Using M to Discover Knowledge

by Charles Williams and Z. Chen

Introduction

Knowledge discovery in databases (also called database mining, or simply data mining) is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data. Knowledge discovery in databases has a very close relationship with machine-learning methods employed in artificial intelligence (AI). Knowledge-discovery techniques have been used to "rediscover" scientific laws such as Boyle's law; they are also useful in other applications such as discovering regularities in business data.[1,2]

According to a 1993 panel discussion, mining data has become a very popular activity, driven by the decision-support problem faced by most large retail-chain enterprises. Such retail chains record every item consumers purchase in every store by capturing these data at the point of sale. Buyers and merchandise arrangers then use this database to rotate stock and make purchasing decisions. The query to such a database is, "Tell me something interesting." Specifically, users want a system that mines the data for useful information.[3]

Relational-database systems offer many attractive features for machine learning. They store a large amount of information in a structured and organized manner. Such uniformity facilitates the development of efficient learning algorithms.[4]

The main purpose here is to show how to incorporate knowledge-discovery techniques into M computing. We first briefly review the basic idea of knowledge discovery in databases, then through example, show how knowledge discovery can be carried out. Sample codes follow in the appendix.

Basics of Knowledge Discovery in Databases

To illustrate the basic idea of knowledge discovery, consider the following process of learning characteristic rules.[5] A characteristic rule is an assertion that characterizes the concept satisfied by all the data stored in the database. Suppose that a student relation in a sample university database consists of the following fields: name, category, major, birth place, and grade point average (GPA). In addition, there is a concept-hierarchy table, which is a concept tree organized as an IS-A hierarchy (such as music and history can be generalized into *art*; junior and senior can be generalized into *undergraduate*, etc.).

Now we want to find out something interesting about graduate students. A four-step algorithm for learning a characteristic rule can be conducted as follows. Step 1 is the extraction of the task-relevant data by performing selection, projection, and join on the relevant relations (such as dropping the student name field, since we are not interested in individual students).

Step 2 is the attribute-oriented induction process; generalization should be performed on attributes by substituting each attribute value with its higher-level concept (such as replacing physics by science).

Step 3 is the simplification of the generalized relation (such as removal of duplication). Step 4 is the transformation of the final relation into a logic formula. Here is a sample rule that may be produced by the discovery process: A graduate student is either a Canadian with an excellent GPA or a foreign student majoring in science with a good GPA.

Note that this rule is not explicitly stated anywhere in the database. Rather, it is *derived* from the stored data.

The most popular format of a database rule takes the format of "If C1 then C2" or of C1 → C2. In fact, a rule is not necessary to cover all instances. If a rule is almost always correct, then it is called a strong rule.

Producing Suggestions with Database Mining

Since M computing has been extensively used for database management, it is important to incorporate knowledge-discovery techniques into M. Recently we have explored one aspect in this regard, namely, how to apply basic knowledge discovery to derive some missing information or provide suggestions to the user by mining the data available in the databases. We have developed a simple experimental program for this purpose. (To simplify our discussion, this experimental program will be referred to as our "system.")

Consider a relational-database schema (namely, a set of fields or attributes) and its relation (namely, the actual rows in the table) containing student records in a university. Part of this relation is depicted below.

s-name	ssn	age	sex	major	hobby	rank (percentage)
j Bohn	999	23	m	appl math	football	10
k Silverman	435	21	f	computer	dance	15
...						
(many other rows)						

As background information, the computer system also has a concept-hierarchy table to indicate the hierarchy of some concepts:

general	special	more special	(fields)
science	math	applied math	(rows in table)
science	computer sc		
art	fine art		
art	drama		
. . .			

The information stored in this table, of course, can be retrieved by users. We are interested in the following scenario, however, which cannot be handled by conventional database processing. Suppose somebody wants to apply for admission to this university, but does not know which major is appropriate. The potential applicant can ask our system for help.

