AAAI-2000, July 28-Aug 3 2000, Austin, Texas, USA.

Intuitive Representation of Decision Trees Using General Rules and Exceptions

Bing Liu, Minqing Hu and Wynne Hsu

School of Computing
National University of Singapore
Lower Kent Ridge Road, Singapore 119260
(liub, huminqin, whsu)@comp.nus.edu.sg

Abstract

Producing too many rules is a major problem with many data mining techniques. This paper argues that one of the key reasons for the large number of rules is that an inefficient knowledge representation scheme has been used. The current predominant representation of the discovered knowledge is the if-then rules. This representation often severely fragments the knowledge that exists in the data, thereby resulting in a large number of rules. The fragmentation also makes the discovered rules hard to understand and to use. In this paper, we propose a more efficient representation scheme, called *general rules & exceptions*. In this representation, a unit of knowledge consists of a single *general rule* and a set of *exceptions*. This scheme reduces the complexity of the discovered knowledge substantially. It is also intuitive and easy to understand. This paper focuses on using the representation to express the knowledge embedded in a decision tree. An algorithm that converts a decision tree to the new representation is presented. Experiment results show that the new representation dramatically simplifies the decision tree. Real-life applications also confirm that this representation is more intuitive to human users.

Introduction

Much of the existing data mining research has been focused on designing efficient techniques to mine regularities or knowledge from databases. The discovered knowledge is commonly represented as a set of if-then rules. An if-then rule is a basic unit of knowledge.

One of the major problems with many existing mining techniques is that they often produce too many rules, which make manual inspection and analysis very difficult. This problem has been regarded as a major obstacle to practical applications of data mining techniques. It represents a major gap between the results of data mining and the understanding and use of the mining results. This paper aims to bridge this gap by focusing on simplifying the mining results so that they can be easily analyzed and understood by the human user.

In the past few years, a number of techniques have been proposed to deal with the problem of too many rules (e.g., Piatesky-Shapiro & Matheus 1994; Silberschatz & Tuzhilin 1996; Liu & Hsu 1996; Liu, Hsu & Ma 1999). The main idea of these techniques is to use the user's domain knowledge or some statistical measures to filter out those uninteresting rules. This work does not research in this direction. Instead, it identifies an important cause of the problem, and deals with the problem at its root.

We show that one of the key reasons for the large number of rules is that an inefficient scheme is used to represent the discovered knowledge. The simple if-then rules based representation has two major deficiencies:

1. If-then rules *fragment* the knowledge that exists in data, which results in a large number of discovered rules. This large number of rules presents three problems for data mining applications:
   - Manual analysis of the rules is very difficult.
   - Even if some interesting rules can be identified, they may not be actionable because the rules are often too specialized (cover too small a population) due to the fragmentation.
   - The fragmentation obscures the essential relationships in the domain and the special cases (or exceptions). This makes the discovered knowledge hard to understand and to use.

   We will use an example to illustrate the points later.
2. The discovered knowledge is represented only at a single level of detail. This flat representation is not suitable for human consumption because we are more used to hierarchical representation of knowledge. Hierarchical representation allows us to easily manage the complexity of knowledge, to view the knowledge at different levels of details, and to focus our attention on the interesting aspects.

We use a decision tree classification example to illustrate the problems, and to introduce the new representation. Although a decision tree, as its name suggests, is in the form of a tree, it essentially expresses a set of if-then rules. Our example data set has two attributes (A1 and A2), 500 data records and two classes, • and **O**. The value range of A1 is from 0-100 and A2 from 0-50. The decision tree system C4.5 (Quinlan 1992) is used to build the tree (for classification), which basically partitions the data space into different class regions. The resulting partitioning is expressed as a decision tree. Figure 1 shows the data space and the partitioning produced by C4.5. The resulting tree (not shown here) has 14 leaves, represented by the 14 rectangular regions. The two classes of data are well separated. The 14 regions or the tree can be conveniently represented as a set of if-then rules, with each tree leaf (or a region) forming a single rule:

