Database Mining

Bio-Data

Jayant Ramaswamy Haritsa is with the Supercomputer Education & Research Centre and the Department of Computer Science & Automation at the Indian Institute of Science. He works on adapting database technology to new application domains such as VLSI Design, Flexible Manufacturing, Biodiversity and Real-Time Systems.

Article Summary

Database Mining refers to the process of efficiently extracting patterns from huge historical databases. The pattern information is useful in helping organizations make better business decisions.

1 Introduction

Organizations typically collect vast quantities of data relating to their operations. For example, the Income Tax department has several *terabytes* of data stored about taxpayers ! In order to provide a convenient and efficient environment for users to productively use these huge data collections, software packages called "DataBase Management Systems" have been developed and are widely used all over the world (see Box 1 for more details on database applications).

A DataBase Management System, or DBMS, as it is commonly referred to, provides a variety of useful features: Firstly, business rules such as, for example, "the minimum balance to be maintained in a savings account is 100 Rupees", are not allowed to be violated. Secondly, modifications made to the database are never lost even if system failures occur subsequently. Thirdly, the data is always kept consistent even if multiple users access and modify the database concurrently. Finally, friendly and powerful interfaces are provided to ask questions on the information stored in the database.

Users of DBMSs interact with them in basic units of work called *transactions*. Transfer of money from one account to another, reservation of train tickets, filing of tax returns, entering marks on a student's grade sheet, are all examples of transactions. A transaction is similar to the notion of a task in operating systems, the main difference being that transactions are significantly more complex in terms of the functionality provided by them (see Box 2 for a more detailed description of transactions).

For large commercial organizations, where thousands of transactions are executed every second, highly sophisticated DBMSs such as DB2, Informix, Sybase, Ingres and Oracle are available and are widely used. For less demanding environments, such as managing home accounts or office records, there are a variety of PC–based packages, popular examples being FoxPro, Microsoft Access and dBase IV.

DataBase Management Systems typically operate on data that is resident on hard disk. However, since disk capacities are usually limited, as more and more new data keeps coming in, organizations are forced to transfer their old data to tapes. Even if these tapes are stored carefully, the information resident in them is hardly ever utilized (similar to the files in our government offices !). This is highly unfortunate since the historical databases often contain extremely useful information, as described below.

Database Mining

The primary worth of the historical information is that it can be used to detect *patterns*. For example, a supermarket that maintains a database of customer purchases may find that customers who purchase coffee powder very often also purchase sugar. Based on this information, the manager may decide to place coffee powder and sugar on adjacent shelves to enhance customer convenience. The manager may also ensure that whenever fresh stocks of coffee powder are ordered, commensurate quantities of sugar are also procured, thereby increasing the company's sales and profits. Information of this kind may also be used beneficially in catalog design, targeted mailing, customer segmentation, scheduling of sales, etc. In short, the historical database is a "gold mine" that can be profitably used to make better business decisions.

In the technical jargon, data patterns (of the type described above) are called "rules" and the process of identifying such rules from huge historical databases is called **database mining**. This issue has recently become a hot topic of research in the database community and is the subject of this article. Readers familiar with learning techniques will be aware that discovering rules from data has been an area of active research in artificial intelligence. However, these techniques have been evaluated in the context of small (in memory) data sets and perform poorly on large data sets. Therefore, database mining can be viewed as the confluence of machine learning techniques and the performance emphasis of database technology. In particular, it refers to the efficient construction and verification of models of patterns embedded in large databases.

2 Rules

In normal usage, the term "rule" is used to denote implications that are always true. For example, Boyle's law *Pressure * Volume = constant* can be inferred from scientific data. For commercial applications, however, this definition is too restrictive and the notion of rule is expanded to denote implications that are *often*, but not necessarily *always*, true. To quantify this uncertainity, a *confidence factor* is associated with each rule. This factor denotes the probability that a rule will turn out to be true in a specific instance. For example, the statement "Ninety percent of the customers who purchase coffee powder also purchase sugar" corresponds to a rule with confidence factor 0.9.

In order for rules of the above nature to be meaningful, they should occur reasonably often in the database. That is, if there were a million customer purchases and, say, only ten of these customers bought coffee powder, then the above example rule would be of little value to the supermarket manager. Therefore, an additional criterion called the *support factor* is used to distinguish "significant" rules. This factor denotes the fraction of transactions in the database that support the given rule. For example, a customer purchase rule with support factor of 0.20 means that twenty percent of the overall purchases satisfied the rule.

At first glance, the confidence factor and the support factor may appear to be similar concepts. However, they are really quite different: Confidence is a measure of the rule's strength, while support corresponds to statistical significance.

Customer	Potatoes	Onions	Tomatoes	Carrots	Beans
1	N	N	N	N	Y
2	Y	Y	N	Y	Y
3	Y	Y	Y	N	N
4	Y	Y	N	N	Y
5	Y	Y	Y	N	N
6	Y	Y	Y	Y	N
7	Y	Y	Y	N	N
8	Y	N	N	N	Y
9	Y	Y	Y	N	N
10	Y	Y	Y	N	Y

Table 1: Vegetable Purchase Database

Formal Definition

Based on the foregoing discussion, the notion of a rule can be formally defined as:

$$X \Longrightarrow Y \ \mid \ (c, s)$$

where X and Y are disjoint subsets of $\mathcal{I}$, the set of all items represented in the database, c is the confidence factor, and s is the support factor. The factors, which are ratios, are usually expressed in terms of their equivalent percentages.

To make the above definition clear, consider a vegetable vendor who sells potatoes, onions, tomatoes, carrots and beans, and maintains a database that stores the names of the vegetables bought in each customer purchase, as shown in Table 1. For this scenario, the itemset $\mathcal{I}$ corresponds to the set of vegetables that are offered for sale. Then, on mining the database we will find rules of the following nature:

$$Potatoes, Onions \Longrightarrow Tomatoes \ \mid \ (75, 60)$$

$$Beans \Longrightarrow Potatoes \ \mid \ (80, 40)$$

which translate to "Seventy-five percent of customers who bought potatoes and onions also bought tomatoes. Sixty percent of the customers made such purchases" and "Eighty percent of the customers who bought beans also bought potatoes. Forty percent of the customers made such purchases", respectively.

A word of caution: The rules derived in the above example are not truly valid since the database used in the example is a "toy" database that has only ten customer records – it was provided only for illustrative purposes. For rules to be meaningful, they should be derived from large databases that have thousands of customer records, thereby indicating consistent patterns, not transient phenomena. A real-world example of rules mined from a large database is shown in Box 3.

3 Rule Discovery

As mentioned in the Introduction, identifying rules based on patterns embedded in the historical data can serve to improve business decisions. Of course, rules may sometimes be "obvious" or "common-sense", that is, they would be known without mining the database. For example, the fact that butter is usually bought with bread is known to every shopkeeper. In large organizations, however, rules may be more subtle and are realized only after mining the database.

Given the need for mining historical databases, we would obviously like to implement this in as efficient a manner as possible since searching for patterns can be computationally extremely expensive. Therefore, the main focus in data mining research has been on designing efficient rule discovery algorithms.

The inputs to the rule discovery problem are $\mathcal{I}$, a set of items, and $\mathcal{D}$, a database that stores transactional information about these items. In addition, the user provides the values for sup_{min}, the minimum level of support that a rule must have to be considered significant by the user, and con_{min}, the minimum level of confidence that a rule must have in order to be useful. Within this framework, the rule mining problem can be decomposed into two sub-problems:

Frequent Itemset Generation

Find all combinations of items that have a support factor of at least sup_{min}. These combinations are called *frequent itemsets*, whereas all other combinations are called *rare itemsets* (since they occur too infrequently to be of interest to the user).

Strong Rule Derivation

Use the frequent itemsets to generate rules that have the requisite strength, that is, their confidence factor is at least con_{min}.

In the following two sections, techniques for solving each of the above sub-problems are presented.

4 Frequent Itemset Generation

A simple and straightforward method for generating frequent itemsets is to make a *single pass* through the entire database, and in the process measure the support for every itemset resident in the database. Implementing this solution requires setting up of a measurement counter for each subset of the set of items $\mathcal{I}$ that occurs in the database, and in the worst case, when every subset is represented, the total number of counters required is 2^M, where M is the number of items in $\mathcal{I}$. Since M is typically of the order of a few hundreds or thousands, the number of counters required far exceeds the capabilities of present-day computing systems. Therefore, the "one-pass" solution is clearly infeasible, and several "multi-pass" solutions have therefore been developed.

A simple multi-pass solution works as follows: The algorithm makes multiple passes over the database and in each pass, the support for only certain specific itemsets is measured. These itemsets are called *candidate itemsets*. At the end of a pass, the support for each of the candidate itemsets associated with that pass is evaluated and compared with sup_{min} (the minimum support) to determine whether the itemset is frequent or rare.

Candidate itemsets are identified using the following scheme: Assume that the set of frequent itemsets found at the end of the kth pass is F_k. Then, in the next pass, the candidate itemsets are comprised of all itemsets that are constructed as *one-extensions* of itemsets present in F_k. A one-extension of an itemset is the itemset extended by exactly one item. For example, given a set of items A, B, C, D and E, the one-extensions of the itemset AB are ABC, ABD and ABE. While this scheme of generating candidate itemsets works in general, in order to start off the process we need to prespecify the candidate itemsets for the very first pass ($k = 1$). This is done by making each individual item in $\mathcal{I}$ to be a candidate itemset for the first pass.

The basic idea in the above procedure is simply that "If a particular itemset is found to be rare, then all its extensions are also guaranteed to be rare". This is because the support for an extension of an itemset cannot be more than the support for the itemset itself. So, if AB is found to be rare, there is no need to measure the support of ABC, ABD, $ABCD$, etc., since they are certain to be also rare. Therefore, in each pass, the search space is *pruned* to measure only those itemsets that are potentially frequent and the rare itemsets are discarded from consideration.

Another feature of the above procedure is that in the kth pass over the database, only itemsets that contain exactly k items are measured, due to the one-extension approach. This means that no more than M passes are required to identify all the frequent itemsets resident in the historical database.

Algorithms that are even better than the one outlined above have been proposed in recent database conferences. In fact, one paper presents a novel technique by which all frequent itemsets are generated in just two passes! Moreover, different approaches to database mining have also been investigated (one of these is described in Box 4).

5 Strong Rule Derivation

In the previous section, we described methods for generating frequent itemsets. We now move on to the second sub-problem, namely that of deriving strong rules from the frequent itemsets. The rule derivation problem can be solved using the following simple method: For every frequent itemset F, enumerate all the subsets of F. For every such subset f, output a rule of the form $f \Longrightarrow (F - f)$ if the rule is sufficiently strong. The strength is easily determined by computing the ratio of the support factor of F to that of the support factor of f. If this value is at least con_{min}, the minimum rule confidence factor, the rule is considered to be strong and is displayed to the user, otherwise it is discarded.

In the above procedure, the only part that is potentially difficult is the enumeration of all the subsets of each frequent itemset. However, efficient algorithms for performing this task are available and therefore the rule derivation problem is easy to handle.

6 Classification and Sequence Rules

The rules that we have discussed so far are called *association rules* since they involve finding associations between sets of items. However, they are only one example of the types of rules that may be of interest to an organization. Other interesting rule classes that have been identified in the literature are *classification rules* and *sequence rules*, and data mining algorithms for discovering these types of rules

have also been developed in the last few years.

The classification problem involves finding rules that *partition* the data into disjoint groups. For example, the courses offered by a college may be categorized into good, average and bad based on the number of students who attend each course. Assume that the attendance in a course is primarily based on the qualities of the teacher. Also assume that the college has maintained a database about the attributes of all its teachers. With this data and the course attendance information, a profile of the attributes of successful teachers can be developed. Then, this profile can be used by the college for short-listing the set of good candidate teachers whenever new courses are to be offered. For example, the rule could be "If a candidate has a master's degree, is less than 40 years old, and has more than 5 years experience, then the candidate is expected to be a good teacher".

Organizations quite often have to deal with *ordered* data, that is, data that is sorted on some dimension, usually time. Currency exchange rates and stock share prices are examples of this kind of data. Rules derived from ordered data are called sequence rules. An example rule of this type is "When the value of the US dollar goes up on two consecutive days and the British pound remains stable during this period, the Indian rupee goes up the next day 75 percent of the time".

7 Summary

The goal of Database Mining is to discover information from historical organizational databases that can be used to improve their business decisions. Developing efficient algorithms for mining has become an active area of research in the database community in the last few years. Although commercial data mining packages are not yet available, there are several research prototypes that have been developed. Examples are QUEST, constructed at IBM's Almaden Research Center in San Jose, California, U.S.A., and DISCOVER, available from the Hong Kong University of Science and Technology. We expect that sophisticated database mining packages will be available soon and that they will become essential software for all organizations within a few years.

Acknowledgements

The material presented in this paper is mainly derived from the publications mentioned in the Suggested Reading list.

This work was supported in part by a research grant from the Dept. of Science and Technology, Govt. of India.

Suggested Reading

Basic Texts on Database Systems

1. H. Korth and A. Silberschatz, *Database System Concepts*, 2nd ed., McGraw-Hill, 1991.

2. R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 2nd ed., Addison-Wesley, 1994.

3. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993.

4. V. Rajaraman, *Analysis and Design of Information Systems*, Prentice-Hall India, 1991.

Research Papers on Data Mining

5. R. Agrawal, T. Imielinski and A. Swami, "Mining Association Rules between Sets of Items in Large Databases", *Proceedings of 22nd ACM SIGMOD International Conference on Management of Data*, Washington D.C., U.S.A., May 1993.

6. R. Agrawal, T. Imielinski and A. Swami, "Database Mining: A Performance Perspective", *IEEE Transactions on Knowledge and Data Engineering*, December 1993.

7. R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules", *Proceedings of 20th Very Large Data Base Conference*, Santiago, Chile, September 1994.

8. A. Savasere, E. Omiecinski and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases", *Proceedings of 21st Very Large Data Base Conference*, Zurich, Switzerland, September 1995.