The system works as follows. It interfaces with the user, asking for information for each field. In case the student does not know the answer for a particular field (such as the "major" field in this case), a blank or a question mark (?) can be used.

The following is a sample session of user interface:

s-name: b johnson
ssn: 435
age: 19
sex: m
major: ?
hobby: baseball
rank (percentage): 15

Notice that this kind of query is different from conventional queries, because this applicant is not in the database. In a conventional database system, nothing will be retrieved for this user. But our system can perform a kind of inference so that the data actually stored in the database can suggest information. In fact, in order to answer this query, the system will search through the existing rows in the original database. As a result of the search, candidate rules will be constructed and stored in table form. Finally, a "most likely" value will be suggested as an answer for the user. In general, a rule table in the system consists of two parts: the "if" part consisting of values of two or more fields (columns) and the "then" part consisting of one or more fields (columns).

Back to our example. A suggestion can be made to the potential applicant. Suppose searching the existing database shows that university students currently enrolled whose hobby and class rank are similar to the user tend to major in science. Then the following rule can be constructed:

```
if hobby = sports, rank(percentage) = 20
   then major = science
```

Consequently, "major = science" will be recommended to the potential applicant.

This rule is formed by dropping some columns in the original databases and performing some necessary generalizations (using the hierarchical table as described above).

A rule may be more specific or more general. For example, the following is another possible rule that is more specific than the one given originally (because the suggested field "computer science" is narrower than "science"):

```
if age < 30, rank(percentage) = 15
   then major = computer science
```

A Learning Algorithm Incorporating M Features

We now briefly describe the algorithm for our experiment in knowledge discovery. We have adopted part of the classification-rule-learning algorithm as briefly summarized in the second section of this article.[6] The original algorithm takes an attribute-oriented approach; we have revised it to incorporate some features of tuple-oriented approaches.[7] Another reason for revising the original algorithm is to incorporate some features of the M language so that it is more suitable for M computing.

Since rules to be discovered have the format of "if condition then conclusion," and since both condition and conclusion parts are concerned with one or more attributes, our algorithm loops through the database N times, where N is the total number of fields minus the number of fields in the drop list (such as student names). The key idea here is to start with fields to be included in the "then" portion of the rule to be discovered, then to work backward to deal with the "if" portion of the rule. In order to do this, we need to combine the other fields in all possible combinations to find the strongest rules. For example, assuming that the "then" part is dealing with the field "major."

A counter is used to record the frequencies of each candidate rule. The rules with highest frequencies will be selected. For example, in a database with 1,000 records, if 50 of them contain ages less than or equal to 30, rank less than or equal to 15, and the major = math, then it is reasonable to form a rule: if age = 20 and rank = 10, then major = math. In general, the number of fields needed to be considered at the "if" part of a rule may be *k* (a number between 2 and N). Whenever many fields are considered at the same time, the processing time could be long. Heuristics (including domain knowledge) can be incorporated to determine an appropriate value of *k*.

To simplify matters, in the following algorithm, K = 2. In this case, the second "for" loop will go through each field also and form all possible pairs of fields. Each occurring pair will be counted and stored as a candidate (possible) rule for each record processed. After all fields have been processed in this manner, the candidate rules stored in a separate table will then be checked for their strength.

Strength is measured by the frequency of occurrences. If the strength of a rule is not greater than some predefined threshold, the candidate rule is eliminated as such. At the end of this entire process, the rule table contains all the valid rules that can be used to offer a suggestion to an incoming query.

In summary, the algorithm used for database mining can be expressed as shown here (using pseudo code):

```
loop_1    for I is 1 to num_of_fields
              if I is not in dropped list
                 do loop_2
loop_2    for J is 1 to num_fields
              if J is not in dropped list and J is
              not equal to I
                 do loop_3
loop_3    for K is 1 to num_fields
              if K is not in dropped list and K is
              not equal to I
                 process record
```

M Implementation

We have implemented the above algorithm on an IBM PC (using an Intel 486 processor). The program is generic so that it is not dependent on a particular domain (such as a student database). We set up tables that contain information such as the names of the attributes, the number of attributes, and those attributes that should be dropped from consideration because they are not used for database-mining purposes (such as student names).

A standard version of M is used in our experiment.[8] A slightly edited version of a sample source program in M (slightly edited due to editorial consideration) is shown in the appendix, where the main program is depicted in figure 1, while the user interface part is shown in figure 2.

There are some advantages in using M as a programming language. The most important advantage is the ability to use strings as array subscripts. This allows us to develop, enumerate, and store discovered rules all within the same array. When retrieving the rules from the rule table, we could use these same subscripts in a comparison with the pattern being searched. The flexibility of M to loop has made the code short and simple.

Various Applications

Database mining can be used in many application domains of the real world. Generally, applications can be used either for predictive (namely, to predict a result) and abductive (namely, to find a most likely explanation) purposes, or a combination of both. For example, in a travel agency with a database recording previous customers' profiles and their destinations, suggestions for possible vacation destinations can be made for travelers who do not have a clear destination in mind.

To realize this, a historical database of user queries can be established by recording user profiles and observing user behavior. Using the algorithm similar to that shown in the previous section, rules can be constructed to associate variables related to user profile on one hand, and destination on the other. A rule might look like this:

```
If income = medium and interest = for_kids
   then destination = Orlando
```

As another example of database mining, from a series of queries submitted by the same user, knowledge-discovery techniques can help to detect what this user really wants. Database mining can also help chart a market trend from the bulk of recent data with many variables.

Related work can be found in recent journals and proceedings; for example, an interval classifier for database mining can be found in Agrawal's article on an interval classifier.[9]

Conclusion

Although our work is still in the experimental stage, and although some improvement is needed, we believe that knowledge-discovery techniques have good potential in M computing, and M computing provides various technical advantages for knowledge discovery.

In addition, we also believe that this study will benefit both industry and educational institutions. In fact, the basic concept of knowledge discovery in databases can be incorporated into upper-level computer-science courses such as artificial intelligence or database-management systems.

Charles Williams is graduating from the University of Nebraska at Omaha. He has five years' experience in the data-processing field, primarily with M. He was a programmer-analyst at World Data, Inc., (Omaha, Nebraska) when he implemented the project described in this article. He is now affiliated with Advertech Ltd. (Atlanta, Georgia).

Z. Chen holds a Ph.D. from Louisiana State University. He has been affiliated with the Department of Computer Science at the University of Nebraska at Omaha since 1988. He is interested in various issues about building intelligent information systems, including knowledge discovery in databases. His e-mail address is cs061@unocss.unomaha .edu.

Endnotes

1. W.J. Frawley, G. Piatetsky-Shapiro, and C.J. Matheus, "Knowledge Discovery in Databases: An Overview." *AI Magazine*, 13: 3 (1992): 57–70.
2. G. Piatetsky-Shapiro and W.J. Frawley, eds., *Knowledge Discovery in Databases* (Cambridge, Massachusetts: MIT/AAAI Press, 1991).
3. M. Stonebraker et al., "DBMS Research at a Crossroads: The Vienna Update," in *Proceedings of the 19th VLDB*, eds. R. Agrawal, S. Baker and D. Bell (Menlo Park, California: Morgan Kaufmann, 1993), 688–692.
4. Y. Cai, N. Cercone, and J. Han, "Attribute-Oriented Induction in Relational Databases," in *Knowledge Discovery in Databases*, ed. G. Piatetsky-Shapiro and W.J. Frawley, 213–228.
5. Cai.
6. Cai.
7. D.J. Russull and B.N. Grosof, "A Declarative Approach to Bias in Concept Learning," *Proceedings 6th AAAI* (Menlo Park, California, 1987), 505–510.
8. *Standard MUMPS Pocket Guide* (Silver Spring, Maryland: M Technology Association, 1990).
9. R. Agrawal et al., "An Interval Classifier for Database Mining Applications," in *Proceedings of the 18th VLDB*, (Vancouver, British Columbia, 1992) 560–573.

Appendix

```
MINING ;Charles Williams; 18 Apr 1994
; Data Dictionary at label DD
MAIN K ^RULES
     S START=1,END=$O(^TABLE("FIELDS",""),-1)
     F P=START:1:END     I '$D(^TABLE("FIELDS",P,"N")) D
     . F Q=START:1:END D:Q'=P&'$D(^TABLE("FIELDS",Q,"N")
     .. F R=START:1:END I R'=P,R'=Q,'$D(^TABLE("FIELDS",R,"N") S RECORD=0
..........F  S RECORD=$O(^DATA("RECORDS",RECORD)) Q:RECORD=""  D
     ... S SUB1=Q_"="_$P(^DATA("RECORDS",RECORD),"",Q)
     ... S SUB2=R_"="_$P(^DATA("RECORDS",RECORD),"",R)
     ... S SUB3=$P(^DATA("RECORDS",RECORD),"",P)
     ... Q:SUB1=""!(SUB2=""!(SUB3=""))
     ... S
^RULES(P,$$convert(SUB1),$$convert(SUB2),$$convert(SUB3))=+$G(^RULES(P,$$convert(SUB1),$$convert(SUB2),$
$convert(SUB3)))+1
     D CLEANUP    ;Cleanout rules below threshold
MAINQ Q1
CLEANUP           ;Cleanout any rules that are below the ;threshold
     S S1=""
     F  S S1=$O(^RULES(S1)) Q:S1=""  S S2="" D
     . F  S S2=$O(^RULES(S1,S2)) Q:S2=""  S S3="" D
     .. F  S S3=$O(^RULES(S1,S2,S3)) Q:S3=""  S S4="" D
     ... F  S S4=$O(^RULES(S1,S2,S3,S4)) Q:S4=""  D
     .... K:^RULES(S1,S2,S3,S4)<^TABLE("THRESHOLD")
^RULES(S1,S2,S3,S4)
CLEANUPQ Q
convert(string)     ;Routine to convert upper case to lower case
     S upper="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
     S lower="abcdefghijklmnopqrstuvwxyz"
     S outstr=$TR(string,upper,lower)
convert   q Q outstr
     ;
```