R1: A1 ≤ 49, A1 > 18 → •
R2: A1 <= 10 → •
R3: A1 > 10, A1 ≤ 18, A2 > 18→ •
R4: A1 > 10, A1 ≤ 18, A2 <= 10 → •
R5: A1 > 10, A1 ≤ 18, A2 > 10, A2 ≤ 19 → **O**
R6: A1 > 86 → **O**
R7: A1 > 49, A1 ≤ 60 → **O**
R8: A1 > 60, A1 ≤ 86, A2 ≤ 10 → **O**
R9: A1 > 60, A1 ≤ 86, A2 > 39→ **O**
R10: A1 > 60, A1 ≤ 68, A2 > 20, A2 ≤ 39 → **O**
R11: A1 > 68, A1 ≤ 86, A2 ≤ 30, A2 > 10 → **O**
R12: A1 > 68, A1 ≤ 80, A2 > 30, A2 ≤ 39 → **O**
R13: A1 > 60, A1 ≤ 68, A2 > 10, A2 ≤ 20 → •
R14: A1 > 80, A1 ≤ 86, A2 > 30, A2 ≤ 39 → •

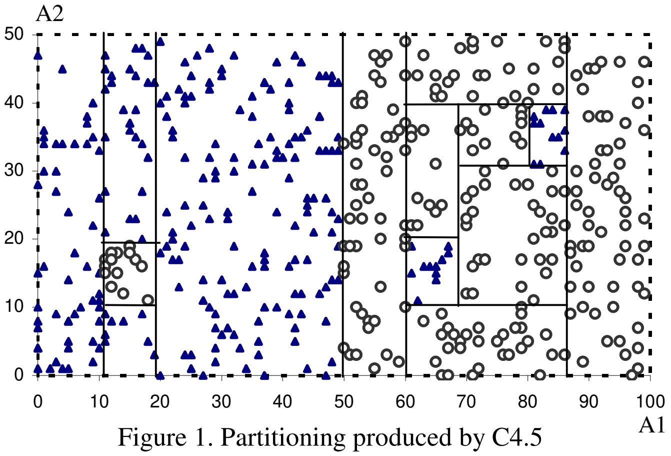


Figure 1. Partitioning produced by C4.5

From Figure 1 and the 14 rules from the decision tree [1], the following observations are made:

- Looking at the 14 rules themselves does not give us a good overall picture of the regularities that exist in data. These rules fragment the knowledge. However, if we change the 14 rules to the following two *general rules & exceptions* (GE) patterns, the picture becomes clear:

GE-1: A1 ≤ 49 → • [sup = 47.2%, conf = 94.4%]
Except R5: A1 > 10, A1 ≤ 18, A2 > 10, A2 ≤ 19 → **O** [sup = 2.8%, conf = 100%]
GE-2: A1 > 49 → **O** [sup = 45.2%, conf = 90.0%]
Except R13: A1 > 60, A1 ≤ 68, A2 > 10, A2 ≤ 20 → • [sup = 2.4%, conf = 100%]
R14: A1 > 80, A1 ≤ 86, A2 > 30, A2 ≤ 39 → • [sup = 2.4%, conf = 100%]

The first part of a GE pattern is a general rule, and the second part (after "Except") is a set of exceptions to the general rule. From the GE patterns, we see the essential relationships of the domain, the two general rules, and the special cases, the exceptions (R5, R13 and R14). This representation is much simpler than the 14 rules because R1-R4 and R6-R12 are not needed [2]. Figure 2 gives a pictorial representation of the GE patterns. Our real-life application users also confirmed that the new representation is more intuitive, and conforms closely to the representation of knowledge in their daily work.

Note that we have used support (*sup*) and confidence (*conf*) for a rule, which have their usual meanings (Agrawal, Imielinski & Swami 1993). Both support and confidence are useful concepts in practical applications.

- The exception regions are the ones that cause the fragmentation. More exceptions result in more severe fragmentation. For example, in Figure 1, on the left of the A1 = 49 line, we have one exception region, and 5 rules were generated. On the right of A1 = 49, we have two exception regions, and 9 rules were generated.

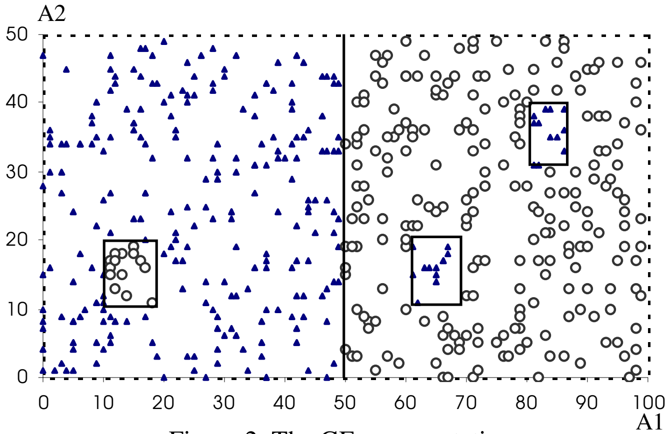


Figure 2. The GE representation

The GE patterns simplifies the rules (or decision tree) by:

using general rules and removing those fragmented rules that have the same classes as the general rules.

These fragmented rules are non-essential, and worse still they make the discovered knowledge hard to interpret. This is not to say that the fragmented rules are completely useless. For example, those fragmented rules may have higher confidences (e.g., 100%). They may be useful because of the high confidences. However, the GE patterns have given us a good summary of the knowledge. If the user is interested in the detailed rules, the general rules can direct him/her to them (using a simple user interface).

In the proposed GE representation, each combination of one general rule and a set of exceptions represents a single piece of knowledge. The general rule gives a basic relationship in the domain. The exceptions are unexpected with respect to the general rule. A general rule normally covers a large portion of the data, while an exception covers a relatively small portion of the data. In the next two sections, we will see that the GE representation naturally introduces a knowledge hierarchy, and that it can be expressed in the form of a tree, which we call the *GE tree*.

In the rest of the paper, we develop the idea further in the context of decision trees. An algorithm that converts a

[1] C4.5 also has a program to produce a set of classification rules. We do not use them in this paper as these rules can overlap one another, which make them more difficult to comprehend and to process. That program is also very inefficient for large data sets.

[2] The new representation does not affect the classification accuracy.

decision tree to a GE tree is presented. Experiment results and real-life applications show that the proposed method simplifies the decision tree dramatically and also produces a more intuitive representation of knowledge.

General Rules and Exceptions

Since this paper focuses on using the GE representation to express classification knowledge embedded in a decision tree, we define GE patterns in this context. A database D for classification consists of a set of data records, which are pre-classified into q (≥ 2) known classes, $C = \{c_1, \ldots, c_q\}$. The objective of decision tree building is to find a set of characteristic descriptions of the classes that can be used to predict the classes of future (unseen) cases.

Definition (GE patterns): A GE pattern consists of two parts, a single general rule (which is an if-then rule) and a set of exceptions. It has the following form:

$X \rightarrow c_i$ [*sup*, *conf*]
Except $E_1, \ldots, E_n$

where:

1. $X \rightarrow c_i$ is the *general rule*. X is a set of *conditions*, and c_i is the *consequent* (a class).
2. $E = \{E_1, \ldots, E_n\}$ is the set of *exceptions*. E may be empty ($E = \emptyset$). Each E_j is a GE pattern of the form:

   $X, L_j \rightarrow c_k$ [*sup*, *conf*]
   Except $E_{j1}, \ldots, E_{jm}$

   where ($X, L_j \rightarrow c_k$) is called a *sub-general rule* (if it has no exceptions, we also called it an *exception rule*). L_j is an additional set of conditions, and $c_k \in C$ and $c_k \neq c_i$. E_{jl} are the exceptions of ($X, L_j \rightarrow c_k$).

