SIGKDD-2000, Aug 20-23 2000, Boston, USA

Multi-Level Organization and Summarization of the Discovered Rules

Bing Liu, Minqing Hu, and Wynne Hsu
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
{liub, huminqin, whsu}@comp.nus.edu.sg

ABSTRACT
Many existing data mining techniques often produce a large number of rules, which make it very difficult for manual inspection of the rules to identify those interesting ones. This problem represents a major gap between the results of data mining and the understanding and use of the mining results. In this paper, we argue that the key problem is not with the large number of rules because if there are indeed many rules that exist in data, they should be discovered. The main problem is with our inability to organize, summarize and present the rules in such a way that they can be easily analyzed by the user. In this paper, we propose a technique to intuitively organize and summarize the discovered rules. With this organization, the discovered rules can be presented to the user in the way as we think and talk about knowledge in our daily lives. This organization also allows the user to view the discovered rules at different levels of details, and to focus his/her attention on those interesting aspects. This paper presents this technique and uses it to organize, summarize and present the knowledge embedded in a decision tree, and a set of association rules. Experiment results and practical applications show that the technique is both intuitive and effective.

1. INTRODUCTION
Producing too many rules has been regarded as a major problem with many data mining techniques [12, 14, 23, 21, 5]. The large number of rules makes it very hard for manual inspection of the rules to gain insight of the domain. This paper argues that the key problem is not with too many rules, but with our inability to organize and to present the rules in such a way that is easily analyzed by the user.

Let us use an analogy to explain. We equate the discovered rules from a database with the knowledge contained in a conventional scientific book. This is reasonable because discovered rules are a form of knowledge. Although a book can contain a huge amount of knowledge in a few hundred pages, people seldom complain that "this book is too thick and contains too much information, and I cannot read and use it". In most cases, a thick book with a comprehensive coverage is of great value. However, we do often hear people complain that a book or an article is badly written and badly organized. A good book is easy to read because the author has put a great deal of effort to organize and present the contents of the book in such a way that is able to constantly draw the reader's attention and enables him/her to follow and to understand the contents easily.

A common strategy for organizing the contents of a book is by hierarchy. In such an organization, the contents are divided into topics, and presented at different levels of details. The main advantage of the hierarchical organization is that it allows the reader to manage the complexity of knowledge, and to view it from general to specific, and to focus his/her attention on those interesting aspects. For a hierarchical organization to work effectively, summarization is crucial. In a book, chapter and section headings, and abstract and/or summaries are used to tell the reader what to expect from each chapter or each section. From the summaries, the reader can obtain an overall view of what the chapter or section is about and decide whether to read the details.

The question is "Can we organize and present the discovered rules like a well organized book so that they can be easily browsed and used by human users?" This paper shows this is possible. Till now, little research has been done in this direction. In this paper, we propose a novel technique, called *general rules, summaries & exceptions* (GSE), to organize, summarize and present the discovered rules.

In the past few years, we had a number of applications in education, medical and other domains. In these domains, users always find it very hard to browse through the discovered rules (even when the set is not very large) to gain a good understanding of the domain. Although there exist techniques to deal with the problem of too many rules, they are still not satisfactory. Through extensive interactions with our users, we discovered that our users always talk about knowledge in a certain way, i.e., in terms of *general patterns* and *special cases*. For example, when we tried to profile the students of an educational institution, our users would make the following comment: In general, our good students have certain profiles. However, in some situations, these may not be true. In the medical applications, the situation is similar. The doctors always say that people with certain characteristics tend to have a particular disease. However, in some special situations, they may not develop the disease. We began to realize that the individual rule based representation of knowledge employed in current data mining techniques is not intuitive for human users. This inspired us to design the proposed technique, which use general rules to give the user a big picture of the domain and exceptions to point out the special cases. The summaries are used to summarize those non-essential rules, which will be clear later.

Existing data mining systems typically generate a list of rules with no further organization. This is like writing a book or a paper by randomly listing out all the facts and formulas. The resulting book will be difficult for anyone to read because one cannot see any relationships among the facts and formulas. In data mining applications, a list of rules presents the following two problems:

1. It obscures the essential relationships and the special cases or exceptions, and makes it hard for the user to piece the information in the rules together to obtain a good overall understanding of the underlying relationships in the domain.
2. It represents the discovered knowledge 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 manage the complexity of knowledge, to view the knowledge at different levels of details, and to focus our attention on the interesting aspects.

Let us use an example to illustrate. Our example dataset has two attributes (A1 and A2), 500 data tuples and two classes, • and **O**. The domain of A1 is {a, b, c, d}, and the domain of A2 is {x, y}. Assume a rule miner found the following rules (*support* and *confidence* have their usual meanings [2]):

R1:	A2 = x → •	(sup = 41%, conf = 77%)
R2:	A1 = a, A2 = x → •	(sup = 14%, conf = 97%)
R3:	A1 = c, A2 = x → •	(sup = 13%, conf = 91%)
R4:	A1 = d, A2 = x → •	(sup = 13%, conf = 93%)
R5:	A1 = b, A2 = y → •	(sup = 6%, conf = 79%)
R6:	A2 = y → **O**	(sup = 38%, conf = 83%)
R7:	A1 = b, A2 = x → **O**	(sup = 10%, conf = 83%)
R8:	A1 = a, A2 = y → **O**	(sup = 13%, conf = 98%)
R9:	A1 = c, A2 = y → **O**	(sup = 12%, conf = 97%)
R10:	A1 = d, A2 = y → **O**	(sup = 12%, conf = 92%)

By simply looking at the 10 rules above, it is hard for us to obtain a good overall picture of the relationships in the data.

These rules are actually generated from the data space given in Figure 1. The two numbers within the brackets next to each junction (or value combination) are the number of • class tuples, and the number of **O** class tuples respectively. Only the majority class symbol is drawn for each value combination.

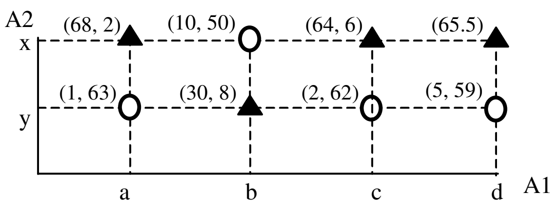


Figure 1. The example data space

Now, let us change the 10 rules to the following two *general rule, summary & exceptions* (GSE) patterns, the picture becomes clear and much easier to understand. Let us ignore the summary component for the time being.