Figure 1. The main routine for knowledge discovery.

```
REPORT ;Charles Williams;11:59 PM 6 Apr 1994
     ;This is the entry point for the user interface to an
     ;Artificial Intelligence Query System. This program uses a
     ;database set up by the MAIN entry point.
     ;The program will allow data entry of record information and
     ;will perform queries on fields that are not filled in.
     W #!!,"Welcome to the Williams' Data Entry Query System."
     W !,"Please fill in the following information. If you do not"
     W !,"know an answer, leave it blank or enter ""?"" and the system"
     W !,"will suggest an answer for you!"
     S FLDNUM=0
     F S FLDNUM=$O(^TABLE("FIELDS",FLDNUM)) Q:FLDNUM="" D GETDAT
     R !!,"Press any key to continue.",ans#1
     W #,"You Entered the following data:",!
     S  FLDNUM =0
     F S FLDNUM=$O(^TABLE("FIELDS",FLDNUM)) Q:FLDNUM="" W !,^(FLDNUM)_": ",$TR(RECORD(FLDNUM),"?")
     W !,"Would you like to start over and re-enter your data? "
     R YN#1
     G:YN?1A&("Yy"[YN) REPORT
     S FLDNUM=0
     F  S FLDNUM=$O(RECORD(FLDNUM)) Q:FLDNUM=""  I RECORD(FLDNUM) =""!(RECORD(FLDNUM)["?") D LOOKUP
     ;Look up the necessary fields!
     W #,!!,"Your final record is :"
     F  S FLDNUM=$O(^TABLE("FIELDS",FLDNUM)) Q:FLDNUM=""  W!,^(FLDNUM)_": ",$TR(RECORD(FLDNUM),"?")
REPORTQ Q
GETDAT    ;Get the data from the user
     W !,^(FLDNUM)_": "
     R RECORD(FLDNUM)
     I
RECORD(FLDNUM)=""!(RECORD(FLDNUM)["?"),$P(^TABLE("FIELDS"),"")
[(","_FLDNUM_",") W !,$TR(^TABLE("FIELDS",FLDNUM),":")_" is a mandatory field." G GETDAT     ;Cannot skip
the mandatory fields
GETDATQ Q
LOOKUP    ;
     W #,"Now searching the "_$TR(^TABLE("FIELDS",FLDNUM),":")_" field"
     S (SUB1,STRONGEST)=0,ANSWER=""
     F  S SUB1=$O(^RULES(FLDNUM,SUB1)) Q:SUB1=""  S SUB2=""
..........F  S SUB2=$O(^RULES(FLDNUM,SUB1,SUB2)) Q:SUB2=""  D LOOK2
     I ANSWER'="" W !,""""_ANSWER_""" was the best fit for this
     field would you like it entered for you? " R YN#1
     S:YN?1A&("Yy"[YN) RECORD(FLDNUM)=ANSWER
LOOKUPQ Q
LOOK2 ;
     Q:$$convert(RECORD($P(SUB1,"=")))'=$P(SUB1,"=",2)
     Q:$$convert(RECORD($P(SUB2,"=")))'=$P(SUB2,"=",2)
     S ANS=""
     F  S ANS=$O(^RULES(FLDNUM,SUB1,SUB2,ANS)) Q:ANS=""  I ^(ANS)>
...........STRONGEST S STRONGEST=^(ANS),ANSWER=ANS
LOOK2Q    Q
```

Figure 2. The user-interface portion of the system.

```
DD    ;^RULES(FLDNUM,SUB1,SUB2,SUB3)=THRESHOLD, doubles as an index
      ;of rules derived from the records in the database
      ;The threshold indicates the strength of the formed rule
      ;START,END - first and last fields
      ;^TABLE("FIELDS",FLDNUM)=FLDNAME, name of fields
      ;^TABLE("FIELDS",FLDNUM,"N")=NULL, index
      ;^DATA("RECORDS"), physical records delimited by ""
      ;FLDNUM, numeric id of the field
      ;RECORD(FLDNUM), an array of the field data entered
      ;STRONGEST, contains the strongest threshold
      ;
```

Figure 3. The data dictionary.

We can add GUI (Graphical User Interface) Capabilities to your MUMPS Applications!

Specialists in the development of
MS Windows applications for MUMPS databases

Give us a call to discuss how we can bring your
MUMPS software into the 90's.

LCI

Software Development & Services

300 Lenora Street
Suite B-265
Seattle, WA 98121

206-998-2371
(FAX) 206-322-3715

Internet: LCI@eskimo.com

We also provide training and other custom software development.

CALIFORNIA PACIFIC

MEDICAL SERVICES
ORGANIZATION

PROGRAMMER (MUMPS)

California Pacific Medical Services Organization is a rapidly growing, nationally recognized, health care organization located in San Francisco.

To help us better meet the complex needs of our growing user community, we are adding an experienced development programmer to our IS team. Reporting to the division's Director, this position involves programming of user interfaces and specialized output and formulation of functional and technical documentation.

Requirements include at least two years M programming experience in a health care setting, BA/BS degree in Computer Science or Information Systems, or an equivalent combination of education and work experience. Knowledge of IDX products is desirable.

As part of our team, you will play a critical role in the future of this innovative organization. We offer an excellent salary and benefits package.

For immediate consideration, fax or send your resume and salary requirements to:

Director, MIS
CPMSO
1388 Sutter St., Suite 400, San Francisco, CA 94109
Fax: 415 749-4800