Notes about the definition:

1. For the sub-general rule, ($X, L_j \rightarrow c_k$), X may not appear explicitly in the conditional part, but needs to be satisfied. For example, if X is A1 > 49, then A1 > 80 can be the only condition of a sub-general rule because data records that satisfy A1 > 80 also satisfy A1 > 49.
2. An exception E_j only covers a subset of the data records covered by its general rule ($X \rightarrow c_i$). The class of E_j's sub-general rule must be different from its general rule.
3. Since each exception is also a GE pattern, this scheme can represent knowledge in a hierarchical fashion.

We now define the (sub-) general rule. We do not n
eed to define exceptions because they are GE patterns.

Definition ((sub-) general rules): A (sub-) general rule is a *significant rule*, and its class is the *majority class* of the data records *covered* by the rule. A rule covers a data record if the data record satisfies the rule's conditions.

It does not make sense that the class of the (sub-) general rule is not the majority class. For example, if our data set has two classes c_1 and c_2, then A1 > 2 $\rightarrow c_1$ with the confidence of 30% cannot be a general rule. For it to be one, its confidence must be greater than 50%.

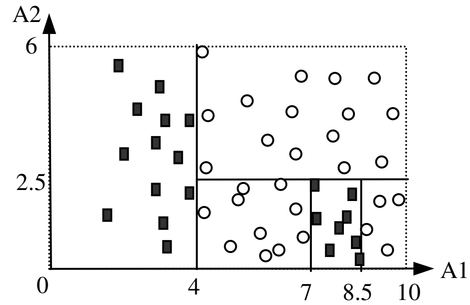
Rule significance: The significance of a rule can be measured in many ways. For example, we can measure it using statistical significance tests and/or minimum confidences (*minconf*) from the user. In this work, we use chi-square test/fisher's exact test [3] (Everitt 1977) and minconf. The way that we test the significance of a rule is similar to that in (Liu, Hsu & Ma 1999).

Representing Decision Trees as GE Patterns

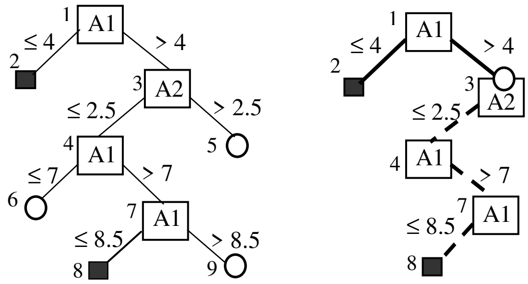
Decision tree construction (Quinlan 1992) is a popular method for building classification models. A decision tree has two types of nodes, *decision nodes* and *leaf nodes*. A *decision node* specifies some test to be carried out on an attribute value with one branch for each possible outcome of the test. A *leaf node* indicates a class.

From a geometric point of view, a decision tree represents a partitioning of the data space. A serial of tests (or cuts) from the root node to a leaf represents a hyper-rectangular region. The leaf node gives the class of the region. For example, the five rectangular regions in Figure 3(A) are produced by the decision tree in Figure 3(B). The tree can also be represented as rules. For example, the leaf node 5 in Figure 3(B) can be represented with the rule,

A1 > 4, A2 > 2.5 $\rightarrow$ **O**.



(A). Partitioning produced by the tree in Figure 3(B)



(B). Decision tree (C). GE tree

Figure 3. An example partitioning of the data space and its corresponding decision tree and GE tree

As mentioned earlier, the GE representation of a decision tree can also be in the form of a tree, which we call the *GE tree*. Below, we present the algorithm that finds general rules and exceptions in the context of a decision tree, or in

[3] Chi-square test can be computed very efficiently. However, it is based on the assumption that the expected frequencies should not be small (Everitt 1977). One popular rule of thumb for a 2x2 contingency table is that the chi-square test should not be used when the expected frequency of any cell is smaller than 5. When such a situation occurs, we use fisher's exact test, which is computationally less efficient. However, such cases do not occur often. The computation does not present a problem.

