machine learning literature (Weiss & Kulikowski 1991). However, most of the current classification algorithms have the restriction that the training data should fit in memory. In data mining applications, very large training sets with several million examples are common. We therefore wanted to design a classifier that scales well and can handle training data of this magnitude (without resorting to sampling/partitioning). The ability to classify larger training data can also lead to improved classification accuracy.

SLIQ (*S*upervised *L*earning *I*n *Q*uest), described in (Mehta, Agrawal, & Rissanen 1996), is a decision tree classifier, designed to classify large training data. It uses a pre-sorting technique in the tree-growth phase. This sorting procedure is integrated with a breadth-first tree growing strategy to enable classification of disk-resident datasets. In the pruning phase, it uses a pruning strategy based on the Minimum Description Length (MDL) principle. The net result of these techniques is that, given training data that can be handled by another decision tree classifier, SLIQ exhibits the same accuracy characteristics, but executes much faster and produces smaller trees. Moreover, SLIQ can potentially obtain higher accuracies by classifying larger (disk-resident) training datasets which cannot be handled by other classifiers.

While SLIQ was the first classifier to address several issues in building a fast scalable classifier and it gracefully handles disk-resident data that are too large to fit in memory, it still requires some information to stay memory-resident. Furthermore, this information grows in direct proportion to the number of input records, putting a hard-limit on the size of training data. We have recently designed a new decision-tree-based classification algorithm, called SPRINT (*S*calable *P*a*R*allelizable *IN*duction of decision *T*rees) that for the first time removes all of the memory restrictions, and is fast and scalable and easily parallelizable. The algorithm, presented in (Shafer, Agrawal, & Mehta 1996), can classify data sets irrespective of the number of classes, attributes, and examples (records), making it an attractive tool for data mining.

Incremental Mining

As the data mining technology is applied in the production mode, the need for *incremental/active* mining arises (Agrawal & Psaila 1995). Rather than applying a mining algorithm to the whole data, the data is first partitioned according to time periods. The granularity of the time period is application-dependent. The mining algorithm is now applied to each of the partitioned data sets and patterns are obtained for each time period. These patterns are collected into a database. In this database, each statistical parameter of a pattern will have a sequence of values, called the history of the parameter for that pattern. We can now query the database using predicates that select patterns based on the shape of the history of some or all parameters. A shape query language is presented for this purpose in (Agrawal *et al.* 1995b).

The user can specify triggers over the database in which the triggering condition is a query on the shape of the history. As fresh data comes in for the current time period, the mining algorithm is run over this data, and the database is updated with the generated patterns. This update causes the histories of the patterns to be extended. This, in turn, may cause the triggering condition to be satisfied for some patterns and the corresponding actions to be executed.

Such active systems can be used, for instance, to build early warning systems for spotting trends in the retail industry. For example, if we were mining association rules, we would have histories for the support and confidence of each rule. Following the promotion for an item X, the user may specify a notification trigger on the rule $X \Rightarrow Y$; the triggering condition being that the support history remains stable, but the confidence history takes the shape of a downward ramp. Firing of this trigger will signify that if the goal of promoting X was to drag the sale of Y, it was not fulfilled. The loyalists continued to buy X and Y together, but the new buyers cherry-picked X.

Parallelism

Given that mining can involve very large amounts of data, parallel algorithms are needed. Quest algorithms have been parallelized to run on IBM's shared-nothing multiprocessor SP2. The parallel implementation of the mining of association rules is described in (Agrawal & Shafer 1996). This implementation shows linear scale-up for association rules. Mining of sequential patterns is also parallelized using similar techniques. We have also parallelized the SPRINT classification algorithm (Shafer, Agrawal, & Mehta 1996), where all processors work together to build a single classification model. Measurements from these implementations show excellent scaleup, speedup and sizeup characteristics.

System Architecture

Figure 4 shows the system architecture of the Quest system. The mining algorithms run on the server close to the data source. Users interact with the system through a GUI that can run on the same work-station or on a different client machine. There is an open API using which the user can optionally import results of any mining operation into software of choice.

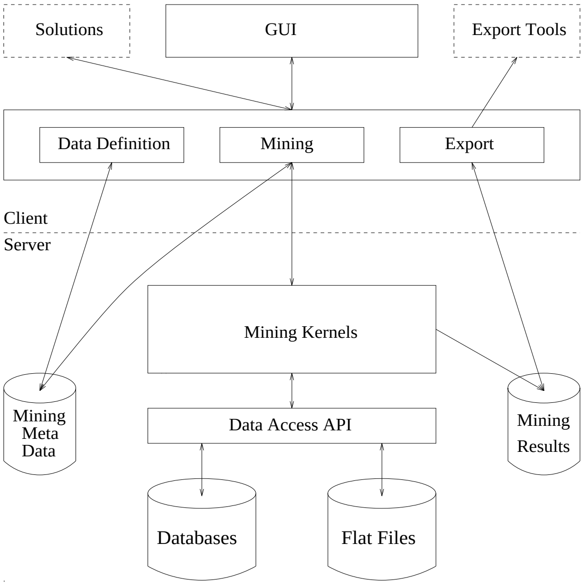


Figure 4: The Quest System Architecture

An interesting aspect of the Quest architecture is its I/O architecture. There is a standard stream interface defined for all accesses to input, insulating the algorithm code from data repository details, which are encapsulated in a data access API. Thus, it is easy to add new data repository types to the Quest system.

