Mining Unexpected Rules by Pushing User Dynamics *

Ke Wang
Simon Fraser University
wangk@cs.sfu.ca

Yuelong Jiang
Simon Fraser University
yjiang@cs.sfu.ca

Laks V.S. Lakshmanan
University of British Columbia
laks@cs.ubc.ca

ABSTRACT

Unexpected rules are interesting because they are either previously unknown or deviate from what prior user knowledge would suggest. In this paper, we study three important issues that have been previously ignored in mining unexpected rules. First, the unexpectedness of a rule depends on *how* the user prefers to apply the prior knowledge to a given scenario, in addition to the knowledge itself. Second, the prior knowledge should be considered right from the start to focus the search on unexpected rules. Third, the unexpectedness of a rule depends on what *other* rules the user has seen so far. Thus, only rules that remain unexpected given what the user has seen should be considered interesting. We develop an approach that addresses all three problems above and evaluate it by means of experiments focusing on finding interesting rules.

1. INTRODUCTION

Data mining aims at finding previously unknown and interesting rules for the user. Most data mining algorithms use *objective measures* of interestingness, such as statistical significance [14, 15], Chi-square test [6] and support/confidence [2]. Often, rules that satisfy such measures are known and uninteresting to the human user [7, 8, 11]. For example, a relationship between two items is deemed interesting by the Chi-square test if it is significantly stronger than expected when items are assumed to be independent, but not interesting to the human user who *knows* such or similar relationships. On the other hand, interesting rules may not satisfy an objective measure. For example, the minimum support requirement in [2] tends to prune unknown association rules that often do not have a large support.

To find interesting rules, *subjective measures* taking into account what the user knows have been suggested [15, 17, 18]. One subjective measure is *unexpectedness*, where a rule is unexpected if it "surprises" the user. Another subjective measure is *actionability*, where a rule is actionable if the user can act upon it to her advantage. Due to the difficulty of modeling the user knowledge, works on subjective measures are much fewer [8, 9, 11, 13, 16, 15]. This paper focuses on the unexpectedness measure. Two previous approaches, the *syntax based* [8, 9] and the *logic based* [12], follow the paradigm of pairwise comparison: compare a rule r found in the data against a *single* rule representing the user knowledge. If the comparison reveals a syntax distance (i.e., a similar body but a dissimilar head) or a logical contradiction (i.e., a specialized body but a negated head), r is considered unexpected.

In this paper, we examine three important issues that have been previously ignored in mining unexpected rules.

Knowledge dynamics: Whether a rule is unexpected depends on how the user intends to apply the prior knowledge to a given scenario, in addition to the prior knowledge itself. To realistically model the user intention and her intuition of unexpectedness, it is critical to consider this dynamic aspect.

Knowledge push: The prior knowledge should be considered right from the start of search to prune uninteresting rules as early as possible. This has the twin advantage of (a) making the algorithms scale better and (b) presenting the user with a much smaller set of rules to examine, where many candidates are more likely to be unexpected.

Unexpectedness dynamics: The unexpectedness of a rule depends on what other rules have been previously presented to the user. In fact, a rule seen earlier could render some remaining rules no longer unexpected. It is important to consider these dynamics if we want to be faithful to the notion of what is truly unexpected.

1.1 The motivating example

The following example motivates our approach.

EXAMPLE 1.1 (UNEXPECTED RULES). Consider finding unexpected rules about how people make donation in a fund raising campaign. Suppose that the user believes, based on past experience, that movie stars tend to donate more than $500 and well paid people tend to donate more than $200, represented as the "knowledge rules":

*Research was supported in part by a research grant from the Networks of Centres of Excellence/Institute for Robotics and Intelligent Systems, and in part by a research grant from the Natural Science and Engineering Research Council of Canada

SIGKDD '03, August 24-27, 2003, Washington, DC, USA.

$R_1 : Salary = high \rightarrow Donation = Above\_200$,
$R_2 : Job = movie\_star \rightarrow Donation = Above\_500$.

Suppose that the following "data rule" is found from the data:

$r : Loc = BH, House = yes \rightarrow Donation = Below\_50$,

which says that people owning houses in Beverly Hills tend to donate less than \$50. r would not be identified as unexpected by the syntax based approach [8, 9] because it is not related in syntax to the knowledge rules. Neither would r be identified as unexpected by the logic based approach [11] because it does not logically contradict the knowledge rules.

However, suppose the user knows that Beverly Hills is an expensive place and the home of movie stars, thus decides it makes sense to apply rules R_1 and R_2 to the subpopulation summarized by r, viz., people owning a house in Beverly Hills. Her rationale might be these rules describe donation patterns by wealthy people and rule r concerns that as well. Thus, she prefers to compare r with rules R_1 and R_2, to judge if the former is unexpected. This comparison reveals that r is indeed unexpected, since its conclusion is significantly different from that of R_1, R_2 with respect to expected donation levels. □

The above example illustrates several interesting points that motivate our work.

1. Given a data rule r, often the human user has her own "best knowledge rules" for the subpopulation summarized by r and judges the unexpectedness of r based on such best knowledge rules. The specification of best knowledge rules is essentially a *preference model* that, given a scenario (e.g., a tuple), triggers a mental search for best knowledge rules about the scenario. The essence of the preference model is to link a given data rule to relevant user knowledge, possibly consisting of more than one rule.

2. To simulate the above user process of determining the unexpectedness of a given data rule r, we can ask the user to specify the preference model for identifying best knowledge rules, called *covering knowledge*, for a given tuple. Each tuple that satisfies r but "violates" its covering knowledge provides an evidence that r is unexpected with respect to the user knowledge. We can then measure the unexpectedness of r by aggregating such evidences over all tuples satisfying r.

3. The above approach emphasizes two important aspects of unexpectedness: the *user-initiated knowledge preference* and the *data-evidenced knowledge violation*. These aspects capture more closely the intention of unexpectedness in real life situations. For example, a data rule can now be unexpected on the basis of summarizing a subpopulation that violates *several* knowledge rules preferred by the user. This is not possible in the previous approaches [8, 9, 12] that must choose a *single* knowledge rule to compare against a rule found in the data.

Remarks. One may argue that the syntax and logic based approaches [8, 9, 12] might identify the *specialized* rules of r as unexpected, obtained by adding the conditions of R_1 and R_2 to r. However, these specialized rules may not be found in the first place because they may not pass the objective measures required (such as the minimum support) in those approaches (say, if movie stars are paid by contract, not by salary). Also, these rules are less desirable than the simple structure r because they fragment a single large violating subpopulation into several small violating subpopulations.

1.2 The proposed approach

We present a new approach to the problem of mining unexpected rules. First, we propose a new notion of unexpectedness by incorporating a preference model specified by the user. The preference model is either a defining criterion or an enumeration of the covering knowledge. Second, we present an algorithm for mining all unexpected rules that satisfy user-specified minimum "unexpectedness significance" and minimum "unexpectedness strength". The algorithm pushes the significance requirement and examines only those rules that satisfy this requirement. Finally, we present a method for selecting a specified number of most unexpected rules, taking into account the unexpectedness dynamics discussed earlier.

The rest of the paper is organized as follows. Section 2 reviews related works. Section 3 presents our representation of user knowledge. Section 4 defines the problem of mining unexpected rules and the problem of selecting unexpected rules. Section 5 presents an algorithm for mining unexpected rules, and Section 6 presents an algorithm for selecting unexpected rules. Section 7 evaluates the effectiveness of the proposed methods. Section 8 concludes the paper.

2. RELATED WORK

Interestingness measures have been studied in the field of data mining [3, 6, 7, 8, 13, 20, 16, 15, 17, 18], with most works on objective measures [20]. Mining unexpected rules was studied in [8, 9, 11, 13, 16, 15]. None of these works considers the user preference in applying knowledge rules. The second difference is that we consider the user knowledge at the beginning of search. In contrast, the post-analysis approach [8, 9, 16] considers the user knowledge at the end of search, as one additional filter to the result found by an objective measure method. In this respect, our approach is similar to [11], but the different notion of unexpectedness makes the two studies very different. Finally, we consider the unexpectedness dynamics discussed earlier. This is quite different from the work on eliminating redundant rules within the framework of objective measures (e.g., [23]).