other words, converts a decision tree to a GE tree. The algorithm consists of two steps:

1. Find high-level general rules: The system descends down the tree from the root to find the nearest nodes whose majority classes can form significant rules. We call these rules the *high-level general rules*.
2. Find exceptions: After the high-level general rules are found, the system descends down the tree further to find exceptions. Since an exception is also a GE pattern, with its sub-general rule and exceptions, the question is how to determine whether a tree node should form a sub-general rule or not. We use the following criteria:
   - Significance: A sub-general rule must be significant and has a different class from its general rule.
   - Simplicity: If we use a tree node to form a sub-general rule, it should result in fewer rules in the final GE representation. The complexity of a GE representation is measured by the sum of the number of high-level general rules and the number of sub-general rules (or exception rules). Let the class of the rule *R* formed by the current node be c_j, and the class of its general rule (before it) be c_i ($i \neq j$). Let the number of leaves below this node in the tree for class c_i be n_i, and for class c_j be n_j. *R* will be a sub-general rule if the following condition is satisfied:

   $$n_j \geq n_i + 1 \qquad (1)$$

   The formula is intuitive because if the inequality holds it means that using *R* to form a sub-general rule with class c_j (1 in the formula represents this sub-general rule) cannot result in a more complex final description of the knowledge.

We now use an example to illustrate the idea. In Figure 3(A and B), the following GE patterns can be formed:

GE-1: A1 ≤ 4 → ■
GE-2: A1 > 4 → **O**
  Except A1 > 7, A1 ≤ 8.5, A2 ≤ 2.5 → ■

A1 ≤ 4 → ■, and A2 > 4 → **O** are high-level general rules (which are formed by node 2 and 3 in Figure 3(B)). For GE-2, we can have an alternative representation, i.e., forming a sub-general rule at node 7 (Figure 3(B)) with its exception (note that we cannot form a sub-general rule at node 4 because its majority class is the same as that of its general rule at node 3, i.e., **O**):

GE-2': A1 > 4 → **O**
  Except A1 > 7, A2 ≤ 2.5 → ■
    Except A1 > 8.5, A2 ≤ 2.5 → **O**

This representation (GE-2') is, however, more complex than GE-2. If we follow formula (1) above and compare the number of rules, we have

$$1 < 1 + 1$$

This means that we should not use GE-2'. Figure 3(C) gives the GE representation in the tree form. The two thick lines with classes represent the two general rules. The thick dash lines represent the exception of GE-2.

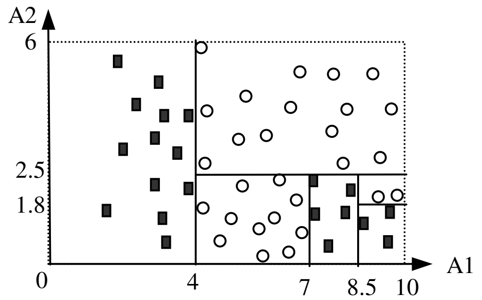
Now let us consider the situation in Figure 4. In this case, a sub-general rule should be formed at node 7 (according to the above formula, 2 ≥ 1 + 1):

GE-2: A1 > 4 → **O**
  Except A1 > 7, A2 ≤ 2.5 → ■
    Except A1 > 8.5, A2>1.8, A2 ≤ 2.5 → **O**

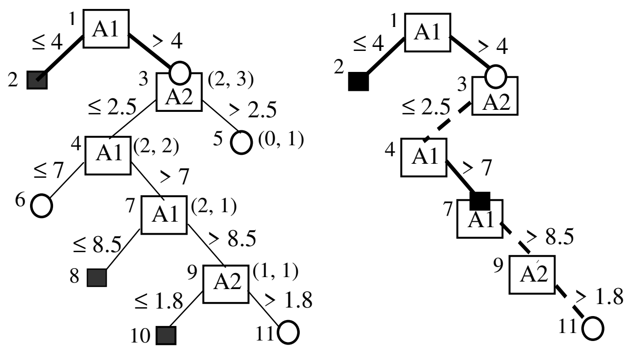
If we do not form a sub-general rule here, we will have:

GE-2': A1 > 4 → **O**
  Except A1 > 7, A1 ≤ 8.5, A2 ≤ 2.5 → ■
    A1 > 8.5, A2 ≤ 1.8 → ■

GE-2 is preferred because it gives us a hierarchy of knowledge and does not increase the knowledge complexity. Intuitively, we can also see that the area within (A1 > 7, A2 ≤ 2.5) is a general area for class ■.



(A). Partitioning produced by the tree in (B)



(B). Decision tree (C). GE tree

Figure 4. A partition of the data space and its corresponding decision tree, and GE tree

The detailed algorithm (findGE) is given in Figure 5. The algorithm first traverses down the tree along each branch recursively to find the high-level general rules (line 2, 5 and 6). In the case of Figure 4(B), it finds two general rules shown in Figure 4(B) and (C), i.e., the two thick lines from the root. The classes (■ or **O**) are also attached.

Algorithm findGE(*Node*)
1 **if** *Node* is a leaf node **then**
2   mark it as a high-level general rule
3 **else** **for** each child node *N* of *Node* **do**
4   c_i = majority class of node *N*;
5   **If** the rule formed from root node to *N* with the class c_i is significant **then**
6   mark it as a high-level general rule;
7   countLeaves(*N*);
8   findExcepts(*N*, c_i)
9   **else** findGE(*N*)
10   **end**

Figure 5: Finding general rules and exceptions

After a high-level general rule is found (line 5), the algorithm goes down the tree further to find its exceptions (line 8). This is carried out by the findExcepts procedure.

As discussed above, to decide whether a node below a general rule should form a sub-general rule or not, we need the numbers of leaves of different classes below the node (formula (1)). The countLeaves procedure (line 7 of Figure 5) performs this task. This procedure is not given here as it is fairly straightforward. In the case of Figure 4(B), the procedure produces the numbers of leaves of different classes below each internal node. These numbers are shown within "()" in Figure 4(B). The first number is the number of ■ class leaves below the node, and the second number is the number of **O** class leaves.

Finally, the procedure findExcepts is given in Figure 6.

```
Procedure findExcepts(Node, c_i)
1   if Node is a leaf then
2      if the class of the leaf is different from c_i then
3         mark it as an exception
4      else   delete the node      /* it is a fragmented leaf or rule */
5   else   for each child node N of Node do
6            c_j = majority class of node N;
7            if c_j ≠ c_i  AND  the rule formed from root node to
                    N with the class c_j is significant  AND
                    n_j ≥ n_i + 1 then
8               mark it as a sub-general rule
9               findExcepts(N, c_j)
10           else    findExcepts(N, c_i)
11           end
12        end
```

Figure 6: Finding sub-general rules and exceptions

The procedure traverses down the tree from a (sub-) general rule to find its exceptions. c_i is the class of the (sub-) general rule. In lines 1-3, if *Node* is a leaf and its class is different from the (sub-) general rule's class c_i, it is reported as an exception. Otherwise, the procedure goes down to its children to find exceptions (line 5). If the conditions in line 7 are satisfied, it means that node N can form a sub-general rule. n_i is the number of leaves of class c_i below the node N. The procedure marks this node to form a sub-general rule and recursively goes down (line 8 and 9). For example, in Figure 4(B), node 7 forms a sub-general rule because the conditions in line 7 are met (assume the rule is significant). If the conditions are not met, the procedure goes down further (line 10). For the tree in Figure 4(B), we finally obtain the GE tree in Figure 4(C), which has two high-level general rules, one sub-general rule and one exception rule.

Complexity: Since the algorithm traverses the decision tree at most twice, the complexity of the whole algorithm is $O(m)$, where m is the number of nodes in the decision tree.