The Quest system runs both on AIX and MVS platforms, against data in flat files as well as DB2 family of database products. Databases can be accessed in a loosely-coupled mode using dynamic SQL. However, for better performance, it is possible to run the mining algorithm in a tightly-coupled mode described in (Agrawal & Shim 1996).

Future Directions

We plan to continue on the current path of identifying new data mining operations and developing fast algorithms for their execution. Two operations that we are currently focusing on are deviation detection (Arning & Agrawal 1996) and segmentation of high-dimensional data. We are also interested in mining data types other than structured data, such as text, particularly in the context of the world-wide web. Finally, we are also exploring the interaction between discovery-driven and verification-driven data mining, especially in OLAP databases.

Acknowledgment The Quest system is the result of efforts of many who rightfully belong to the author list. They include George Almasi, Ute Baumbach, Dominique Brodbeck, Srinivas Bulusu, Mike Carey, Stefan Dipper, Christos Faloutsos, Takeshi Fukuda, Sakti Ghosh, Ashish Gupta, Howard Ho, Maurice Houtsma, Tomasz Imielinski, Bala Iyer, David Lin, Hansel Miranda, Shinichi Morishita, Leon Pauser, Giuseppe Psaila, Prabhakar Raghavan, Jorma Rissanen, Harpreet Sawhney, Sunita Sarawagi, Ashok Savasere, Kyuseok Shim, Arun Swami, Peter Vigants, Marisa Viveros, Ed Wimmers, and Mohamed Zaït. Bill Bath, Cindy Baune, Tony Clacko, Ron Groves, Barry Mason, and Molly Nicole were some of the early adopters of the Quest system. Finally, the Quest sytem would not have become the reality but for the faith of Chris Arnold, Walter Baur, Ashok Chandra, Bill Cody, Peter Greissl, Laura Haas, Frank Leymann, John McPherson, Dragutin Petkovic, Pat Selinger, Mario Schkolnick, and Irv Traiger.

References

Agrawal, R., and Psaila, G. 1995. Active data mining. In *Proc. of the 1st Int'l Conference on Knowledge Discovery in Databases and Data Mining.*

Agrawal, R., and Shafer, J. 1996. Parallel mining of association rules: Design, implementation and experience. Research Report RJ 10004, IBM Almaden Research Center, San Jose, California. To appear in *IEEE Transactions on Knowledge and Data Engineering.*

Agrawal, R., and Shim, K. 1996. Developing tightly-coupled data mining applications on a relational database system. In *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining.*

Agrawal, R., and Srikant, R. 1994. Fast Algorithms for Mining Association Rules. In *Proc. of the 20th Int'l Conference on Very Large Databases.*

Agrawal, R., and Srikant, R. 1995. Mining Sequential Patterns. In *Proc. of the 11th Int'l Conference on Data Engineering.*

Agrawal, R.; Lin, K.-I.; Sawhney, H. S.; and Shim, K. 1995a. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proc. of the 21st Int'l Conference on Very Large Databases.*

Agrawal, R.; Psaila, G.; Wimmers, E. L.; and Zaït, M. 1995b. Querying shapes of histories. In *Proc. of the 21st Int'l Conference on Very Large Databases.*

Agrawal, R.; Imielinski, T.; and Swami, A. 1993a. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering* 5(6):914–925.

Agrawal, R.; Imielinski, T.; and Swami, A. 1993b. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, 207–216.

Arning, A., and Agrawal, R. 1996. A linear method for deviation detection in large databases. In *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining.*

Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.; and Uthurusamy, R., eds. 1995. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press.

Mehta, M.; Agrawal, R.; and Rissanen, J. 1996. SLIQ: A fast scalable classifier for data mining. In *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT).*

Nearhos, J.; Rothman, M.; and Viveros, M. 1996. Applying data mining techniques to a health insurance information system. In *Proc. of the 22nd Int'l Conference on Very Large Databases.*

Piatetsky-Shapiro, G., and Frawley, W. J., eds. 1991. *Knowledge Discovery in Databases.* Menlo Park, CA: AAAI/MIT Press.

Shafer, J.; Agrawal, R.; and Mehta, M. 1996. Fast serial and parallel classification of very large data bases. In *Proc. of the 22nd Int'l Conference on Very Large Databases.*

Shim, K.; Srikant, R.; and Agrawal, R. 1996. The ϵ-k-d-b tree: A fast index structure for high-dimensional similarity joins. Research Report, IBM Almaden Research Center, San Jose, California.

Srikant, R., and Agrawal, R. 1995. Mining Generalized Association Rules. In *Proc. of the 21st Int'l Conference on Very Large Databases.*

Srikant, R., and Agrawal, R. 1996a. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. of the ACM SIGMOD Conference on Management of Data.*

Srikant, R., and Agrawal, R. 1996b. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT).*

Weiss, S. M., and Kulikowski, C. A. 1991. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufman.