Rule Discovery using Patterns from Joined Table of Relational Databases

Hyontai Sug and Douglas D. Dankel II
Department of Computer & Information Science and Engineering
University of Florida
Bldg. CSE, Room 301
Box 116120
Gainesville, FL 32611-6120
hyontai, ddd@cise.ufl.edu

Abstract

Several problems exist for data mining in real world databases: the difficulty of determining a decision attribute when limited domain knowledge exists, the difficulty in selecting a decision attribute from a new table formed from joining several relations, and the problem of elaborate data selection in the knowledge discovery process. This paper presents algorithms to solve these problems using methods that 1) determine a good decision attribute based on an approach developed from rough set theory and decision tree generation and 2) find meaningful frequent patterns based on attributes and dependencies in relational databases which are used to cluster values and generate tables. Moreover, our methods take advantage of the fact that more general concepts occur more frequently, making stepwise refinement possible.

1 Introduction

Prediction and description are two distinct and different goals for data mining. To achieve these goals involves the following principal data mining tasks [4]:

1. Classification maps a datum into one of a set of predefined classes.
2. Clustering is needed if the user does not define the classes. To cluster, systems must first discover subsets of related data items, then find appropriate description for each subset.

But we may have a question, *"Is it true that we surely know or absolutely don't know the class of each object?"* Most KDD (Knowledge Discovery in Databases) systems assume a good training data set has been selected during the data selection process, which usually leads to the assumption that a decision attribute is predefined, or can be selected, by the domain expert. But in real world databases we may have some difficulty in determining a decision attribute if we have limited domain knowledge. Moreover, if several relations are joined together, we may have a similar problem when we want to select a decision attribute from the new table. So, an objective assessing tool could greatly assist humans.

Because a raw database may contain a significant quantity of unimportant data that could be a hindrance to data mining, most KDD systems rely on a careful data selection process before the data is supplied to the data mining step, and the selection process relies heavily on human interaction since it is not easy to impose background and domain knowledge in the system that is as capable as a human's. As a result, *the data selection process is time consuming.* One approach to reduce this problem is to select only frequently occurring patterns from the raw data based on the philosophy that the infrequent cases will be so rare that they are worthless. The frequent cases are then organized with other related data to prepare for the data mining process.

Although Decision Trees [9] are the most successful data mining technique, they present a *comprehensibility problem if the generated tree is very large.* For example, SLIQ system by IBM can generate decision trees for very large databases [7]. If we take advantage of the fact that more general concepts occur more frequently, stepwise refinement of the generated decision trees is possible using our method.

We have devised algorithms that solve the problems of selecting a good decision attribute and selecting data elaborately when the target database is a relational database. Our method consists of two phases to:

- Find meaningful frequent patterns, grouping them together to generate tables by their relationships based on attributes and dependency.

- Determine a good decision attribute based on an approach developed from rough set theory and decision tree complexity.

In section 2, we first briefly provide the related works to our research, and in sections 3 and 4 we present our method in detail and illustrate the proposed method. Finally section 5 provides some conclusions.

2 Related Works

Many classification systems have been implemented including decision tree systems [9, 1], neural networks [6], rough set-based systems [11], etc. The basic assumption in all of these approaches is that we know the class of each example beforehand. But this assumption can be a limitation for the applicability of these systems since it is not always clear that we know exactly which attribute in a database table is the decision attribute. In other words, data mining systems based on classification are a kind of legacy of machine learning systems which rely on teachers to give good training examples. Due to this, an elaborate selection process is needed unless the database has been arranged in a very simple structure.

When we don't know exactly the class of each object or row of a database table, we should rely on clustering. Earlier work on this area has been mostly performed outside AI under the name of cluster analysis using numerical taxonomy and distance measures to define similarity between objects [2]. But the thresholds separating one cluster from another are prone to be arbitrary and there is limited use of semantic information about the objects. Conceptual clustering methods have been developed to take advantage of the object's semantics. CLUSTER/S [12] uses background knowledge to determine the class of an object. But there is no guarantee that the built–in background knowledge can be very helpful for near-optimal clustering, and evaluation using LEF(Lexical Evaluation Function) is prone to be arbitrary and difficult to justify. COBWEB [5] defines classes as a probability distribution over the values of attributes of objects and generates a hierarchy of classes. The system uses a category utility measure which is similar to Gini index [1]. COBWEB generates a taxonomy based on the probability distribution. But this taxonomy may differ from a human's since no external knowledge is used for the classification, so that it must rely on human's verification of the generated tree.