GSE-1: A2 = x → • (sup = 41%, conf = 77%)
Except R7: A1=b, A2=x → **O** (sup = 10%, conf = 83%)

GSE-2: A2 = y → **O** (sup = 38%, conf = 83%)
Except R5: A1=b, A2=y → • (sup = 6%, conf = 79%)

The first part of a GSE pattern is a general rule, and the second part (after "Except") is a set of exceptions to the general rule. From these two parts of the GSE patterns, we see the essential relationships of the domain, the two general rules, and the special cases, the exceptions (R5 and R7). This representation is much simpler than the 10 rules because R2-R4 and R8-R10 are not used. These rules (R2-R4 and R8-R10) fragment the knowledge in the data. They are non-essential, and worse still they make the discovered knowledge hard to interpret.

This, however, is not to say that the fragmented rules are completely useless. The fragmented rules may be interesting due to their higher confidences (e.g., 97% and 98%). We use the summary component to summarize these fragmented rules. Here, we use the highest confidence of the rules as the summary (see Section 3). Thus, the complete GSE-1 and GSE-2 are as follows:

GSE-1: A2 = x → • (sup = 41%, conf = 77%)
Summary Highest conf. of • class rules = 97%
Except R7: A1=b, A2=x → **O** (sup = 10%, conf = 83%)

GSE-2: A2 = y → **O** (sup = 38%, conf = 83%)
Summary Highest conf. of **O** class rules = 98%
Except R5: A1=b, A2=y → • (sup = 6%, conf = 79%)

The GSE representation of the discovered knowledge has two key advantages:

Simplification: It simplifies the discovered rules using general rules and their exceptions, and hides those fragmented rules that have the same classes as the general rules. The fragmented rules are non-essential. They are summarized with the summary field of the GSE pattern. If one is interested in them, one can follow this link to view them.

Organization: It organizes the discovered knowledge in an intuitive way, which is easy for the user to follow because the GSE representation is a common way of representing knowledge in our daily lives. What is also important is that this organization naturally draws the user's attention. In our applications, we found that exceptions often attract the user because they are unexpected with respect to the general rules. When the user sees some general rules, he/she is almost always eager to see the exceptions.

In Section 3, we will see that the GSE representation also naturally introduces a hierarchical (multi-level) organization of the discovered rules.

This paper presents the GSE representation, and uses it to organize, summarize and present the knowledge embedded in a decision tree and a set of association rules. Since decision trees and association rules embed somewhat different types of knowledge, the proposed technique is applied differently depending on the context. Experiment results and practical applications show that the proposed technique is very effective and efficient.

2. RELATED WORK

In the past few years, a number of techniques were proposed to deal with the problem of too many rules [21, 23, 12, 14, 9, 19, 1]. The main idea of these techniques is to use the user's knowledge or statistical measures to remove those uninteresting rules. Our work is different. We aim to organize and summarize the rules so that the user can browse through them easily and effectively.

[9] proposes a template-based approach for finding interesting rules. This approach first asks the user to specify what rules he/she wants. The system then finds those matching rules. The technique assumes that the user knows exactly what he/she is looking for. However, in many situations, the user does not know

what to find exactly. His/her needs can only be met by incorporating browsing as well as template-based search in the system. It is well known that humans are better able to recognize something than to generate a description of it [11].

[23, 12, 14, 19] propose an expectation or belief based approach to find interesting rules. In this approach, the user is also required to input his/her existing knowledge about the domain, and the system then finds those conforming and unexpected rules. This approach is more advanced than the template-based approach because it not only finds those rules that satisfy the templates but also the unexpected rules. However, since it requires input knowledge from the user, it suffers from the problem of knowledge acquisition. That is, one may know a great deal, but it is very difficult for one to tell what he/she knows [14].

[24, 18] investigate using item constraints specified by the user in association rule mining to generate only the relevant rules. This approach also requires the user to know precisely what he/she wants or does not want.

[14] presents a technique to summarize the discovered association rules. It selects a subset of the rules, called direction setting rules (DS rules), to summarize the discovered rules. DS rules represent the essential relationships of the domain. For example, we have the following discovered rules from a loan application domain:

R1: Job = yes → Loan = Approved (sup = 40%, conf = 70%)
R2: Own_house = yes → Loan = Approved
(sup = 30%, conf = 75%)

If these two rules are known, then the following association rule is not so surprising to us:

R3: Job = yes, Own_house = yes → Loan = Approved
(sup = 20%, conf = 90%)

because it intuitively follows R1 and R2. R1 and R2 (the DS rules) are used to provide a summary of the three rules. R3 is non-essential. This technique is proven useful in our real-life applications. However, some shortcomings are also exposed. Firstly, it takes at least two rules to summarize another rule. However, the user only sees one rule at a time. Thus, when the user just sees a particular DS rule, it is not easy to imagine those rules that have been summarized because he/she may not have seen the other DS rules. Secondly, comparing to GSE representation, DS rules are less intuitive as DS rules do not directly reflect the way that we think about knowledge. Finally, the technique cannot be applied to summarize the rules in a decision tree because it requires at least two rules to summarize another. However, one of the rules may not be in the tree as a decision tree typical only discovers a subset of the rules in data.

In [15], we proposed to use general rules & exceptions (GE), without the summary component, to represent decision trees. The GE representation is less effective than GSE as the information on the fragmented rules is lost. [15] also does not study organization, summarization and presentation of association rules, which presents a different set of problems.

[26, 16] study the mining of exception rules from the data given some general rules. Our work is different. We aim to organize and summarize the mining results. We do not report another technique to mine exceptions from the data.

[4] proposes *ripple-down rules,* which are rules with exceptions, for knowledge acquisition in AI. This confirms that general rules and exceptions are intuitive to human experts.

3. GENERAL RULES, SUMMARIES AND EXCEPTIONS

We now present the GSE representation. In this paper, we are interested in rules generated from a relational table D, which consists of a set N of data tuples described by m distinctive attributes A. Rule mining in such a dataset is typically targeted at a specific attribute because the user normally wants to know how the other attributes are related to this *target* attribute (which can have many values) [13, 3]. This target attribute is often called the class attribute, and its values, the classes (denoted by C). Below, we define the GSE pattern in this context.

