

FINDING RULES IN DATA

BY BEVERLY THOMPSON AND WILLIAM THOMPSON

An algorithm for extracting knowledge from data

MUCH OF THE FOCUS of computing in the past has been on collecting, manipulating, and disseminating data. Many people are now saying that the primary focus of computing in the future will be on the collection, manipulation, and dissemination of knowledge and that our view of it will be profoundly changed in the process. Much artificial intelligence research to date has been concerned with representing knowledge in a way that can be efficiently collected, stored, and utilized by a computer.

In this article we describe one method of obtaining knowledge directly from a set of data. This knowledge will be represented in a series of if-then statements called rules. The method used, the ID3 algorithm, was developed by J. Ross Quinlan (reference 1) and is the method most commonly used in the commercial expert systems that employ induction methods to generate rules.

CLASSIFICATION TREES

One structure that has been extensively used to represent knowledge is the classification tree (also called the "decision tree"). A simple example is the best way to show how a classifica-

tion tree works. Suppose that you want to invest money in a company in the computer industry and are seeking advice from a friend who is a financial expert in that industry. When you call him on the phone, something like the following conversation may take place:

Expert: Is the company a hardware or software company?

You: Software.

Expert: Would you say that the company's main product is new, in midlife, or old technology?

You: Midlife.

Expert: Does this product have any significant new competition?

You: No.

Expert: From what you've told me, it seems that the company's profits should continue to go up.

Figure 1 shows how this same exchange could be represented as a classification tree. This partial tree completes only the branch of the tree that represents the answers you supplied during your conversation. A complete tree would fill in all of the questions and answers that could possibly take place during a consultation session.

Although trees show the relationships that exist among the various components, they can be very difficult to manipulate. One structure that can represent similar information but is easier to use is called a rule. The rule that can be made from the tree in figure 1 is this:

If type is software
 and age is midlife
 and competition is no
 then profit is up.

One rule is made to represent each completed branch of the tree, with the subject of each question being represented by a keyword called an attribute. The question associated with the attribute can be stored along with the rule in the form of a prompt. An example of a prompt would be

prompt type

Is the company a hardware or software company?

The entire collection of rules and

(continued)

Beverly and William Thompson are consultants specializing in the design of knowledge-based systems for microcomputers. They can be contacted at MicroExpert Systems, R.D. 2, Box 430, Nassau, NY 12123.

prompts is called a knowledge base.

Each rule is a single fact that can be easily verified or modified. In addition, work done on expert system shells provides us with many excellent methods for using a set of rules to conduct a consultation. For a detailed description of one of these methods (called a backward-chaining inference engine), see our article "Inside an Expert System" in the April 1985 BYTE.

THE KNOWLEDGE ACQUISITION BOTTLENECK

If the problem of selecting a winning company were as easy as our example makes it appear, there would be no problem stating all of the knowledge about the subject in a simple set of rules and we could make a fortune in the stock market. Unfortunately, the complexity of real-world problems often makes it difficult to design a detailed set of rules. In some problem areas the amount of information needed for a solution is prohibitively large. In others, the knowledge is not well enough defined to put into rules. Even in cases where the problem is manageable, the number of experts with the inclination and the time to work on these systems is small. This situation is often referred to as the knowledge acquisition bottleneck.

In order to solve the problem of acquiring expertise, we should ask ourselves how the experts became experts in the first place. Why, for example, did our financial analyst ask those specific questions? People learn through their experiences. The financial analyst, for instance, constantly absorbs data about different companies, their products, and their financial situations. His mind has the ability to observe patterns in data and

organize it. This process allows a person to extract meaning and thus knowledge from data. As we said, the computer has revolutionized the collection and storage of data, but have we really been able to make the most use of that data? Shouldn't it be possible to find some way to organize data to recognize patterns and extract knowledge directly from it? The ID3 algorithm attempts to do just that. No one claims that it works in any way like our own brains, but it does provide a way to produce a classification tree directly from a set of examples within a problem area. Once we can make a classification tree, it is a direct step to rules that can be manipulated by an expert system shell.