Another related family of KDD systems are the Association Rule Discovery systems [13] which find frequent patterns from transactional databases, like "80% of customers who buy instant coffee also buy non–dairy creamer." Because the Association Rule algorithm was invented to find rules from a transactional database which has the property of a simple unique structure containing attributes like product_ID, quantities_sold, there seems to be no need to consider the possibility of finding a common decision attribute that might be applicable to the some subset of the database. But, if we consider the case of finding regularities from a general database, we need to cover the possibility of finding the best decision attribute by grouping similar rules and simplifying the rules in these groups/clusters.

Among the related works Rough set theory [8], the Association Rule discovery method [13], and the method of decision tree generation [9], all of which form the basic ingredients of our research.

3 The Proposed Methods

3.1 Making a Table from Several Relations

Because our interest is in discovering rules for a relational database, we start by creating a table that comprises all of the relations of interest. To make a table, we need to perform natural join of the related relations, by focusing on a relation that has central meaning for the data mining. Note that our method of finding frequent patterns and converting them into rules only processes symbolic values, so we need to convert numeric values into nominal values.

3.2 Rule Discovery

3.2.1 Table Generation Algorithm for Joined Relation

The following algorithm creates a single table from the found frequent patterns.

1. Run an Association Rule Finding Algorithm with an appropriate support number (e.g., 3)[1] until {the number of item sets left} = 1.

2. Let the user select an iteration number of preference based on the found patterns.

3. Discard rare patterns: Find patterns that do not have the most common attributes including candidate decision attributes among the patterns in the selected iteration. If the frequency of the patterns is less than the given threshold (e.g. less than 5% of the found patterns), ignore these; otherwise, include the patterns.

4. Pick up rows from the joined table:

[1] Note that a support number is chosen based on the characteristics of the raw data. As the frequency of patterns become larger in the database, we may assign a larger support number to generate smaller size of frequent itemsets

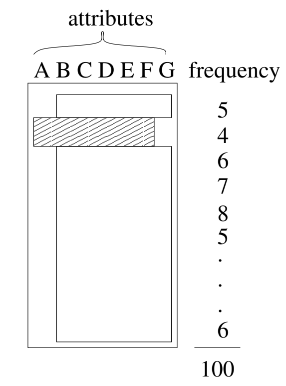

Figure 1. Discarding Rare Patterns

(a) Match all attributes in iteration 2 with the attributes of the chosen patterns at step 3.

(b) Select all patterns in iteration 2 which have values of the chosen attribute and candidate decision attribute.

(c) Select the rows which have the corresponding values with the patterns selected from step 4.(b).

5. Select a decision attribute from the table.(Details presented in the next section.)

6. Find the rules for the table.

7. Assign a certainty value to each rule found.

Note, if we adopt a higher iteration number (i.e., a value larger than 2) as a base for mapping to the original joined table, we would have a smaller table for rule generation.

Example: We discard rare patterns that have an attribute other than a most common attributes and which also occur rarely, e.g., relative frequency $< 5\%$. See the discarding rare patterns in Figure 1.

3.2.2 Selecting a Decision Attribute from the Table

Because we want to find a decision attribute from the table made from joining individual relations, it may be unclear which is a good decision attribute. But it may be possible to select a few attributes as candidates. Our basic idea is to select an attribute that has the largest positive region and the smallest boundary region based on rough set theory. But to prevent preferring many-valued attributes, we give some bias. In addition, to generate a simpler decision tree or rule set, we also consider the complexity of trees:

k_1 ×(computed value based on the size of positive region)
+ k_2 ×(computed value based on decision tree complexity)
where $k_1 + k_2 = 1$

Depending on the user preference, one may increment or decrement the k_1 and k_2 values. If k_1 is increased, categorization accuracy is emphasized and if k_2 is increased a simpler rule set is emphasized. We select the attribute having the largest value of the above equation as a decision attribute.

If the computation of the tree complexity-based value results in a large decision tree, the computation time will be significant. To limit this time we may specify a threshold value as the error rate of the trial decision tree. Then we can examine the error rate of the trial decision tree as the tree is being generated in a BFS(Breadth First Search) manner. The error rate is computed by applying Quinlan's PEP(Pessimistic Error Pruning) [3] method to take advantage of its top down pruning and simple computation while beholding its good performance. The error rate of the tree being generated is,

$$\frac{\sum_{t \in L}(e(t) + 0.5)}{\sum_{t \in L} n(t)}$$

where L is the set of leaf nodes, e(t) is the number of objects not belonging to the majority in leaf node t, and n(t) is the number of objects in leaf node t.

If the tree cannot be further generated to fulfil the given error rate due to the exhaustion of test attributes, the tree complexity is adjusted by,

(the number of generated nodes so far) ×
(1 + k_3 × (error rate))

where k_3 is the third parameter the user can choose which is usually set to 1.

The score based on the complexity of tree is adjusted to assign a larger score to the simpler tree. The simplest tree receives a score of 100 and other trees receive scores based on their relative complexity using the following equation:

100 / ((a tree size)/(the sipmlest tree size))

To compute the value based on the size of the positive region repeat the following steps for the each user-selected potential decision attribute.

Steps:

1. Select an attribute.

2. Sort the table using all attributes of the table other than this decision attribute.

3. Find the size of the positive region: count the number of rows having the same value on the condition attributes but having different values on the decision attribute. Note that this number is the number of rows belonging to the boundary region. By subtracting the value from the total number of rows, we obtain the number of rows belonging to the positive region.

Obj.#	a	b	d	e	f
1	0	0	1	1	1
2	0	1	2	2	2
3	0	2	2	2	3
4	1	0	3	1	4
5	1	1	1	1	5
7	1	1	2	2	5
6	1	2	2	1	6
8	1	2	1	1	7

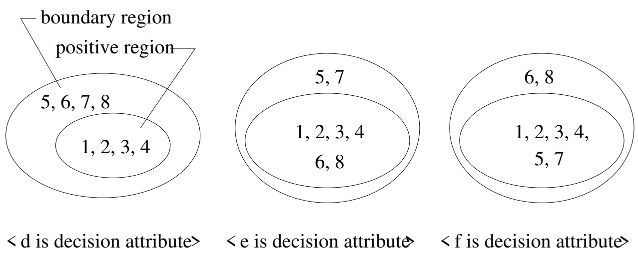

Figure 2. An Example Table sorted by Condition Attribute Values and Corresponding Diagrams

If an attribute has key-like characteristics, it will have a very small Boundary Region. Since these attributes should be avoided, we need some adjusting factor to score the attributes:

$$k_1 \times (\frac{|X|}{|T|})/(\frac{|D|}{|B|})$$

where

- X = the number of rows belonging to positive region,
- T = the total number of rows of the table,
- D = the number of values (classes) possessed by the decision attribute currently being considered, and
- B = the number of values of the decision attribute which has the largest number of values among all potential decision attributes.

Example: Consider the table in Figure 2 where the condition attributes are a and b, and the candidate decision attributes are d, e, and f.

Although decision = 1 is made by objects 1, 5, and 8 when the decision attribute is d, objects 5 and 8 cannot belong to the positive region since object 5 conflicts with object 7 which has decision = 2, and object 8 conflicts with object 6 which has decision = 2.

So, $\underline{C}D_1$ = $\{1\}$. Similarly, $\underline{C}D_2$ = $\{2, 3\}$, $\underline{C}D_3$ = $\{4\}$, and, $\underline{C}E_1$ = $\{1, 4, 6, 8\}$, $\underline{C}E_2$ = $\{2, 3\}$, and, $\underline{C}F_1$ = $\{1\}$, $\underline{C}F_2$ = $\{2\}$, $\underline{C}F_3$ = $\{3\}$, $\underline{C}F_4$ = $\{4\}$, $\underline{C}F_5$ = $\{5, 7\}$, $\underline{C}F_6$ = $\underline{C}F_7$ = $\emptyset$.

Supplier

S#	Sname	Location
s_1	...	l_1
s_2	...	l_1
s_3	...	l_2
s_4	...	l_2

Part

P#	Quality	Price
p_1	B	H
p_2	A	H
p_3	A	H
p_4	B	M
p_5	A	M
p_6	B	M
p_7	A	M
p_8	A	H

Supplied_Part

S#	P#	Quantity
s_1	p_1	...
s_1	p_8	...
s_2	p_2	...
s_2	p_3	...
s_3	p_4	...
s_3	p_5	...
s_4	p_7	...
s_2	p_6	...

Figure 3. The Example Relations

See the example table sorted by condition attribute values and corresponding diagrams in Figure 2. Thus,

When the decision attribute is D, $\frac{4}{8}/\frac{3}{7}$ = 1.1666

When the decision attribute is E, $\frac{6}{8}/\frac{2}{7}$ = 2.6251

When the decision attribute is F, $\frac{6}{8}/\frac{7}{7}$ = 0.75

Since E has the largest value, if we select it as the decision attribute, we will have less conflict. Note that attribute F has key-like characteristics (i.e., it has smallest computed value), so it may not be selected as the decision attribute.

The score based on the positive region is adjusted to give the ideal one 100. The score is assigned by the linear interpolation of three points x_0, x_1, and x_2 where

x_2 = 1 / ($\frac{|C|}{|B|}$)
x_1 = 0.5 / ($\frac{(|B|+|C|)/2}{|B|}$)
x_0 = 0

where B is defined as above and C is the number of values of the decision attribute which has the smallest number of values among all potential decison attributes. The resulting score for x_2, x_1, and x_0 are 100, 50, and 0 respectively. So, the score y for some x is,

$y = 50\{x/(x_2 - x_1) + 1 - x_1/(x_2 - x_1)\}$ when $x_1 \leq x \leq x_2$
$y = 50(x/x_1)$ when $0 \leq x \leq x_1$

4 An Example

To illustrate the table generation algorithm for joined relations suppose we have the three relations Supplier, Part, and Supplied_part as shown in Figure 3 and the relation Supplied_part has the central meaning for our data mining. We want to know which one, the Quality or Price of part, is a better decision attribute. Joining the three relations and projecting on Location, P#, Quality, and Price, we obtain the joined and projected table in Figure 4.

Step 1: Apply the association rule finding algorithm to find frequent patterns.

The frequent patterns when the support number is 3 are:

Location	P#	Quality	Price
l_1	p_1	B	H
l_1	p_8	A	H
l_1	p_2	A	H
l_1	p_3	A	H
l_2	p_4	B	M
l_2	p_5	A	M
l_2	p_7	A	M
l_1	p_6	B	M

Figure 4. The Joined and Projected Table

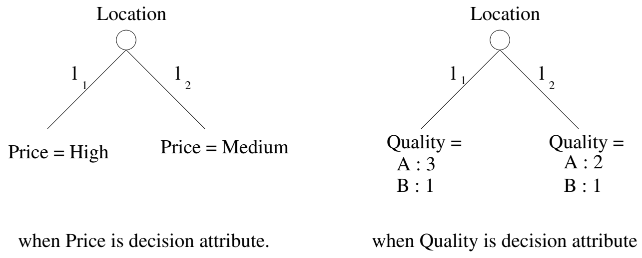

Figure 5. Trial Decision Trees

iteration 2

#	*freqnt-itmset*			*fy.*
1		A	H	3
2	l1		H	4
3	l2		M	3
4	l1	A		3

iteration 3

#	*freqnt-itmset*			*fy.*
1	l1	A	H	3

Step 2: Select iteration 3 as interesting.
Step 3: Pattern l_1AH is the only pattern, so select it.
Step 4: Associate all frequent itemset in iteration 2 with the frequent itemset in iteration 3.

All frequent itemsets in iteration 2 are selected. Now, associate the original row in the joined relation with itemsets 1, 2, 3, and 4. The result is a table having frequent patterns and related patterns only. Note that the last row in the joined and projected table in Figure 4 has been omitted by the algorithm since it belongs to a very rare case.

l_1BH (support number = 1)
l_1AH (support number = 3)
l_2BM (support number = 1)
l_2AM (support number = 2)

Step 5: Select a decision attribute from the table.

We used the entropy measure of Quinlan's C4.5 [9] for trial decision tree and remember that we have chosen Quality and Price as candidate decision attributes. In each case below, we have listed all rows from the table to aid us in our computation.

Step 5.1: Sort the table and count non–conflicting rows with respect to decision attribute values.

1) When Price is a decision atttribute:

l_1A H l_2B M
l_1A H l_2A M
l_1A H l_2A M
l_1B H

Price has a 100% positive region because there is no conflict in the decision.

2) When Quality is a decision atttribute:

l_1H A l_2M B
l_1H A l_2M A
l_1H A l_2M A
l_1H B

We have nothing in the positive region.

Step 5.2: Compute score based on tree complexity.

1) When Price is a decision atttribute:
For Location, gain_ratio = 0.9856/0.9856 = 1. For Quality, gain_ratio = 0.0068/0.8632 = 0.079. Therefore, Location is chosen as the root of the decision tree resulting in the complexity for the decision tree of $3 \times 1.1429 = 3.4287$.

2) When Quality is a decision atttribute:
For Location, gain_ratio = 0.012/0.9856 = 0.0122. For Price, gain_ratio = 0.012/0.9856 = 0.0122. Since both Location and Quality have identical gain_ratios, Location is chosen randomly as the root of the decision tree. Since only 5 of the rows are in the majority class, we increase the tree size by 3/7 as shown in the trial decision tree in Figure 5, resulting in 4.3.

So, the score is 100 when Price is a decision attribute, whereas 100/(4.3/3.4287) = 80 when Quality is a decision attribute.

Step 5.3: Combine the computed value based on the size of positive region with the computed value based on the decision tree complexity.

We assume $k_1 = k_2 = 0.5$,

1) when Price is the decision attribute:
Since all six rows are in the positive region, numerator = 7/7. Since the number of values of both attribute Quality and Price is 2, denominator = 2/2. So the computed value based on the size of positive region = 100. Combining the result of computed score based on tree complexity, which is 100, we have $0.5 \times 100 + 0.5 \times 100 = 100$.

2) when Quality is the decision attribute:
Similarly, numerator = 0/7 and denominator = 2/2. So the computed value based on the size of positive region = 0. Combining with the result of computed score based on tree complexity, 80, we have $0 + 0.5 \times 80 = 40$.

As a result, we choose Price as the decision attribute and generate rules.

Step 6: Find rules for the table.

By going through the above steps, we now have a complete table and a decision attribute for rule generation. Rules can be generated using a conventional classification system like the decision tree-based system, C4.5 [9], or a rough set-based system [11].

Step 7: Compute the confidence of each rules found.

$$l_1 \rightarrow \text{H with confidence of } 100\%$$
$$l_2 \rightarrow \text{M with confidence of } 100\%$$

Note that the confidence does not count the rare case.

5 Conclusions

Although it is difficult to say that there are only two situations that we surely know (classification) or absolutely don't know the class of each object (clustering), yet little research on the situation between two cases has been done. As a method to deal with the situation between clustering and classification, we proposed our method which is based on three well-known techniques; rough set theory, decision tree generation method, and association rule finding algorithms. The generalization methods developed for the patterns found by association rule finding algorithms are ad hoc in nature, but our new method is more systematic since we consider possible relationships between found patterns by making trial decision trees and correctness of the rule set which will be found. Another good point of our method is that by giving the system different support numbers, stepwise refinement is possible. So, users can grasp the global view of the target database by using bigger support number, then he/she can focus on more specific parts of the generated decision tree by making a table which satisfies the specific condition. The bottleneck of the algorithm is the database access and our algorithm requires database access not more than three times [13]. Some research on the parallelization of decision tree generation and association rule finding algorithms has been done, so faster turnaround time will be also applicable to our method [10, 14].

References

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, Inc., 1984.

[2] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1993.

[3] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997.

[4] U. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/The MIT Press, 1996.

[5] D. Fisher. Iterative optimization and simplification of hierarchical clustering. *Journal of Artificial Intelligence Research*, 4:147–179, 1996.

[6] L. Fu. *Neural Networks in Computer Intelligence*. McGraw Hill, Inc., New York, 1994.

[7] M. Mehta, R. Agrawal, and J. Rissanen. Sliq: A fast scalable classifier for data mining. In *Proceedings of EBDT France*, March 1996.

[8] Z. Pawlak. *Rough Sets: theoretical aspects of reasoning about data*. Klumer Academic Publisher, 1991.

[9] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., 1993.

[10] A. Savasere, E. Omiecinski, and S. Navathe. *An Efficient Algorithm for Mining Association Rules in Large Databases*. College of Computing, Georgia Institute of Technology, Technical Report No.: GIT–CC–95–04.

[11] R. Slowinski, editor. *Intelligent Decision Support: Handbook of Applications and Advances of the Rough Set Theory*. Kluwer Academic Publishers, 1992.

[12] Stepp and Michalski. *Conceptual Clustering: Inventing Goal–oriented Classification of Structured Objects*, pages 471–498. Morgan Kaufmann Publishers, Inc., 1986.

[13] H. Toivonen. *Discovery of Frequent Patterns in Large Data Collections*. phD thesis, Department of Computer Science, University of Helsinki, Finland, 1996.

[14] J. Venegas. *On Decision Tree Induction for Knowledge Discovery in Very Large Databases*. phD thesis, University of Florida, 1996.