Box 1 (DATABASE APPLICATIONS)

Database Management Systems are used in a variety of applications all over the world. Typical applications and examples include the following:

Communications : Setting up and billing for telephone calls, electronic mail, data transfer, etc.

Finance : Banking, stock trading, merchandise purchases, etc.

Travel : Reservations and billing for airlines, hotels, cars, etc.

Manufacturing : Order entry, job and inventory planning and scheduling, accounting, etc.

Process Control : Control of machines, transporters, warehouses, etc.

Educational : Library search and reservation, reference material generation, etc.

Database Management Systems are meant to support applications that have to interface with large quantities of data. Typical sizes of databases are shown in the table below for a representative set of applications:

Application	Typical Size
Gene Mapping, Income Tax Returns, Voter Information	Terabytes
Banks, Libraries, Multimedia, Computer-Aided Design	Gigabytes
Office Administration, Store Management, Timetables	Megabytes

Box 2 (TRANSACTIONS)

The state of a database, at any given time, is defined to be the collection of data items in the database and their current values. A database state is said to be consistent if the current values of the data items in the database obey all the rules specified by the organization that is maintaining the database.

A transaction is the fundamental unit of database processing, which operates on the database and takes it from one state to another. It is a software module that is a sequence of actions with the following properties:

Atomicity : The changes made by the transaction to the database are atomic, that is, either all happen or none happen.

Consistency : The transaction is a correct program in that if it executes on a consistent database, the resultant database state is also guaranteed to be consistent.

Isolation : Although transactions are individually correct (by the Consistency property), unrestricted *concurrent* execution of transactions in a multiuser database system may lead to an inconsistent database state. Therefore, transactions regulate their execution in a manner such that, even though they execute concurrently, it appears as if each transaction executes in isolation from all other transactions.

Durability : After a transaction completes successfully, the changes it has made to the database state are durable. That is, these changes are not lost even if system failures occur subsequently.

Collectively, the four properties listed above are called **ACID** properties! They are enforced by the DataBase Management System software.

To illustrate the transaction concept, consider a university depositing a monthly scholarship into a student's account. This transaction is composed of two actions: debiting the amount from the university's account, and, crediting the amount to the student's account. The transaction is atomic if it updates *both* the accounts. It is consistent if the credited amount is the same as the debited amount. It is isolated if the transaction can be unaware of other programs operating on the student's account concurrently (for example, the student making a withdrawal to pay his tuition fees). And it is durable if, once the transaction is complete, the balance in both accounts is guaranteed to reflect the transfer forever in the future.

Box 3 (LARGE DATABASE MINING EXAMPLE*)

A group of researchers recently collected sales data from a large retailing company located in the U.S.A. There were a total of 46,873 customer transactions in this data. Each transaction contained the names of the departments from which a customer bought an item during a visit. The store was comprised of 63 departments. Rule mining was executed on this data to determine the associations between departments in the customer purchasing behavior.

After mining, the following rules were found for a minimum support of 1 percent and minimum confidence of 50 percent.

Tires	$\Longrightarrow$	Automotive Services	(98.80, 5.79)
Auto Accessories, Tires	$\Longrightarrow$	Automotive Services	(98.29, 1.47)
Auto Accessories	$\Longrightarrow$	Automotive Services	(79.51, 11.81)
Automotive Services	$\Longrightarrow$	Auto Accessories	(71.60, 11.81)
Home Laundry Appliances	$\Longrightarrow$	Maintenance Agreement Sales	(66.55, 1.25)
Children's Hardlines	$\Longrightarrow$	Infant and Children's Wear	(66.15, 4.24)
Men's Furnishings	$\Longrightarrow$	Men's Sportswear	(54.86, 5.21)

* This example is taken from Reference 5 in the Suggested Reading list.

Box 4 (DATABASE SAMPLING)

At first glance, it would appear that rules of the type discussed in this article can be derived easily by using well-known statistical methods such as, for example, sampling. In sampling, inferences about an entire population are made based on characteristics exhibited by a representative subset of the population. This approach is especially attractive for data mining since, instead of scanning the whole database, only a small part of it has to be processed, thereby leading to much better performance.

The main drawback of the sampling approach is that it may not identify *all* the applicable rules, especially when the minimum support factor specified by the user is low. So, for example, if the minimum support factor is one percent, then sampling will usually identify all the high support rules but miss out on rules whose support is low (below five percent, say). This is because the chosen subset of the database may not reflect these low support rules. Recent research results indicate that for a minimum support of one percent, using even a sample that is as large as 50 percent of the database is not sufficient to ensure that all rules are generated !

If complete accuracy is not required by the user, however, then database sampling can be fruitfully incorporated into the data mining framework by executing the data mining algorithm on the sample rather than on the original database.