Definition 1 (GSE patterns): A GSE pattern consists of three components, a single general rule (an if-then rule), a summary and a set of exceptions. It is of the following form:

$X \rightarrow c_i$ $(sup, conf)$
Summary
Except $E_1, \ldots, E_n$

where:

1. $X \rightarrow c_i$ is the *general rule*. X is a set of *conditions*, and $c_i \in C$ is a class. *Sup* and *conf* are support and confidence of the rule with their usual meanings [2].
2. The 'Summary' field gives some essential information of the rules *below* the general rule. A rule r is below another rule R if r only *covers* a subset of the data tuples that are covered by R. A rule covers a data tuple if the data tuple satisfies the rule's conditions.
3. $E = \{E_1, \ldots, E_n\}$ is the set of *exceptions*. E may be empty ($E = \varnothing$). Each E_j is a GSE pattern of the form:
   $X_j \rightarrow c_k$ $(sup, conf)$
   Summary
   Except $E_{j1}, \ldots, E_{jw}$
   where $(X_j \rightarrow c_k)$ is called an *exception rule*. X_j is a set of conditions, and $c_k \in C$ and $c_k \neq c_i$. Data tuples that satisfy X_j must satisfy X. E_{jl} are the exceptions of $(X_j \rightarrow c_k)$. Note that $(X_j \rightarrow c_k)$ now behaves as the general rule of E_{jl}.

We now discuss the intuitive meanings of the three components of the GSE pattern:

1. The general rule gives the user a general relationship or a big picture of the domain.
2. 'Summary' highlights some key information of the rules covered by the general rule. Based on the summary, the user can make choices to explore only those interesting subset of the rules. The 'Summary' field is not defined formally. It varies according to implementation needs.
3. Exception rules are unexpected rules with respect to the general rule. They allow the user to see those special cases that do not follow the general rule. An exception rule covers only a subset of the data tuples covered by its general rule, and the class of the exception rule is different from that of its general rule.

Since each exception is also a GSE pattern, the GSE pattern can naturally represent knowledge in a hierarchical fashion. That is, a general rule can have exceptions, and an exception can also have its own exceptions, and so on. However, the *depth* (or the number of *levels*) of the hierarchy should not be too large, otherwise it also becomes difficult to understand. In an actual implementation, the user can decide when the system should stop using GSE patterns. This will be explained in Section 4, which is also applicable to Section 5.

We will not give a definition of a general rule (or an exception rule) here because for different data mining tasks it may be defined differently (see Section 4 and 5).

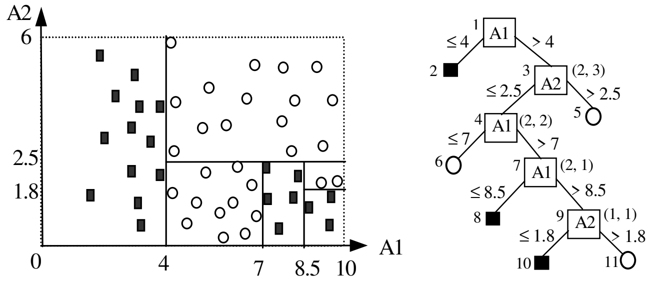
4. REPRESENTING DECISION TREES USING GSE PATTERNS

Decision tree construction [22] is a popular method for building classification models. In this section, we use GSE patterns to organize and summarize the knowledge embedded in a decision tree. The GSE representation can also be expressed as a tree, which we call a *GSE tree*.

Our main objective here is to simplify a decision tree by converting it to a GSE tree. The resulting GSE tree can still be used for classification to produce the same result as the original decision tree.

A decision tree has two types of nodes, *decision nodes* (*internal node*) and *leaf nodes*. A *decision node* specifies some test to be carried out on an attribute value. A *leaf node* indicates a class. A decision tree basically 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 regions in Figure 2(A) are produced by the tree in Figure 2(B) (ignore the numbers within () for the time being). The tree essentially represents a set of rules. For example, the leaf node 6 in Figure 2(B) can be expressed with the rule,

$$A1 > 4, A1 \le 7, A2 \le 2.5 \rightarrow \mathbf{O}$$



(A) (B)

Figure 2. A partitioning of the data space and its decision tree

We now discuss how to convert a decision tree to a GSE tree. Before presenting the detailed algorithm, we first define a general rule or an exception rule in the context of a decision tree.

Definition 2 (general or exception rules): A general (or an exception) rule is a *significant rule*, and its class is the *majority class* of the data tuples covered by the rule.

The significance of a rule can be measured in various ways. In this work, we measure it using chi-square (χ^2) test/fisher's exact test [1] [6]. The way that we measure the significance of a rule is the same as that in [14]. We will not discuss it further in this paper.

[1] Chi-square test can be computed very efficiently. However, it is based on the assumption that the expected frequencies should not be small [6]. One popular rule of thumb for the 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.

We require that the class of a general rule be the *majority class* because a classification system always outputs the majority class in prediction. In the GSE representation, we aim to keep the original spirit of the underlying data mining task.

We now describe the conceptual algorithm for converting a decision tree to a GSE tree. The algorithm consists of two steps (summarization is done in both steps, see the algorithm later):

1. Find top-level general rules: We descend down the decision tree from the root node to find the nearest nodes whose majority classes can form significant rules. We call these rules the *top-level general rules*.
2. Find exceptions: We go down from each top-level general rule to recursively find exceptions, exceptions of the exceptions and so on. We determine whether a tree node should form an exception rule or not using the following criteria:
   - *Significance*: An exception rule must be significant, and its class is the majority class and is different from that of its general rule (following **Definition** 1 and 2).
   - *Simplicity*: If we use a tree node to form an exception rule, it should result in fewer rules in the final GSE representation (GSE tree). The complexity of a GSE representation of a decision tree is measured by the sum of the number of top-level general rules and the number of exception rules, which is also the number of leaves in the GSE tree. Let the class of the rule R formed by the current node be c_j, and the class of its general rule 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 an exception rule if the following condition is satisfied:

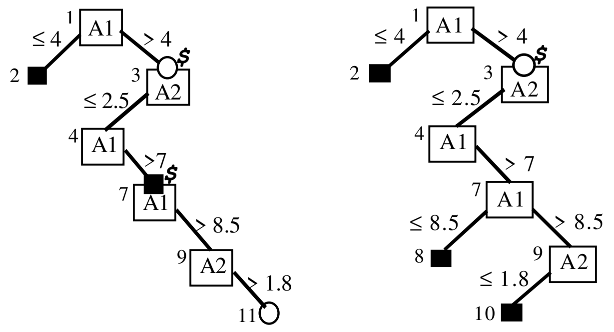
   $$n_j \ge n_i + 1 \qquad (4.1)$$

   If this inequality holds, it means that using R as an exception rule with class c_j (1 in the formula represents this exception rule) cannot result in a more complex final description of the knowledge. The number of leaves l of the other classes is not included in (1) because l should appear on both sides of (1) and get canceled. n_i+1+l (or n_j+l) is the total number of exception rules if we use (or do not use) R as an exception rule at this stage.

We use the example in Figure 2 to illustrate the idea. From Figure 2(B), the following GSE patterns can be formed (for simplicity, we omit the support and confidence information):

GSE-1: A1 ≤ 4 → ■
GSE-2: A1 > 4 → **O**
Summary Highest conf. of **O** = 100%
Except A2 > 7, A2 ≤ 2.5 → ■
Summary Highest conf. of ■ = 100%
Except A1 > 8.5, A2>1.8, A2 ≤ 2.5 → **O**

A1 ≤ 4 → ■ and A1 > 4 → **O** are top-level general rules (which are formed by node 2 & 3 in Figure 2(B)). An exception rule (with its exception) is formed at node 7 (according to formula (4.1), we have 2 ≥ 1 + 1). Intuitively, we can also see in Figure 2(A) that the area within (A1 > 7, A2 ≤ 2.5) is a general area for class ■. This gives us a 3-level hierarchical representation of knowledge. Figure 3(A) shows the corresponding GSE tree, which has the complexity of 4 as it has 4 leaves. An internal node with a class (■ or **O**) represents a (general or exception) rule. ***$*** represents the summary at the node. For simplicity, we do not list the content of the summary in the GSE tree. Those fragmented tree leaves or rules are omitted. In our implementation, these leaves or rules can be accessed by clicking on the summary.



(A). GSE-1 and GSE-2 (B). GSE-1 and GSE-2'

Figure 3. Two alternative GSE tree representations

As mentioned in Section 3, when the depth of a hierarchy is too deep, the knowledge embedded in it becomes harder to understand. An alternative representation of GSE-2 is GSE-2', which stops using the GSE representation after node 3.

GSE-2': A1 > 4 → **O**
Summary Highest conf. of **O** = 100%
Except A1 > 7, A1 ≤ 8.5, A2 ≤ 2.5 → ■
A1 > 8.5, A2 ≤ 1.8 → ■

The hierarchy of GSE-2' has only two levels. This may be easier to understand. The user can obtain the desired output by specifying the maximum depth of the hierarchy. After this depth, no further split of general rules and exceptions will be performed. Figure 3(B) gives the GSE tree for GSE-1 and GSE-2'.

The detailed algorithm, called findGSE, is given in Figure 4. Notice that finding general rules and finding exceptions rules are essentially the same because exception rules are general rules for their own exceptions. Thus, the two intuitive steps can be performed with a single procedure, i.e., the findExcepts procedure in Figure 4.

findGSE calls two procedures, countLeavesAndSummarize, and findExcepts. As discussed above, to decide whether a node below a general rule should form an exception rule or not, we need the number of leaves of each class below the node (formula (4.1)). The countLeavesAndSummarize procedure performs this task and also summarizes the tree leaves (or rules) below each internal node, which in our implementation records the highest confidence of the leaves of each class below the node. This procedure is fairly straightforward and is not discussed here. In the case of Figure 2(B), countLeavesAndSummarize produces the numbers of leaves of different classes below each internal node as shown within "()" in Figure 2(B). The highest confidence of the rules of each class below each node is not shown. The first number within () is the number of ■ class leaves below the node, and the second number is the number of **O** class leaves.

The findExcepts procedure traverses down the tree from a general rule to find its exceptions. c_i is the class of the general rule. Note that when starting from the root node, the second input parameter is set to 0 (which is a special value, not a class in the database) to indicate that there is no general rule at the root node. Thus, the first exception found along each path of the tree is a top-level general rule. In lines 1-3, if *Node* is a leaf and its class is different from the 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 *child* can form an exception rule. n_i is the number of leaves of class c_i below the node *child*. The procedure marks this node to form an exception rule and recursively goes down (line 8 and 9). For example, in Figure 2(B), node 7 forms an exception 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 2(B), we obtain the GSE tree in Figure 3(A), with two top-level general rules and two exception rules.

A GSE tree can be constructed very efficiently because the algorithm traverses the decision tree at most twice, once by countLeavesAndSummarize, and once by findExcepts.

Algorithm findGSE(*RootNode*)
1 countLeavesAndSummarize(*RootNode*);
2 findExcepts(*RootNode*, 0)

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 in *Node.children* **do**
6 c_j = majority class of node *child*;
7 **if** $c_j \neq c_i$ AND the rule formed from root node to *child* with the class c_j is significant AND $n_j \geq n_i + 1$ **then** /* $n_0 = 0$ for root node */
8 mark it as an exception rule;
9 findExcepts(*child*, c_j)
10 **else** findExcepts(*child*, c_i)
12 **end**

Figure 4: Finding general rules, summaries and exceptions

5. ORGANIZING ASSOCIATION RULES USING GSE PATTERNS

We now discuss how to represent association rules using GSE patterns. Since association rule mining is not aimed at producing a classification model but to find all significant rules, we need to apply the proposed technique differently. These will become clear later. The nature of association rule mining also presents some special problems that make it hard to find *real* exceptions because of the minimum support and rule overlapping (the same data tuple may be covered by many rules). In this section, we first present the basic framework of using GSE patterns to organize association rules. We then present and deal with the special problems.

5.1 Association Rule Mining

The association rule mining model is stated as follows [2]: Let $I = \{i_1, \ldots, i_r\}$ be a set of *items*, and D be a set of data cases (transactions). Each data case consists of a subset of items in I. An *association rule* is an implication of the form $X \rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. The rule $X \rightarrow Y$ holds in D with confidence c if c% of data cases in D that support X also support Y. The rule has support s in D if s% of the data cases in D contains $X \cup Y$. The problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support and minimum confidence.

In this work, we focus on association rule mining from a relational table, which is described by a number of attributes. An item is an attribute value pair, i.e., (attribute = value) (numeric attributes are discretized). We also have a target or class attribute, which can have a number of values C (or classes). With the class attribute, we only mine rules of the form:

$$X \rightarrow c_i$$

where $c_i \in C$ is a class, and X is a set of items from the rest of the attributes. We say a rule is *large* if it meets the minimum support. We will not discuss the algorithm for mining such rules as the existing algorithm in [2] can be easily modified for this purpose (see also [13]).

Rule pruning: It is well known that many discovered associations are redundant or minor variations of others. Their existence may simply be due to chance rather than true correlation. Those insignificant rules should be removed. [14] studies the pruning of association rules. Its basic idea can be shown with the following two rules from a loan application data.

R: Job = yes → Loan = Approved (sup = 60%, conf = 90%)
r: Job = yes, Credit_history = good → Loan = Approved (sup = 40%, conf = 91%)

If we know *R*, then *r* is of limited use because it gives little extra information. Its slightly higher confidence is more likely due to chance than to true correlation.

We say *r* can be pruned with respect to *R* because within the subset of data tuples covered by *R*, *r* is not significant. The significance (or positive correlation) of a rule is measured by chi-square test (χ^2) for correlation from statistics [6]. The pruning is done as follows:

- Given a rule *r*, we try to prune *r* using each *ancestor rule R* (which has the same consequent as *r* but fewer conditions) of *r*. In the case that *r* (e.g., "$x \rightarrow c_i$") has only one conditions, its ancestor rule is "$\rightarrow c_i$". That is, we perform a χ^2 test on *r* with respect to *R*. If the test shows a positive correlation, it is kept. Otherwise, *r* is pruned. The reason for the pruning is because within the data covered by *R*, *r* is not significant.

See [14] for more details about pruning. Note that we do not use minimum confidence in our framework. Minimum confidence (a user-specified threshold) does not reflect the underlying relationships of the domain [3, 14]. Instead we use statistical significance as the basis for finding rules that represent the fundamental relations of the domain.

5.2 Representing Association Rules with GSE Patterns

Since association rule mining aims to find all rules in data, it often generates a huge number of rules. Even after pruning, the number of rules left can still be very large, in the hundred or thousands (see Section 6). Clearly, such a large number of rules are very difficult, if not impossible, to be analyzed by a human user.

We now discuss how to use GSE patterns to organize and present the discovered associations so that the user can browse through them and focus on the interesting subset of the rules. The key issue still is how to determine whether a rule should form an exception rule. In the case of a decision tree, we make the decision based on two criteria, *significance* (statistically significant and majority class) and *simplicity*. Statistical significance still applies (after pruning every association rule is statistically significant). However, we do not require that an exception rule's class be the majority class. Since the aim of association rule mining is to find all significant rules, thus we cannot require each rule to use the majority class.

The simplicity criterion does not apply here either because if we find a significant rule and we do not use it as an exception rule, then the rule is lost. This is undesirable. In the case of a decision tree, the original rules for classification are formed only at the leaf level. Thus, if an internal node does not form an exception rule, we do not lose any information.

Hence, for association rules, we only use statistical significance to determine whether a rule should form an exception (of course, its class must be different from its general rule). Since every rule is significant after pruning, we do not need any more significance test.

To convert association rules to GSE patterns, we also need to use a different technique. It is not time and space efficient to build the whole structure like the GSE tree for association rules. The reason is that this structure could be too large due to duplicate rules appearing in different parts of the structure, i.e., the same rule may appear in many places as exceptions.

For example, we have two general rules, (A1 = a → C1), and (A2 = b → C1). Assume that we also have the rule, (A1 = a, A2 = b → C3). Then, this rule will appear as an exception rule of A1 = a → C1 and as an exception rule of A2 = b → C1. In the case of a decision tree, no rule will be an exception rule for more than one general rule because rules in a decision tree are disjoint. This is not the case for association rules as the above example shows.

A question that one may ask is "why cannot we consider each rule as an exception only once?" We could not do this because the new technique aims to allow the user to browse and explore the rules effectively. If we have 2000 rules, obviously the user is unlikely to see all of them. GSE patterns organize the rules in an intuitive way so that the user can focus his/her attention on only a subset of interesting rules. However, we will not know where the user will choose to focus on beforehand. Thus, we are unable to decide whether the rule (A1 = a, A2 = b → C3) should appear as an exception of (A1 = a → C1) or (A2 = b → C1).

In this work, we adopt a lazy approach, i.e., we only compute the exceptions of a general rule when the user requests it. This approach is reasonable because of two reasons:

1. the computation can be done very efficiently without the user even noticing it (see the experiment results in Section 6);
2. even if we display all the information, the user will not be able to view all of it.

While finding the exception rules of a general rule, we also summarize all the rules below each exception rule. This summary highlights the main characteristics of the rules below the exception rule. From the summary, the user can decide whether to go down further to see more details. If he/she decides to do so, he/she can click and the system will compute the exceptions of this exception rule (which now behaves as a general rule).

The summary field contains the following information:

- The coverage of each class: This shows how big a population that each class covers. If a class covers too few data tuples, it may not be interesting.
- The highest confidence of the rules in each class: This gives the highest confidence of the rules below the exception rule. It indicates the quality of the subsequent rules.
- The number of rules in each class: This allows the user to decide whether to explore further using GSE representation or simply see the rules if the number of rules is small.

The set of characteristics can be further expanded when the need arises. However, in our applications, we found that these together with the exception rules are sufficient for the user to decide where to focus on.

Before presenting the algorithms for converting association rules to GSE patterns, we first show how the rules are stored in memory. In essence, they are represented as a tree, called a rule tree. A rule tree is different from a decision tree because every node in the rule tree can be attached with a class distribution list representing some rules, and the branches of the tree at each level do not partition the data. Each tree node contains three pieces of information:

1. An item, which is represented by an integer number.
2. A support count, which is the number of data tuples that fall into this node.
3. A class distribution list, which stores the number of data tuples falling into each class of this node. If the rule formed with the items from the root to the current node with a class is not a significant rule, we indicate it with a "0".

An example rule tree is given in Figure 5, which uses 4 items (1, 2, 3, 4). The class distribution list and the support counts are not shown. Note that the items (represented in the tree) are in an ascending order either horizontally or vertically. This is inherited from association rule mining. This ordering greatly facilitates our computation later.

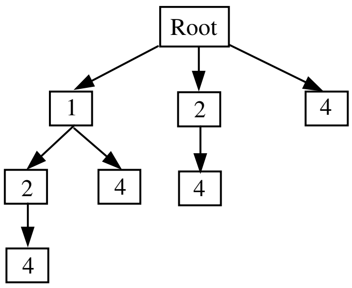


Figure 5. An example rule tree (no rule formed with item 3)

We now present the overall algorithm, findGSE (Figure 6), to find all the exception rules of a general rule. The general rule is represented by the root node rootNode of a rule tree and the class grClass of the general rule. We will see later that a tree is constructed dynamically for each general rule, which is used to find its exceptions. The tree for a general rule contains only those rules (called superset rules below) that are below the general rule. The top-level general rules can also be seen as exception rules that do not have a general rule above them.

```
Algorithm findGSE(rootNode, grClass)
1  exceptionR = {};
2  findErules(rootNode, grClass);
3  for each er ∈ exceptionR do
4      supersetR = {};
5      findSupersetR(er.cond, er.class, rootNode);
6      summarize(supersetR);
7      buildTree(cond(er), supersetR)
8  endfor
```

Figure 6. The overall findGSE algorithm

Line 1 initializes a variable *exceptionR* to store all the exception rules of a general rule. The findErules procedure (line 2) finds all the exception rules starting from *rootNode*. From line 3-8, for each exception rule *er*, we find all the rules below it, i.e., those rules whose conditions are a superset of the conditions of *er* (line 5). We call them the *superset rules* of *er*. *er.cond* gives the set of conditions of *er*, and *er.class* gives the class of *er*. Line 4 initializes a variable to store all the superset rules of *er*. Line 6 summarizes the superset rules of each class. This procedure is simple to design, and thus will not be discussed further. In line 7, we build a new tree using all rules in *supersetR* with all the conditions in *er* removed from each rule. This tree will be used to find exceptions below *er* (*er* will behave as a general rule). That is why the algorithm findGSE always starts from the root node of a tree. The buildtree procedure is omitted, as it is straightforward.

The findErules procedure (Figure 7) finds all exception rules from a root node representing a general rule with its class *grClass*. For finding top-level general rules, the same procedure is used. In this case, *grClass* is a special value (we use 0) that is not a class in the data.

In line 2, findErules descends down the tree to find exception rules from each child node. If an exception rule is found from a child (line 3), it is recorded (see getER procedure in Figure 8). Otherwise, the procedure traverses further down to find exception rules. The procedure (getER) for finding exception rules within a node is given in Figure 8.

```
findErules(rootNode, grClass)
1  if rootNode.children ≠ NIL then
2      for each child in rootNode.children do
3          if not(getER(child, grClass)) then
4              findErules(child, grClass);
```

Figure 7. Finding exception (or general) rules

The getER procedure (Figure 8) uses the class distribution list of the node (in *node.classDistr*) to see whether any rules are formed here. If $c = 0$, it means that this class does not form a rule. If $c \neq 0$ and c is not the same as *grClass*, an exception rule is found (line 4-5). The variable *found* indicates whether any exception rules are found.

```
getER(node, grClass)
1  found = FALSE
2  for each c in node.classDistr do
3      if c ≠ 0 then
4          if c ≠ grClass then  /* we have found an exception rule */
5              insert the exception rule to exceptionR;
6              found = TRUE
7          endif
8      endif
9  endfor
10 return(found)
```

Figure 8. Finding exception or general rules from a node

Figure 9 gives the findSupersetR procedure that finds all superset rules of an exception rule. These rules will be summarized to the user. Assume the set of conditions of an exception rule is $S = \{s_1, \ldots, s_k\}$, which is a set of numbers sorted in the ascending order. findSupersetR starts off from the root node of the current tree and traverses down every possible path of the tree to find all superset rules. Note that we cannot simply traverse down the tree from the exception rule to find its superset rules because some of its superset rules may appear in other parts of the tree as well.

Figure 10 gives an example organization of a set of rules, which shows how general rules, summaries and exceptions are organized and presented. The rules are generated from a loan application data. The classes in the data are *Approved* and *Not_Approved*, indicating whether a loan is approved or not. Lines 1, 5, 7 and 11 show 4 general rules. Lines 2-3, line 6, lines 8-9, and lines 12-13 summarize the rules below the 4 general rules

```
Job = Yes → Loan = Approved                              (sup = 34%; conf = 75%)
        Summary   Approved:      coverage = 34%, highest confidence = 100%, no of rules = 4;
                  Not_Approved:  coverage = 13%, highest confidence = 96%, no of rules = 2;
    Except …
Own_house = Yes → Loan =  Approved                       (sup = 24%; confidence = 100%)
        Summary   Approved:      coverage = 24%, highest confidence = 100%, no. of rules = 1;
Credit_history = bad → Loan = Not_Approved               (sup = 20%; confidence = 67%)
        Summary   Approved:      coverage = 10%, highest confidence = 100%, no. of rules = 2;
                  Not_Approved:  coverage = 20%, highest confidence = 100%, no. of rules = 2;
    Except …
Job = No, Saving = No → Loan = Not_Approved              (sup = 15%; confidence = 75%)
        Summary   Approved:      coverage = 5%, highest confidence = 100%, no. of rules = 3;
                  Not_Approved:  coverage = 15%, highest confidence = 100%, no. of rules = 2;
    Except …
```

Figure 10. An example GSE representation of association rules

respectively. "Except …" indicates that there are exceptions, which are not shown. The user needs to click on it to see the exception rules, which calls the findGSE procedure. The user can also click on the class *Approved* or *Not_Approved* in the summary field to see all rules of the class below each general rule.

```
findSupersetR(S, node)
    If node.children ≠ NIL then
        for each child in node.children do
            findFirst(hd(S), tail(S), child)
                    /* hd(S) gives the first element of S */
        endfor      /* tail(S) gives the rest of the elements of S */
    endif

findFirst(first, rest, node)
    If first ≤ node.id then   /* node.id is the item number */
        If first = node.id then
            If rest = ∅ then getRules(node)
            else findSupersetR(rest, node)
        else findSupersetR({first} ∪ rest, node)
    endif

getRules(node)
    for each c ∈ node.classDistr do
        if c ≠ 0 then insert the rule into supersetR endif
                    /* a superset rule is found */
    endfor
    goDown(node)

goDown(node)
    if node.children ≠ NIL then
        for each child of node.children do
            getRules(child)
        endfor
    endif
```

Figure 9. Finding all superset rules

5.3 Some Special Problems

We now discuss three problems in using the GSE representation to organize and summarize association rules. The solution to these problems is also presented.

1. Due to the minimum support constraint some exceptions for a general rule may not be discovered by the mining algorithm. This causes problems in understanding the domain because it may mislead the user to believe that no exception exists.
2. Due to pruning in rule generation, some exceptions may be pruned but their existence is important to the understanding of the domain. For example, we have the following two rules:
   R1: A1=a, A2=b → C1 (sup = 5%, conf = 98%)
   R2: A1=a, A2=b, A3=e → C1 (sup = 2%, conf = 98.2%)
   If R1 exists, R2 is pruned because R2 is insignificant with respect to R1. This pruning is clearly reasonable. Otherwise, we will end up with a huge number of insignificant rules. However, this pruning may cause problems for the user in understanding the domain, in particular, the aspect of the domain related to A1=a, A2=b, A3=e. For instance, if the user is inspecting the following rule:
   R3: A1=a, A3=e → C2 (sup = 15%, conf = 70%)
   No exception to R3 is found by the system. This is misleading because R2 is an obvious exception. R2 is not reported as an exception because it has been pruned. This is undesirable. One may argue that R2 should indicate to the user that R3 has exceptions, but the user may not know R2 when he/she is drilling down to R3 [2].
3. Due to overlapping of rules, it is harder to find good exceptions. Overlapping of rules means that the same data tuples may be covered by many rules. The user typically likes to find disjoint rules that cover the exception tuples. Note that rules in a decision tree are disjoint.

We deal with these problems using two methods.

Lower down the minimum support: When the user is interested in a general rule and its direct exceptions, he/she can request the system to extract the data tuples that satisfy the conditions of the general rule, and to use this subset of data to run the association rule miner again with a lower minimum support. The output rules are organized and presented to the user in the same way as described above. This method is effective in finding good exceptions if the items involved is not too many so that a very low minimum support can be used, otherwise it can still cause combinatorial explosion. The method does not help to solve the third problem.

Integrate with a decision tree algorithm: Extract the data tuples that satisfy the conditions of the general rule, and use this subset of data to run a decision tree system. The output rules (or the tree) are presented to the user in the same way as described in Section 4. This method is not constrained by the minimum support. The output rules are also disjoint. The disadvantage is that it does not find all possible rules.

[2] In our initial applications, the first two situations surfaced quite a few times and the users thought that there were bugs in the system.

6. EMPIRICAL EVALUATION

We now evaluate the effectiveness and efficiency of the proposed technique. We first discuss the experiment results with decision trees, and then the experiment results with association rules. Finally, we discuss our application experiences.

6.1 Experiments with Decision Trees

We applied the proposed technique to the decision trees produced by C4.5 [22] using 20 datasets in the UCI Machine learning repository [17]. Our objective here is to see how much the GSE representation is able to simplify the decision tree. Simplification, however, is not the only objective of the proposed technique. Another important aspect is the organization of the discovered knowledge. Unfortunately, organization is hard to quantify. We use our application experiences to show that users like the proposed technique due to its intuitive representation.

Table 1 shows the average results over the 20 datasets. See [15] for the detailed results. The first column gives the average number of leaves in the decision trees produced by C4.5 (after pruning) for the 20 datasets. The second column gives the average number of leaves in the GSE trees for the 20 datasets.

Table 1: Experiment results with decision trees

No. of decision tree leaves	No. of GSE tree leaves
105.2	38.1

From Table 1, we can see that the number of leaves (or rules) in the GSE tree is substantially smaller. On average, over the 20 datasets, the number of leaves in the GSE tree is only 36% of that of the original decision tree. This shows that GSE trees significantly simplify decision trees. The execution times are not reported as the algorithm runs so fast that they cannot be logged.

In the experiments, chi-square test at the significance level of 95% is used to measure the significance of a rule (this is a commonly used level [6]). When an expected frequency is below 5 for chi-square test, fisher's exact test is employed instead.

6.2 Experiments with Association Rules

For the experiments with association rules, it is not appropriate to use the percentage of reduction to assess the effectiveness because there are duplicate exception rules in many places and our implementation is based on a lazy approach. The system only finds exception rules of a general rule when requested by the user. In this situation, efficiency is important because we cannot let the user wait for too long to obtain the results. We show the execution time taken to compute the exceptions of a general rule, and to find and summarize all their superset rules. The datasets used here are the same as those used in the experiments with decision trees. The experiments are done on Pentium-II 350 with 128MB of memory.

We use the system in [14] to generate association rules and to prune those insignificant rules (without generating the DS rules). The minimum support is set to 1% as it is shown in [13] that for these datasets, rules with this support threshold are sufficiently predictive. Pruning of association rules also uses the χ^2 value at significance level of 95%. Many datasets we use contain numeric attributes. Since association rule mining does not handle numeric attributes. We discretize these attributes into intervals using the target (class) attribute. We use the entropy-based method in [7]. The code is taken from MLC++ [10].

Table 2 shows the execution time taken to find all top-level general rules and their superset rules (including summarization) for each dataset. We only use the top-level general rules to give an indication of efficiency because the number of top-level general rules is normally larger than the number of exceptions of a general rule below them. They also tend to have more superset rules as the number of conditions in a top-level general rule is small.

In Table 2, column 1 gives the name of each dataset. Column 2 gives the number of rules found after pruning for each dataset. Although the number of rules left can still be quite large, it does not present a big problem any more as we will see in Section 6.3. Column 3 shows the number of top-level general rules found for each dataset. Column 4 gives the execution time. We can see that the execution is very fast. The user can hardly notice that the computation is done on the fly.