Empirical Evaluation

To test the effectiveness and efficiency of the proposed technique, we applied it to the decision trees produced by C4.5 using 20 data sets in the UCI Machine Learning Repository (Merz and Murphy 1996). We also used the technique in a number of real-life applications.

Experiment results on the 20 data sets: Table 1 shows the experiment results. The first column gives the name of each data set. The second column gives the number of leaves in the decision tree produced by C4.5 (after pruning) for each data set. The third column gives the total number of leaves (or rules) in the GE tree (high-level general rules, sub-general rules and/or exception rules).

Table 1: Experiment results

	Data Sets	no. of decision tree leaves	no. of GE tree leaves
1	adult	518	132
2	anneal	53	16
3	austra	31	12
4	auto	49	22
5	breast	18	9
6	chess	28	11
7	cleve	30	16
8	crx	30	11
9	diabetes	22	12
10	german	103	38
11	glass	23	13
12	heart	18	10
13	iono	18	10
14	kdd	330	51
15	mushroom	26	12
16	pima	22	12
17	satimage	216	127
18	splice	157	63
19	tic-tac	95	39
20	waveform	317	146
	Average	105.2	38.1

Comparing column 2 and column 3, we can see that the number of leaves (or rules) in the GE tree is substantially smaller. On average, over the 20 data sets, the number of leaves (or rules) in the GE tree is only 36% of that of the original decision tree. This shows that the GE tree significantly simplifies the decision tree. The execution times are not reported as the algorithm runs so fast that they cannot be logged.

For these experiments, chi-square test at the confidence level of 95% is used to measure the significance of a rule.

Applications: The proposed technique has been used in a number of real-life applications. Due to space limitations, we can only briefly describe one here. In a disease application, the original tree produced by C4.5 from the data has 35 leaves, while the GE tree has only 10 leaves or rules. For instance, a general rule says that if the patient's age is less than 44, the chance of having the disease is very small with 3 exception regions. However, the corresponding sub-decision tree has 14 leaves. Another general rule says that if the patient's age is greater than 62, the chance of having the disease is very high with only one exception. However, C4.5 produces 5 fragmented rules. What is also important is that the users find that the decision tree is hard to understand because it is difficult to obtain a good overall picture of the domain from the fragmented decision tree leaves, which are hard to piece together. The GE tree, on the other hand, is more intuitive.

Related Work

The problem of too many rules has been studied by many researchers in data mining. However, to the best of our knowledge, there is no existing work that tackles the problem from a knowledge representation point of view.

In the *interestingness* research of data mining, a number of techniques (e.g., Piatesky-Shapiro & Matheus 1994; Klemetinen et al 1994; Silberschatz & Tuzhilin 1996; Liu & Hsu 1996; Padmanabhan & Tuzhilin 1998) have been proposed to help the user find interesting rules from a large number of discovered rules. The main approaches are: (1) using some interestingness measures to filter out those uninteresting rules; and (2) using the user's domain knowledge to help him/her identify unexpected rules. However, none of the methods touches the knowledge representation issue.

In (Liu, Hsu & Ma 1999), a method is proposed to summarize the discovered associations (Agrawal, Imielinski, & Swami 1993). The main idea is to find a set of essential rules to summarize the discovered rules. It does not study the knowledge representation issue. Furthermore, the technique cannot be used for decision trees because it requires all possible associations to be discovered. It also does not work with continuous attributes.

(Suzuki 1997; Liu et al 1999) study the mining of exception rules from data given some general rules. (Compton & Jansen 1988) proposes *ripple-down rules* [4], which are rules with exceptions, for knowledge acquisition. (Kivinen, Mannila & Ukkonen 1993) reports a theoretical study of learning ripple-down rules from data. Our work is different. We aim to simplify and summarize the data mining results by using the GE representation. We do not report another technique for mining general rules and/or exceptions from the data. Many existing data mining methods are already able to mine such rules and also the fragmented rules. The mining results are, however, not represented in an intuitive manner. Ripple-down rule mining and exception rule mining do not mine fragmented rules. We believe that such rules should also be discovered because they may be interesting to the user as well (see the Introduction section).