The work on mining association rules [2] searches for rules of support above a given threshold. The support measures the frequency of a rule in the data, which does not consider the novelty with respect to the user knowledge. Consequently, if the support threshold is too high, unexpected rules may not pass the threshold, and if the support threshold is too low, many rules known to the user are returned. The correlation approach such as [6] measures the support relative to the independence assumption, which makes sense only if the user knows nothing about the domain. Our notion of unexpectedness significance captures the part of support that is responsible for violating the user knowledge. In this sense, it addresses the novelty of rules.

The work on constrained data mining [4, 10, 19, 21, 22] considers *what the user wants*, i.e., constraints, and searches for rules that satisfy the specified constraints. The unexpected rule mining considers *what the user knows*, i.e.,

knowledge, and searches for rules that surprise the user with new information. Unexpected rules may not satisfy the specified constraints, and satisfying rules may not be unexpected. Indeed, it is a rather difficult task for the user to specify what she wants for something that she wishes to be a surprise.

3. USER KNOWLEDGE

We describe our representation of user knowledge. Consider a relational table T of tuples over several attributes. Mining in such data is typically targeted at a specific attribute because the user normally wants to know how other attributes are related to this *target* attribute. Our objective is to find *data rules* of the form

$$A_1 = a_1, \ldots, A_k = a_k \rightarrow C = c,$$

where A_i is a non-target attribute, a_i is a domain value for A_i, C is the target attribute, c is a domain value for C. We refer to c as the *class*. $A_1 = a_1, \ldots, A_k = a_k$ is the *body* and $C = c$ is the *head* of the rule. For a data rule r, $b(r)$ denotes the body of r and $h(r)$ denotes the head of r.

A tuple *matches* a data rule r if $b(r)$ holds on the tuple. A tuple *satisfies* a data rule r if both $b(r)$ and $h(r)$ hold on the tuple. $MAT(r)$ denotes the set of tuples that match r and $SAT(r)$ denotes the set of tuples that satisfy r. $sup(r)$, called *support* of r, denotes the fraction of tuples that satisfy r, i.e., $sup(r) = |SAT(r)|/|T|$. $sup(b(r))$ denotes the fraction of tuples that match r, i.e., $sup(b(r)) = |MAT(r)|/|T|$. $conf(r)$, called *confidence* of r, denotes the fraction of tuples that satisfy r given that they match r, i.e., $conf(r) = sup(r)/sup(b(r))$.

We represent the user knowledge $\mathcal{K}$ in two parts, the knowledge rules and the preference model.

3.1 Knowledge rules

While data rules are of the form above, we allow for slightly more general form for the rules capturing user's prior knowledge, called *knowledge rules* and denoted $\mathcal{K}$, by allowing fuzzy terms like "high", "low", etc., as suggested in [8, 9]. It will then be necessary to define a degree of match between domain values appearing in data rules and in data tuples on the one hand and these fuzzy terms appearing in knowledge rules on the other. We use the upper case letter R for a knowledge rule, and use the lower case letter r for a data rule. Like for a data rule, $b(R)$ and $h(R)$ denote the body and head of a knowledge rule R. $m(v, v')$ measures the *match degree* between a domain value v and a user value v' (for the same attribute), which can be either specified by the user or chosen from a pre-determined list. The match degree is in the range [0,1], with a larger value corresponding to a better match. The following is an example of knowledge rules involving fuzzy terms.

EXAMPLE 3.1 (DEGREE OF MATCH). Suppose that the domain of the attribute Education is

$$\{Primary, Secondary, University, Graduate\}.$$

A fuzzy term *Low* (serving a user value) could be described by a *fuzzy set*

$$\{(Primary, 1), (Secondary, 0.5)\}.$$

1 and 0.5 are the match degrees between the domain values $Primary, Secondary$ and the fuzzy term *Low*. So, $Primary$ perfectly matches *Low*, and $Secondary$ partly matches *Low*. Since the domain values $University$ and $Graduate$ are not listed in the fuzzy set, they do not match *Low*, i.e., the match degree is 0. Suppose that a knowledge rule r has the form $Education = Low \rightarrow Loan = No$. A tuple t containing $Education = Secondary$ and $Loan = Yes$ will have a body match degree of 0.5 and head match degree of 0. □

The purpose of the match degree measure m is to define the notions of match and violation between a tuple and a knowledge rule. First, consider two bags of values: $V = \{v_1, \cdots, v_d\}$ and $V' = \{v'_1, \cdots, v'_d\}$, where v_i and v'_i are in the range [0,1]. We write $V \leq V'$ if for $1 \leq j \leq d$, there is a distinct i_j such that $v_j \leq v'_{i_j}$. An aggregate function agg is *well-behaved* if (1) $min(V) \leq agg(V) \leq max(V)$, and (2) for $V \leq V'$, $agg(V) \leq agg(V')$. For example, $avg, max, min, medium$ are well-behaved, but not sum because it is not in the range [0,1].

DEFINITION 3.1 (RULE MATCH). Consider a tuple t, a knowledge rule R and a well-behaved aggregate function agg. $bm(t, R) = agg(\{m(v, v')\})$ measures the *body match degree* of t with R, where v and v' are the values in t and R over the attributes in $b(R)$. $hm(t, R)$ measures the *head match degree* of t with R, defined as the match degree m of the class of t and $h(r)$. $bm(t, R)$ and $hm(t, R)$ are in the range [0,1], with a larger value corresponding to a better match. Let $\overline{hm}(t, R) = 1 - hm(t, R)$. For a given threshold σ, we say that t *matches* R if $bm(t, R) \geq \sigma$. □

The intuition is that the more the body matches degree and the less the head matches degree, the more the violation. This is formalized below.

DEFINITION 3.2 (SINGLE KNOWLEDGE RULE VIOLATION). The *violation* of R by t, denoted $v(t, R)$, is defined as

$$v(t, R) = \begin{cases} \overline{hm}(t, R) \times bm(t, R) & \text{if } bm(t, R) \geq \sigma \\ & \wedge \overline{hm}(t, R) \geq \sigma' \\ 0 & \text{otherwise} \end{cases}$$ □

Other functions are possible, such as $\sqrt{\overline{hm}(t, R) \times bm(t, R)}$, as long as they capture the intuition of violation. We leave the choice to the application. $v(t, R)$ essentially depends on the measure of match degree m, of which we also leave the choice to the application. Importantly, our discussion below does not depend on these choices.

3.2 The preference model

The *preference model* specifies the user's knowledge about how to apply knowledge rules to a given scenario or a tuple. This is done by specifying the "covering knowledge" for each tuple. For a given tuple, the *covering knowledge* refers to one or more knowledge rules that match the tuple and the user prefers to apply to the tuple. The *covering depth* d refers to the number of knowledge rules in the covering knowledge, usually a small integer such as 1, 2, or 3. The reason for allowing the covering depth d greater than 1 is simple: the user sometimes has several (preferred) knowledge rules about a scenario and want to factor in all of them in the unexpectedness measure. This does not mean that the total number of knowledge rules will be $|T| \times d$, because the covering knowledge for different tuples is not required to be

disjoint. Here are several examples of specifying the covering knowledge: (1) enumerating the covering knowledge for each tuple by the user, (2) ranking knowledge rules so that the covering knowledge is the first d matching knowledge rules in the list, (3) specifying a preference measure, such as *maximum strength preference* (e.g, belief degree, confidence, etc.), *best match preference* (i.e,, maximizing $m(t, R)$), or *minimum violation preference* (i.e,, minimizing $v(t, R)$). (1) is not scalable for large databases, whereas (2) and (3) are because they can be automated.

EXAMPLE 3.2. Suppose that a banker has the following knowledge rules about loan approval, ranked in this order:

$Own\_house = yes \rightarrow Loan = yes$,
$Job = no \rightarrow Loan = no$.

For an applicant who owns a house but has no job, the first knowledge rule will serve the covering knowledge (of covering depth 1), which implies the application is expected to be approved. If the rank is reversed, $Job = no \rightarrow Loan = no$ will be the covering knowledge of the applicant instead, which implies the different expectation that the application should be rejected. This example shows how the user applies knowledge rules will affect the user's expectation, and hence the unexpectedness of the rules mined from the data. □

The above has assumed that at least d knowledge rules match a given tuple, where d is the covering depth. In general, we can add d duplicates of the *default rule*, denoted $NULL$, to the user knowledge $\mathcal{K}$, with $bm(t, NULL) = 1$ and $hm(t, NULL) = 0$ for any tuple t. The default rules are used only if there are not enough matching knowledge rules, therefore, are least preferred in any preference model. The choice of $bm(t, NULL) = 1$ and $hm(t, NULL) = 0$ implies $v(t, NULL) = 1$ for any tuple t. Hence, with the lowest preference, the default rules implement the *unexpectedness presumption* in the absence of knowledge.

4. PROBLEM STATEMENTS

4.1 Unexpected rule mining problem

With the notion of covering knowledge, we can now measure the violation of user knowledge $\mathcal{K}$ by a data tuple t.

DEFINITION 4.1 (VIOLATION OF USER KNOWLEDGE). Let $\mathcal{C}_t$ be the covering knowledge of a tuple t with respect to $\mathcal{K}$. The *violation* of t with respect to $\mathcal{K}$ is defined by

$$v_{\mathcal{K}}(t) = agg(\{v(t, R) \mid R \in \mathcal{C}_t\})$$

for some well-behaved aggregate function agg. □

To define the notion of unexpectedness of a data rule r, we consider each tuple t that satisfies r but violates the covering knowledge of the tuple as an evidence that r is unexpected with respect to $\mathcal{K}$. This evidence is quantitatively measured by $v_{\mathcal{K}}(t)$. For r to be considered as unexpected, there must be "sufficient" evidence in the data as formalized by several measures defined below.

DEFINITION 4.2 (UNEXPECTEDNESS MEASURES). Consider a database T, a data rule r, and the user knowledge $\mathcal{K}$. The *unexpectedness support* of r with respect to $\mathcal{K}$ is defined by

$$Usup(r) = \Sigma\{v_{\mathcal{K}}(t) \mid t \in SAT(r)\}/|T|.$$

The *unexpectedness confidence* of r with respect to $\mathcal{K}$ is defined by

$$Uconf(r) = Usup(r)/sup(b(r)).$$

The *unexpectedness* of r with respect to $\mathcal{K}$ is defined by

$$Unexp(r) = Usup(r)/sup(r).$$

(Note that $Usup(r), Uconf(r), Unexp(r)$ are in the range [0,1].) □

Remarks:

1. $Usup(r)$ measures the unexpectedness significance in terms of *the violation in the whole database*. Since the violation is measured using the tuples that satisfy r, $Usup(r)$ captures the part of the support $sup(r)$ responsible for violating the user knowledge. $Usup(r) = sup(r)$ if each tuple that satisfies r has a perfect violation (i.e., $v_{\mathcal{K}}(t) = 1$).

2. $Uconf(r)$ measures the unexpectedness strength in terms of *the violation in a tuple that matches* r. A high $Uconf(r)$ implies that a matching tuple tends to satisfy r and a satisfying tuple tends to violate the user knowledge. $Uconf(r) = conf(r)$ if each tuple that satisfies r has a perfect violation.

3. $Unexp(r)$ measures the unexpectedness strength in terms of *the violation in a tuple that satisfies* r. A high $Unexp(r)$ implies that a satisfying tuple of r tends to violate the user knowledge. Unlike $Uconf(r)$, $Unexp(r)$ does not factor in the likelihood that a matching tuple satisfies r. $Unexp(r) = 1$ if each tuple that satisfies r has a perfect violation.

We use the notation $Ustr(r)$ to denote $Uconf(r)$ or $Unexp(r)$, when we don't want to distinguish between them.

DEFINITION 4.3 (UNEXPECTED RULE MINING PROBLEM). Given a database T, the user knowledge $\mathcal{K}$ and thresholds σ_1, σ_2 (in the range [0,1]), the *unexpected rule mining* is to find all data rules r such that $Usup(r) \geq \sigma_1$ and $Ustr(r) \geq \sigma_2$. □

Two points are worth noting. First, if the user has no prior knowledge, the covering knowledge of each tuple t is the NULL rule and $v_{\mathcal{K}}(t) = 1$. In this case, $Usup(r) = sup(r)$, $Unexp(r) = 1$ and $Uconf(r) = conf(r)$. Therefore, the classic association rule mining [2] is the degenerate case of the unexpected rule mining where every satisfying tuple serves as a perfect violation. From this standpoint, it makes much sense to discriminate among the satisfying tuples in terms of their degree of confirming or violating the user knowledge and use those with a large violation to characterize which rules are unexpected. Second, this approach depends on the functions bm, hm, $v_{\mathcal{K}}(t)$ and a preference model, but does not depend on *how* they are specified, therefore, affords a great deal of flexibility for incorporating domain specific knowledge through particular specifications of these functions.

An algorithm for mining unexpected rules is given in Section 5.

4.2 Unexpected rule selection problem

Using unexpectedness in place of mere numerical thresholds like support/confidence already is a huge value-add. Still, the number of unexpected rules found can be too large for a human user. In the rule selection problem, we would like to select a *specified* (usually small) number of rules from a set of rules found so that they are as unexpected as possible. Simply ranking all rules by a criterion based on $Usup(r)$ and $Ustr(r)$ does *not* serve this purpose because several similar rules could be ranked high but may not provide new insights. Our approach is to model the unexpectedness of remaining rules in the presence of the rules already presented to the user. Consider the user knowledge $\mathcal{K}$ and a set of data rules $\mathcal{R}$ found. Suppose that we have selected $\mathcal{R}_i = \{r_1, \ldots, r_{i-1}\}$ from $\mathcal{R}$, where $i \geq 1$. We want to select the next most unexpected rule r_i from $\mathcal{R} - \mathcal{R}_i$, assuming that the user has seen $\mathcal{R}_i$. The implication of having the rules in $\mathcal{R}_i$ is stated in the following assumption.

The See-and-Know assumption: *After seeing $\mathcal{R}_i$, the user is aware of the rules in $\mathcal{R}_i$, therefore, is interested in only those rules that are unexpected with respect to $\mathcal{R}_i$.* Therefore, if r_i is "similar" to *any* rule in $\mathcal{R}_i$, r_i is not unexpected to the user. We can model this semantics by treating $\mathcal{R}_i$ as new knowledge rules with the *minimum violation preference* (see subsection 3.2): the covering knowledge of a tuple t is defined by the least violated (i.e., closest) matching rules in $\mathcal{R}_i$, denoted by $\mathcal{C}_t^i$. Let $\mathcal{C}_t$ denote the (usual) covering knowledge with respect to $\mathcal{K}$. Whenever r_i is unexpected with respect to either of $\mathcal{K}$ and $\mathcal{R}_i$, r_i is unexpected with respect to $< \mathcal{K}, \mathcal{R}_i >$ as a whole. Therefore, we define the covering knowledge of t with respect to $< \mathcal{K}, \mathcal{R}_i >$ as the d least violated rules in $\mathcal{C}_t \cup \mathcal{C}_t^i$, where d is the covering depth. This preference model is called *See-and-Know preference* for $< \mathcal{K}, \mathcal{R}_i >$.

DEFINITION 4.4 (RULE SELECTION PROBLEM). Let $Usup(r_i)$ and $Ustr(r_i)$ be defined wrt $< \mathcal{K}, \mathcal{R}_i >$. Given a database T, the user knowledge $\mathcal{K}$, a set of data rules $\mathcal{R}$, thresholds σ_1, an integer k, and a selection criterion $f(Usup, Ustr)$ (for a single rule), the *unexpected rule selection* is to find k data rules $r_1, \ldots, r_k$ from $\mathcal{R}$ such that, for $1 \leq i \leq k$,

1. $Usup(r_i) \geq \sigma_1$, and
2. $f(Usup(r_i), Ustr(r_i))$ is maximum in $\mathcal{R} - \mathcal{R}_i$, where $\mathcal{R}_i = \{r_1, \ldots, r_{i-1}\}$. □

In words, $r_1, \ldots, r_k$ are the list of rules from $\mathcal{R}$ such that, for $1 \leq i \leq k$, r_i has sufficient unexpectedness support and is most unexpected (as per the selection criterion f) with respect to the knowledge $\mathcal{K}$ plus the previously selected rules $r_1, \ldots, r_{i-1}$. The selection criterion $f(Usup, Ustr)$ provides a way for the user to weigh between $Usup$ and $Ustr$. For example, f can correspond to the ranking criterion using the key $(Usup, Ustr)$ (or $(Ustr, Usup)$) where $Usup$ and $Ustr$ are the primary and secondary keys, respectively, or in general, f can be some weighed sum or combination of $Usup$ and $Ustr$. f is a criterion for individual rules r_i, not for a collection of selected rules. The latter has the disadvantage of high complexity of finding an optimal collection of rules. We address the interaction among selected rules by dynamically updating the knowledge $< \mathcal{K}, \mathcal{R}_i >$ wrt which $Usup(r_i)$ and $Ustr(r_i)$ are defined.

An algorithm for selecting unexpected rules is given in Section 6.

Algorithm 1 UMINE: the unexpected rule mining

Input: a database T, user knowledge $\mathcal{K}$, thresholds σ_1 and σ_2

Output: data rules r such that $Usup(r) \geq \sigma_1$ and $Ustr(r) \geq \sigma_2$

```
1: /* The violation computing phase */
2: T' = ∅;
3: for all tuple t in T do
4:    compute v_K(t);
5:    if v_K(t) > 0 then
6:       add < t, v_K(t) > to T';
7:    end if;
8: end for;
9: /* The rule generating phase */
10: U_1 ← all rules r of length 1 with Usup(r) ≥ σ_1;
11: k = 1;
12: while U_k ≠ ∅ do
13:    generate candidate rules C_{k+1} using U_k;
14:    for all tuple < t, v_K(t) > in T' do
15:       for all rule r ∈ C_{k+1} such that t ∈ SAT(r) do
16:          Usup(r) = Usup(r) + v_K(t)/|T|;
17:       end for;
18:    end for;
19:    U_{k+1} ← all rules r in C_{k+1} with Usup(r) ≥ σ_1;
20:    k++;
21: end while;
22: /* The final phase */
23: for all tuple t in T do
24:    for all rule r ∈ ∪U_k such that t ∈ SAT(r) do
25:       sup(r)++;
26:       sup(b(r))++;
27:    end for;
28: end for;
29: output the rules r in ∪U_k such that Ustr(r) ≥ σ_2.
```

5. THE MINING ALGORITHM

We present the algorithm for mining unexpected rules, called UMINE, in Algorithm 1. There are three phases: the violation phase, the rule phase, and the final phase.

5.1 The violation phase

The violation phase computes and stores $v_{\mathcal{K}}(t)$ for all tuples t in the database T. $< t, v_{\mathcal{K}}(t) >$ denotes the tuple t with stored $v_{\mathcal{K}}(t)$. An important optimization is to prune all tuples t with $v_{\mathcal{K}}(t) = 0$ because such tuples have no contribution to $Usup(r)$. This is established below and implemented in lines 1-8 by storing only $< t, v_{\mathcal{K}}(t) >$ with $v_{\mathcal{K}}(t) > 0$ in T'.