Table 2: Execution time for finding top-level general rules and summarizing their superset rules

	Dataset	no. of rules after pruning	no.of top level general rules	execution time (sec.)
1	adult	1452	58	0.09
2	anneal	338	32	0.04
3	austra	329	26	0.02
4	auto	1872	61	0.09
5	breast	96	19	0.02
6	chess	1814	46	0.07
7	cleve	180	19	0.02
8	crx	393	28	0.04
9	diabetes	44	12	0.02
10	german	387	24	0.04
11	glass	153	14	0.02
12	heart	131	16	0.02
13	iono	646	91	0.05
14	kdd	5914	68	0.22
15	mushroom	2398	57	0.08
16	pima	44	12	0.01
17	satimage	7515	191	0.30
18	splice	4302	100	0.15
19	tic-tac	266	11	0.02
20	waveform	1480	106	0.07
	Average	*1487.7*	*50*	*0.07*

6.3 Applications

We built two systems in this work, one for converting a decision tree to a GSE tree (we call this system GSEtree) and one for organizing, summarizing and presenting association rules (we call this system GSErule). GSEtree is also integrated into GSErule, i.e., GSErule can call GSEtree when needed (see Section 5.3). The two systems have been used in a number of applications for two educational institutions, and a medical center.

The GSEtree system was used in a number of classification tasks. For example, in a medical application, we needed to build a classifier to predict whether a person has a particular kind of disease. We used the C4.5 system, which produced a decision tree with 35 leaves. Our GSE tree had only 10 leaves or rules. For example, a general rule says that if the patient's age is less than 44, the chance of having the disease is small with 3 exceptions. However, the corresponding sub-decision tree has 14 leaves (or rules). 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 produced 5 fragmented rules. The doctors said that they could not obtain an overall picture of the domain from the fragmented decision tree leaves (or rules). The GSE tree gives them exactly the types of knowledge they want. In the education application, the situation is similar. Our users agree that the GSE representation is simple and more intuitive.

The GSErule system was used in all our applications. The users find it more effective than our existing systems. Before this work, we had two previous systems. The first system is based on user expectations or beliefs [12]. In this system, the user first inputs some existing knowledge, and the system then finds those conforming and unexpected rules. The main problem with this system is that the users found it difficult to give existing domain knowledge. Very quickly they gave up and asked us to do it for them. The second system is the DS rule system [14]. DS rules are the essential rules in the domain. Since the number of DS rules is not very large in these applications, less than one hundred, they can be browsed manually. However, it is still hard to gain a good overall view of the domain using the DS rule system since the DS rules are just a list of rules with no organization. The expectation system and the DS rule system are also hard to understand. It often takes many sessions for a student to explain to our users what these systems do. The GSErule system, on the other hand, is much easier to understand and to use because it is very intuitive. In fact, our users had inspired us to design the new representation. With the GSE representation and the GSErule system, browsing through the rules is no longer so difficult and boring as the exception (unexpected) rules always draw the users' attention.

The new systems also provide a convenient tool for us, as we need to identify interesting rules for our users in some applications. Our users could only explain to us the types of rules that may be interesting to them. Using the GSErule system, we are able to identify interesting rules easily. We can also present the rules to our users in the way (using the GSE representation) that they can easily understand and appreciate.

In all our applications, we typically have 200-1000 association rules after pruning. The large number of rules does not present a major problem as the users always have some tasks in mind and thus are only interested in certain aspects of the discovered rules. The GSE representation makes it convenient to focus on these interesting aspects and to obtain a good overall picture of them. It is also quite interesting to note that even those less interesting aspects are often explored because exceptions (unexpected rules) can be quite tempting, i.e., people are always curious about unexpected things.

7. CONCLUSION

This paper proposed a novel technique to organize, summarize and present the discovered rules. The technique is based on the representation and organization of knowledge using general rules, summaries and exceptions, i.e., GSE patterns. The GSE pattern is very intuitive as it is closely related to the way that we think and talk about knowledge in our daily lives. It allows the user to see an overall picture of the domain first and then the special cases. It manages the complexity of the discovered rules in a hierarchical fashion, which enables the user to focus his/her attention on the interesting aspects of the rules. In this paper, we applied the representation to organize and summarize the knowledge embedded in a decision tree and a set of association rules. Our practical applications show that the representation is simple and more intuitive than a list of individual rules.

Acknowledgement: We thank Shuik-Ming Lee, Shanta C Emmanuel, Paul Goh, Jonathan Phang, Hing-Yan Lee 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

[1] Adomavicius, G. and Tuzhilin, A. "User profiling in personalization applications through rule discovery and validation." *KDD-99*, 1999.

[2] Agrawal, R. and Srikant, R. "Fast algorithms for mining association rules." *VLDB-94*.

[3] Bayardo, R., Agrawal, R, and Gunopulos, D. "Constraint-based rule mining in large, dense databases." *ICDE-99*.

[4] Compton, P. and Jansen, R. Knowledge in context: a strategy for expert system maintenance. *AI-88*, 1988.

[5] Dong, G. and Li, J. "Interestingness of discovered association rules in terms of neighborhood-based unexpectedness," *PAKDD-98*. 1998.

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

[7] Fayyad, U. M. and Irani, K. B. "Multi-interval discretization of continuous-valued attributes for classification learning." *IJCAI-93*, 1993.

[8] Han, J. and Fu, Y. "Discovery of multiple-level association rules from large databases." *VLDB-95*, 1995.

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

[10] Kohavi, R., John, G., Long, R., Manley, D., and Pfleger, K. "MLC++: a machine learning library in C++." *Tools with artificial intelligence*, 1994.

[11] Large, A. Tedd, L. & Hartley, R *Information seeking in the online age: principles and practice.* Bowker. 1999.

[12] Liu, B., Hsu, W. "Post-analysis of learnt rules" *AAAI-96*.

[13] Liu, B., Hsu, W. and Ma, Y. "Integrating classification and association rule mining." *KDD-98*, 1998.

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

[15] Liu, B., Hu. M., and Hsu, W. "Intuitive representation of decision trees as general rules and exceptions." *AAAI-2000*.

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

[17] Merz, C. J. & Murphy, P. UCI repository of ML databases, 1996. [http://www.cs.uci.edu/~mlearn/MLRepository.html].

[18] Ng. R. T. Lakshmanan, L., and Han, J. "Exploratory mining and pruning optimizations of constrained association rules." *SIGMOD-98*, 1998.

[19] Padmanabhan, B., and Tuzhilin, A. "A belief-driven method for discovering unexpected patterns." *KDD-98*.

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

[21] Piatesky-Shapiro, G., and Matheus, C. "The interestingness of deviations." *KDD-94*.

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

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

[24] Srikant, R., Vu, Q. and Agrawal, R. "Mining association rules with item constraints." *KDD-97*.

[25] Suzuki, E. "Autonomous discovery of reliable exception rules." *KDD-97*, 1997.