(Vilalta, Blix & Rendell 1997) studies the decision tree fragmentation problem in the context of machine learning. The problem there refers to producing trees that are too large and too deep. The research focus is on reducing the tree size by constructing compound features, by reducing the number of partitions, etc. Our work is different as we do not change the learning process, but only post-process the learning results to produce a more compact and easy-to-understand representation.

(Clark & Matwin 1993; Pazzani, Mani & Shankle 1997) study the problem of producing understandable rules by making use of existing domain knowledge in the learning process. They are different from our work as they are not concerned with different representations.

[4] Thanks to an anonymous reviewer for pointing us to the references of ripple-down rules.

Conclusion

In this paper, we have shown that the conventional if-then rules based representation of knowledge is inefficient for many data mining applications because the rules often fragment the knowledge that exists in data. This not only results in a large number of rules, but also makes the discovered knowledge hard to understand. We proposed a more efficient and easy-to-understand representation, which is in the form of general rules and exceptions. This representation is simple and intuitive, and also has a natural way of organizing the knowledge in a hierarchical fashion, which facilitates human analysis and understanding. Experiment results and real-life applications show that the proposed representation is very effective and efficient.

Acknowledgement: We would like to thank Shuik-Ming Lee, Hing-Yan Lee, Shanta C Emmanuel, Paul Goh, Jonathan Phang and King-Hee Ho, for providing us the data and giving us feedbacks. The project is funded by National Science and Technology Board, and National University of Singapore under RP3981678.

References

Agrawal, R., Imielinski, T., Swami, A. 1993. Mining association rules between sets of items in large databases. *SIGMOD-93*.

Clark, P. and Matwin, S. 1993. Using qualitative models to guide induction learning. *ICML-93.*

Compton, P. and Jansen, R. 1988. Knowledge in context: a strategy for expert system maintenance. *Proceedings of 2nd Australian Joint AI Conference* (*AI-88*).

Everitt, B. S. 1977. *The analysis of contingency tables*. Chapman and Hall.

Kivinen, J., Mannila, H. and Ukkonen. E. 1993. Learning rules with local exceptions. *EuroCOLT-93*.

Klemetinen, M., Mannila, H., Ronkainen, P., Toivonen, H. and Verkamo, A. 1994. Finding interesting rules from large sets of discovered association rules. *CIKM-94.*

Liu, B. and Hsu, W. 1996. Post-analysis of learned rules. *AAAI-96*.

Liu, B., Hsu, W and Ma, Y. 1999. "Pruning and Summarizing the discovered associations." *KDD-99.*

Liu, H., Lu, H., Feng, F and Hussain, F. 1999. Efficient search of reliable exceptions. *PAKDD-99*.

Merz, C. J. and Murphy, P. 1996. UCI repository of machine learning databases, [http://www.cs.uci.edu/~mlearn].

Padmanabhan, B. and Tuzhilin, A. 1998. A belief-driven method for discovering unexpected patterns. *KDD-98*.

Pazzani, M., Mani, S. and Shankle, W. 1997. Beyond concise and colorful: learning intelligible rules. *KDD-97*.

Piatesky-Shapiro, G. and Matheus, C. 1994. The interestingness of deviations. *KDD-94*.

Quinlan, R. 1992. *C4.5*: *program for machine learning.* Morgan Kaufmann.

Rivest, R. 1987. Learning Decision Lists. *Machine Learning*, 2.

Silberschatz, A., and Tuzhilin, A. 1996. What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Know. and Data Eng.* 8(6).

Suzuki, E. 1997. Autonomous discovery of reliable exception rules. *KDD-97.*

Vilalta, R., Blix, G. & Rendell, L. 1997. Global data analysis and fragmentation problem in decision tree induction. *ECML-97.*