To illustrate how the algorithm works, let's return to the problem of predicting whether a given company's profits will increase or decrease. This time, let's suppose that when you ask your friend for advice he tells you that he makes it a policy not to give financial advice to friends. Instead, he suggests some magazines that you could read to familiarize yourself with the ups and downs of the industry. You take his advice but find it's very difficult to make use of all the reading material. So you make a table that lists some of the companies, some facts about them, and whether their profits have increased or decreased in the last quarter. A sample of this table is shown in table 1. The labels on the columns "profit," "age," "competition," and "type" are the attributes for which the values are stored in the table. Each row in the data table is called an example. The first attribute, "profit," is called the class attribute. Your goal is to determine a relationship between the class attribute and the values of the other attributes. The

first example says that "profits were down in a company whose product was old, had no significant competition, and which produces software." You can see that the table alone does not give you much insight into predicting when a company's profits are likely to increase. What is needed is a way to use the examples in the table to produce a classification tree.

BUILDING THE CLASSIFICATION TREE

To build a classification tree, you select one of the attributes to be the starting point or root node of the tree. Once you select this attribute, you split up (partition) the example set into a number of smaller tables, each containing examples with the same value of the selected attribute. If you select "age" as the root of the tree, the table will be split into the sets shown in figure 2. You can see in figure 2 that when "age" has the value "old," the value of the class, "profit," is always "down." When the value of "age" is "new," the class value is always "up." In these two new example sets, no further classification is necessary. However, in the example containing "age = midlife," you must select a new attribute and split the set again. Figure 3 shows the results of a split on the attribute "competition." Since each partition now contains only a single value for the class attribute, the tree-building process is complete.

You can produce a set of if-then rules from this tree by following each branch from the root to a terminal node. Each rule is a series of conditions consisting of attribute and value pairs, followed by a single conclusion that contains the class and the corresponding class value. The inter-

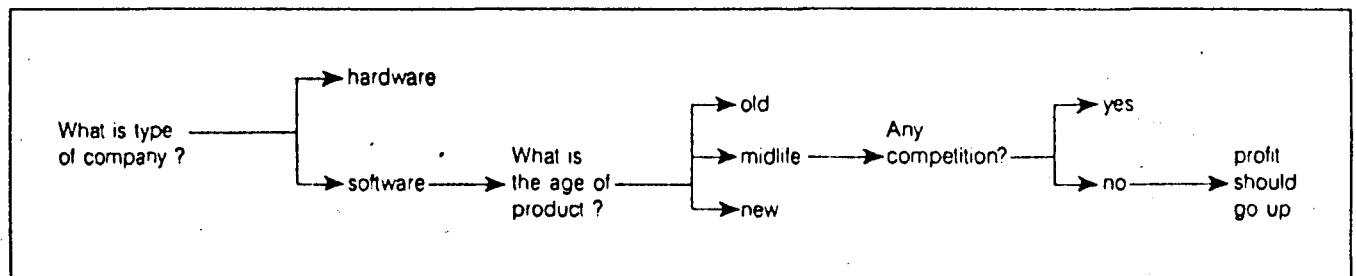


Figure 1: Classification tree showing the results of one consultation session.

FINDING RULES

mediate nodes and their branches form the conditions of the rules; the terminal nodes form the rules' conclusions. For example, following the first branch on the right results in the rule

if age is old
then profit is down.

One rule is produced for each terminal node of the tree. The rules that can be formed from this tree are shown in figure 4.

Figure 3 illustrates an interesting side effect of the tree-building process. Even though the original example set contained three attributes, you did not need the attribute "type" to classify the examples in the set. This is a valuable result because it can reduce the amount of data that needs to be collected. Brieman et al. (reference 2) used a classification-tree-building technique called CART to classify the mortality risk of heart attack victims. This process allowed them to reduce the number of attributes in the data set from 19 to 3.

WHY THIS TREE?

The tree in figure 3 is certainly not the only possible tree that could have been generated from this set of examples. Rather than selecting "age" as the first attribute on which to split the example set, you could have selected

(continued)

Table 1: Table of example set.

Profit	Age	Competition	Type
Down	Old	No	Software
Down	Midlife	Yes	Software
Up	Midlife	No	Hardware
Down	Old	No	Hardware
Up	New	No	Hardware
Up	New	No	Software
Up	Midlife	No	Software
Up	New	Yes	Software
Down	Midlife	Yes	Hardware
Down	Old	Yes	Software

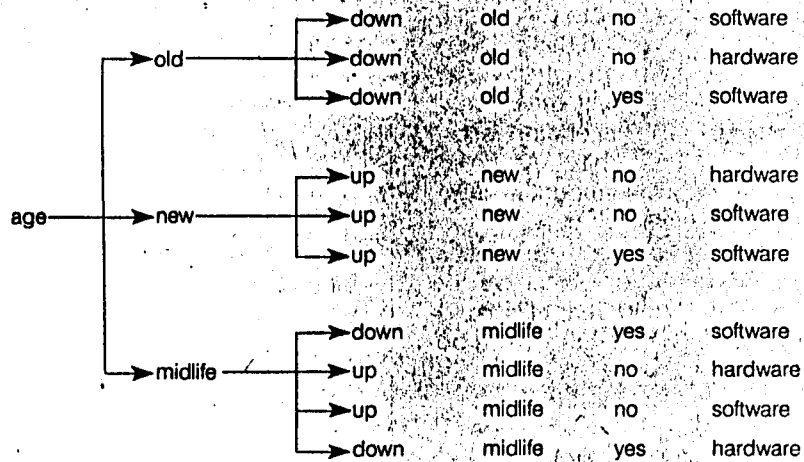


Figure 2: Example set from table 1 split on attribute "age."

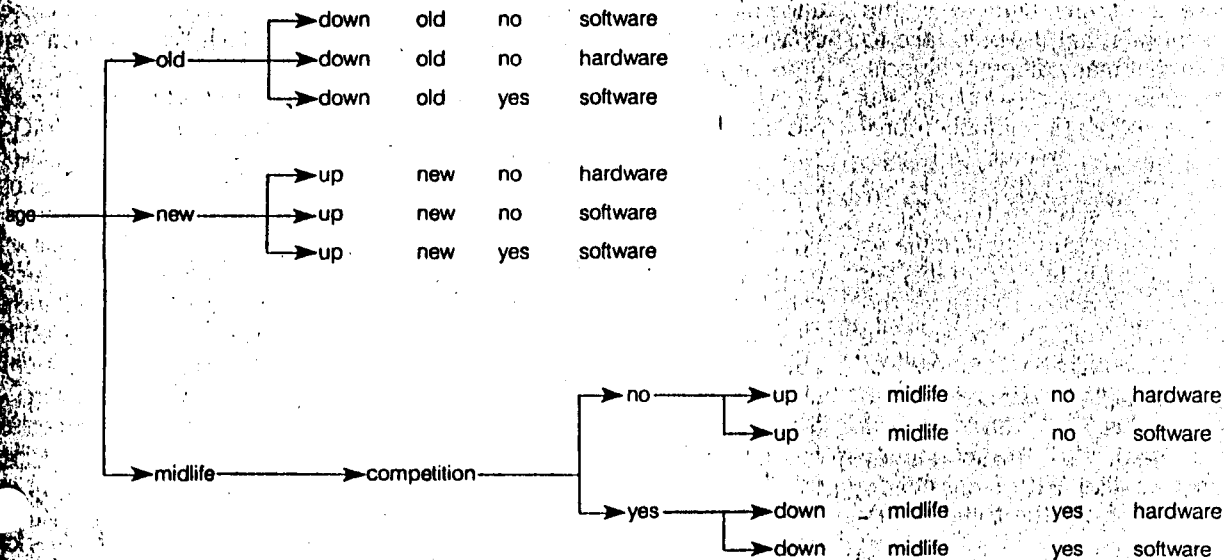


Figure 3: Example set of figure 2 after second split—this time on "competition."

one of the other attributes. This would have resulted in a different tree and a different set of rules. Since some attributes tell us more about how to classify an object than others (in our example, "type" was not even needed). It is important to split the example set using attributes that lead to efficient classification trees. Put another way, how do you measure the amount of information about classification contained in a single attribute?

ENTROPY

If we turn to communication theory, we find that there is a precise measure of information called entropy. Applying this concept to the classification problem, we find that if an object can be classified into one of several different groups, the entropy is a measure of the uncertainty of the classification of that object. As the entropy increases, the amount of information that we gain by knowledge of the final classification increases. Mathematically, if an object can be classified into N different classes, c_1, \dots, c_N , and the probability of an object being in class i is $p(c_i)$, then the entropy of classification, $H(C)$, is

$$H(C) = -\sum_{i=1}^N p(c_i) \log_2 p(c_i)$$

If you are a bit rusty on logarithms, recall that $\log_2(x) = y$ means the same as $2^y = x$ or, more plainly, the log to the base 2 of any number is the number of bits that it would take to represent that many different objects. Consequently, $\log_2(16) = 4$ tells you that it takes 4 bits to uniquely represent 16 different objects. (All logarithms mentioned in this article are assumed to be taken to base 2.)

Let's apply the entropy formula to the example set in table 2. In this set there are two possible values for the class attributes, "up" and "down." The probability (actually the frequency of occurrence) of the class having the value of "up" is 5 out of a total sample set of 10, or 5/10. The probability of "down" is also 5/10. The entropy of classification for the total set is

$$H(C) = -p(\text{up}) \log p(\text{up}) - p(\text{down}) \log p(\text{down}) = -5/10 \log_2(5/10) -$$

If age is old
then profit is down.
If age is new
then profit is up.
If age is midlife
and competition is no
then profit is up.
If age is midlife
and competition is yes
then profit is down.

Figure 4: Rules produced from the classification tree of figure 3.

Table 2: Part of the example set split on "competition."

Competition	Profit
No	Down
No	Up
No	Down
No	Up
No	Up
No	Up
Yes	Down
Yes	Up
Yes	Down
yes	Down

$$5/10 \log_2(5/10) = 1.00$$

Although this number represents the uncertainty about profits going up or down based on the data in table 1, it does not tell us anything about the amount of information contained in the individual attributes.

CALCULATING THE ENTROPY OF CLASSIFICATION OF AN ATTRIBUTE

What we really want to know to help us decide the attribute on which to split is the entropy of classification after deciding on a particular attribute. This entropy represents the amount of uncertainty about a particular outcome, so we'll want to split on the attribute that results in the smallest entropy of classification.

The first step in calculating the entropy of classification after deciding on a partitioning attribute, symbolized by $H(C|A)$, is to split the table into subtables where each example has

the same value of the partitioning attribute. Table 2 shows a partition of the example set after splitting on the attribute "competition." The entropy of each subtable, $H(C|a_j)$, is calculated for each value of the attribute, a_j . $H(C|a_j)$ is given by the expression

$$H(C|a_j) = -\sum_{i=1}^M p(c_i|a_j) \log p(c_i|a_j)$$

The function $p(c_i|a_j)$ is the probability that the class value is c_i when the attribute has its j th value.

We can now calculate the entropy of each subtable:

$$H(C|\text{competition=no}) = -p(\text{up}|\text{competition=no}) \times \log p(\text{up}|\text{competition=no}) - p(\text{down}|\text{competition=no}) \times \log p(\text{down}|\text{competition=no}) = -4/6 \log(4/6) - 2/6 \log(2/6) = 0.918$$

$$H(C|\text{competition=yes}) = -p(\text{up}|\text{competition=yes}) \times \log p(\text{up}|\text{competition=yes}) - p(\text{down}|\text{competition=yes}) \times \log p(\text{down}|\text{competition=yes}) = -1/4 \log(1/4) - 3/4 \log(3/4) = 0.811$$

The expression $p(\text{up}|\text{competition=no})$ is the probability that the class value is "up" when the value of the attribute "competition" is "no." It is just the number of times class "up" appears in a row with "competition" equals "no" divided by the total number of cases in which "competition" equals "no."

In order to find the entropy of the entire table after the split, $H(C|\text{competition})$, we must take the sum of the entropy of each of the values of the attribute multiplied by the probability that the value will appear in the table. Stating all of this concisely, the entropy of classification after choosing a particular attribute, $H(C|A)$, is the weighted average of the entropy for each value a_j of the attribute.

Mathematically this is expressed as

$$H(C|A) = \sum_{j=1}^M p(a_j) H(A|a_j)$$

M is the total number of values for the attribute A .

In this example this gives

(continued)

```
( ( down old no software )
  ( down midlife yes software )
  ( up midlife no hardware )
  ( down old no hardware )
  ( up new no hardware )
  ( up new no software )
  ( up midlife no software )
  ( up new yes software )
  ( down midlife yes hardware )
  ( down old yes software ) )
```

Figure 5: Lists used to represent the example set.

```
( ( profit down up )
  ( age old midlife new )
  ( competition no yes )
  ( type software hardware ) )
```

Figure 6: List used to store attributes and their values.

```
( age ( old classify ( ( ( down old no software )
                        ( down old no hardware )
                        ( down old yes software ) ) ) )
      ( new classify ( ( ( up new no hardware )
                        ( up new no software )
                        ( up new yes software ) ) ) )
      ( midlife classify ( ( ( down midlife yes software )
                          ( up midlife no hardware )
                          ( up midlife no software )
                          ( down midlife yes hardware ) ) ) ) )
```

Figure 7: The list equivalent to the tree in figure 2. This list is returned after the first call to the classify routine.

```
( age ( old ( profit ( down ) ) )
      ( new ( profit ( up ) ) )
      ( midlife ( competition ( no classify ( ( ( up midlife no hardware )
                                                ( up midlife no software ) ) )
      ( yes classify ( ( ( down midlife yes software )
                        ( down midlife yes hardware ) ) ) ) ) )
```

Figure 8: The list equivalent to the tree in figure 3.

```
( age ( old ( profit ( down ) ) )
      ( new ( profit ( up ) ) )
      ( midlife ( competition ( no ( profit ( up ) ) )
                             ( yes ( profit ( down ) ) ) ) ) )
```

Figure 9: The final list returned by the classify routine.

$$H(C|competition) = 6/10 \times 0.918 + 4/10 \times 0.811 = 0.8752$$

If we perform these same calculations for the other attributes in our example, we find that $H(C|age) = 0.4$ and $H(C|type) = 1.0$. Since $H(C|age)$ gives us the smallest entropy and thus the least uncertainty, "age" is the best attribute to select for the initial split.

IMPLEMENTING THE ALGORITHM

To implement this algorithm we need to make some decisions about how to represent the example tables in the program. We will store the table of examples as a list. Each element of the list is another list that contains one example. Figure 5 shows the list containing the example set from table 1. Each list is enclosed in parentheses. Even

though considerable overhead is associated with the use of list structures in a program, lists provide a great deal of flexibility and allow us to use a single representation for both the examples and the final classification tree.

We will also use a list to keep track of the attributes and their values. This is another list of lists. The first sublist consists of the class name and associated class values. The other sublists contain the attribute names and each attribute's values. Figure 6 shows the attribute list for the example set.

The actual tree-building procedure is performed by a function called classify. If you pass an example list to this function, it returns the classification tree for that example set. This tree is also represented by a list. The first element in the list is an attribute or class name, and it is followed by a series of lists. Each list contains a class value if the first item was the class name; otherwise, each list contains a value for the attribute followed by the tree produced by classifying the partition of the example set that has that value. In other words, classify is a recursive procedure that either returns the class name and the class value of the example set or calls itself to classify the new partitioned example set. The clearest way to explain classify is to demonstrate how it would process our example set.

Figure 7 shows a list that is equivalent to the tree returned by classify. By calculating $H(C|A)$ for each attribute, classify has chosen "age" as the attribute on which to split. It returns the attribute name followed by three lists. Each list contains a value of the attribute followed by the classification of the appropriate subset of examples. The calls to classify in the first two lists (the values "old" and "new") will result in no further recursion because each of the lists contains only a single class. In both cases, classify will return the class name, "profit," and the appropriate class value. The third list requires a new partitioning of the example set. Figure 8 shows the results when classify splits the new example set on "competition." The final tree returned

(continued)

by classify is shown in figure 9. If you examine the "list of lists" in figure 8, you can see that its structure is similar to the classification tree shown in figure 3.

The classify function produces the tree in a depth-first manner. In our example that means classify would continue to the end of the branch for "age" is "old" before looking at the other values for "age."

HANDLING CONFLICTS IN THE DATA

Conflicts among the examples can lead to the generation of erroneous rules. A conflict occurs when two examples contain identical values for all attributes but have different class values. A conflict usually signifies that the attributes chosen are inadequate for the classification task. You can remove this problem by introducing additional attributes. Deciding which new attributes to include is a task for an expert in the problem domain

being considered, but identifying conflicts is relatively easy. Since each example is stored as a list, you can recognize a conflict by comparing the list representing an example against each of the other lists. A conflict occurs when the tails (the entire list except for the first item) of two lists match, but the first items on the list are different.

"DON'T CARE" VALUES

When creating examples, we find it useful to specify that a particular attribute does not play a role in the classification. We use a special symbol, called a "don't care" value, to indicate this fact. For instance, (down old no *) indicates that if the value of the attributes "age" and "competition" are "old" and "no," respectively, the class value is "down" no matter what the value of "type." An asterisk represents a "don't care" value. Examples containing "don't care" values are expanded into a new set of examples,

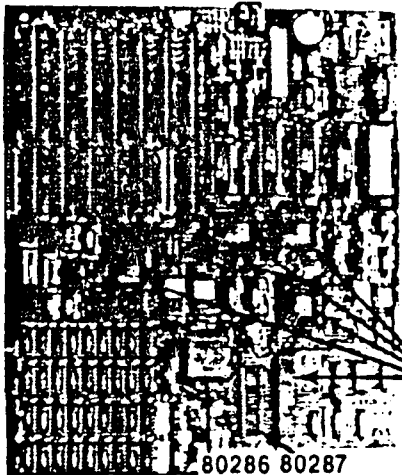
each containing one of the possible values for the "don't care" attribute. The example above would be expanded to (down old no hardware) and (down old no software).

ATTRIBUTES WITH NUMERIC VALUES

All of the attributes in the example set had values chosen from a limited group of possibilities. Suppose that instead of assigning the values "old," "new," and "midlife" to the attribute "age," you wanted to assign numeric values. In this case, you would use the numeric values of "age" to create a set of new attributes that contain the ranges of possible values. For example, assume that you have four companies with products aging 6, 10, 16, and 36 months. You would make three new attributes, each representing the range formed by splitting the consecutive values at their midpoints. The new attributes would be "age < 8,"

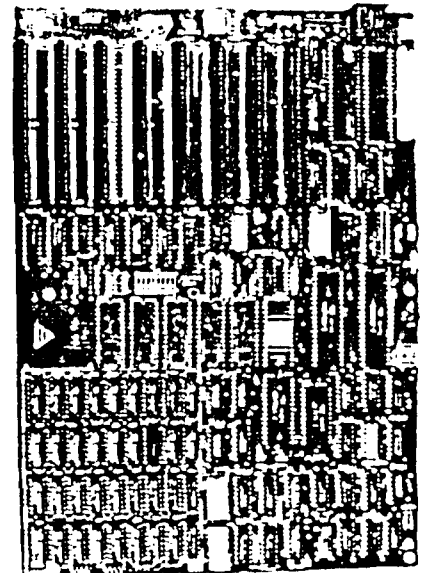
(continued)

WiseTEK INT'L, INC.



← TURBO AT

- 80286 IBM Compatible
- 6 MHz, 8 MHz, or 10 MHz Keyboard Switch
- 5 CHIPS SET—(Lower Heat, Higher Quality)
- 0 or 1 Wait State
- 1 MB Memory
- On Board Battery
- 80287 Socket Ready
- 3MB with serial/parallel port on board
- XT size AT JR motherboard available



5VLS1 TURBO XT →

- 8088-2 IBM-XT Compatible
- 4.77 MHz or 8 MHz Keyboard Switch
- 4 Layer PC Board
- 640K Memory
- 8087 Socket Ready

WiseTEK International, Inc.
513 Valley Way
Milpitas, CA 95035

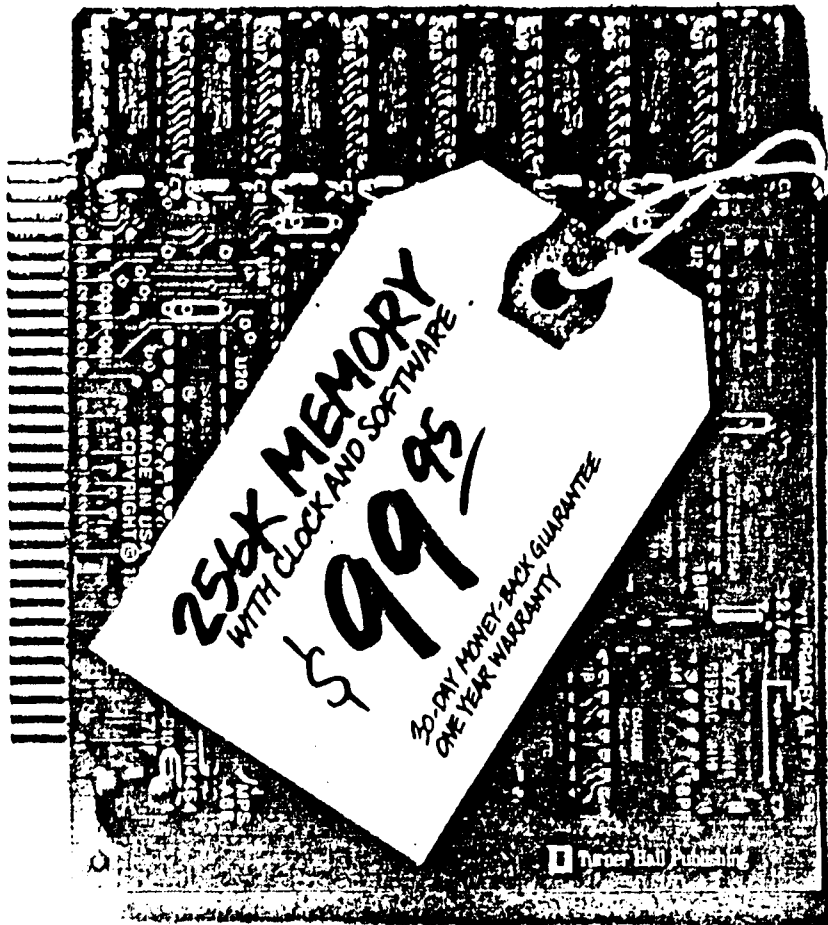
Micronic
108 East 16th Street
New York, NY 10003

See us at
COMDEX/Fall '86
November 10-14, 1986
Riviera Hotel
Las Vegas, Nevada
Booth #8421

*Distributors, Dealers,
OEM Welcome*

TEL: (408) 263-1237
(212) 529-4699
FAX: (408) 263-1870

ACTUAL SIZE.



ACTUAL PRICE.

Introducing The Turner Hall™ Card. The lowest priced complete 256K memory expansion board you can buy

We made it so inexpensive by using the very latest 256K RAM chips instead of four times as many 64K chips

That same technology makes the Card fit in a half-length PC/XT™ slot. And the reduced chip count increases reliability, so we can offer a 30-day money-back guarantee and 1-year warranty

The Card comes with a clock/calendar with replaceable battery backup, illustrated Owner's Manual, and software including clock, print spooler, and disk emulator. That's everything the most popular

multifunction boards have. Except a couple of extra ports and a lot of extra cost

IBM® or Compaq® owners will find the Card remarkably easy to install. And if you have any questions after you buy, call our Help Hotline

The Turner Hall Card is just \$99.95,* plus \$2.00 shipping (\$12.00 outside of U.S.A.).

Order by phone. We accept MasterCard or Visa. Or send us a check or money order with your business card attached.

Turner Hall Publishing
10201 Torre Ave., Cupertino, CA 95014

1-800-556-1234 x526.
(In CA 800-441-2345 x526).

*CA residents add 7% sales tax (\$7.00). Requires IBM PC, PC/XT, Portable PC, or Compaq with at least 256K of memory. Turner Hall is a trademark of Turner Hall Publishing. IBM is a registered trademark and PC/XT is a trademark of International Business Machines Corp. Compaq is a registered trademark of Compaq Computer Corp.

FINDING RULES

"age < 13." and "age < 26." The values in the original table under "age" are used to determine if the value of the new attributes is "yes" or "no." The entropy of classification after splitting is calculated for each new attribute along with the other attributes to determine the best attribute to use to partition the data set.

THE COMPLETED PROGRAM

A Turbo Pascal program called INDUCE for the IBM PC and compatibles implements the techniques described above. [Editor's note: INDUCE is available on disk, in print, and on BIX; see the insert card following page 352 for details. It is also available on BYTEnet; see page 4.] The program contains a collection of low-level routines for the manipulation of the list structures described in this article. Since Turbo Pascal is not optimized for list processing, the program is slow compared to some of the commercial implementations of the ID3 algorithm, but we hope that by examining the commented source code, programmers can gain some insight not only into the ID3 algorithm but also into the power of symbolic computation ← using list-like structures.

The program produces rules in the format accepted by the expert system shell MicroExpert, but you can easily modify the program to produce rules for another expert system shell. You can also modify the program to produce Prolog sentences.

Also available is Crossref, a parser that reads and parses rules that use the same rule syntax as the one we have described. A description of this program and how to write an inference engine that uses the rules can be found in our April 1985 article. [Editor's note: Crossref is available on BIX and BYTEnet.] ■

REFERENCES

1. Quinlan, J. Ross "Learning Efficient Classification Procedures and Their Application to Chess End Games." In *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski et al., eds. Palo Alto, CA: Tioga Publishing Co., 1983.
2. Brieman, Leo, et al. *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group, 1984.