THEOREM 5.1. Let $T' = \{< t, v_{\mathcal{K}}(t) > | \ t \in T \wedge v_{\mathcal{K}}(t) > 0\}$. Then for any data rule r, $Usup(r)$ is the same with respect to T as well as T'. □

In words, we can replace T with T' without affecting $Usup(r)$. Note that $v_{\mathcal{K}}(t) = 0$ if, for every covering knowledge rule R of t, $\overline{hm}(t, R) < \sigma'$ in Definition 3.2 (note that $bm(t, R) \geq \sigma$ for such R). In this case, t is consistent with the user knowledge. This pruning strategy makes much sense: if the data is largely consistent with or confirming the user knowledge, the confirming tuples have no value for mining unexpected rules.

5.2 The rule phase

The rule phase (lines 9-21) generates all rules with $Usup(r) \geq \sigma_1$ using T'. Simply enumerating all possible rules r is prohibitive. We are interested in a method that generates only promising rules. First, we show a property about $Usup(r)$ that forms the basis of this method.

THEOREM 5.2. *$Usup(r)$ is anti-monotone with respect to $b(r)$, in that $Usup(r)$ decreases as the body $b(r)$ grows. Further, this property is independent of the choice of the preference model and violation function $v_{\mathcal{K}}(t)$.*

Proof: Consider two data rules r and r' such that the body of r' has a superset of conditions compared to the body of r. A tuple t in $SAT(r')$ is also in $SAT(r)$. Since $v_{\mathcal{K}}(t)$ is non-negative, $\Sigma\{v_{\mathcal{K}}(t) \mid t \in SAT(r')\} \leq \Sigma\{v_{\mathcal{K}}(t) \mid t \in SAT(r)\}$. Thus, the anti-monotonicity follows. To complete the proof, note that $v_{\mathcal{K}}(t)$ on both sides of $\leq$ is the same for the same tuple t, independently of how the covering knowledge of t is specified and how $v_{\mathcal{K}}(t)$ is defined. □

Following Theorem 5.2, we need to examine a longer rule $x_1, \ldots, x_{k-1}, x_k, x_{k+1} \rightarrow c$ only if both shorter rules

$$x_1, \ldots, x_{k-1}x_k \rightarrow c \text{ and } x_1, \ldots, x_{k-1}, x_{k+1} \rightarrow c$$

pass the threshold σ_1 for $Usup(r)$, where x_k and x_{k+1} are domain values for different attributes. Thus, we can search for all rules r with $Usup(r) \geq \sigma_1$ as follows. In the first iteration, we generate all rules r of length 1 of the form $x_1 \rightarrow c$ such that $Usup(r) \geq \sigma_1$. These rules are stored in U_1. Subsequently, in the $(k+1)$th iteration $(k > 0)$, we generate all candidate rules r of length $k+1$, denoted by C_{k+1}, of the form $x_1, \ldots, x_{k-1}, x_k, x_{k+1} \rightarrow c$ using two rules in U_k of the above form. From Theorem 5.2, C_{k+1} contains all rules r with $k+1$ conditions in the body such that $Usup(r) \geq \sigma_1$. We compute $Usup(r)$ for all r in C_{k+1} in one scan of T' while accumulating $v_{\mathcal{K}}(t)$ over the tuples $< t, v_{\mathcal{K}}(t) > \in T'$ such that $t \in SAT(r)$. U_{k+1} contains those rules r in C_{k+1} with $Usup(r) \geq \sigma_1$. These steps are described in line 9-21 in Algorithm 1.

The above pruning strategy was first suggested by Agrawal et al. [2] for mining association rules of sufficient support *sup*. However, the notion of support does not take into account the "novelty" of a rule with respect to the user knowledge: rules of sufficient support may not be unexpected, and unexpected rules may not have sufficient support. We address this problem using the notion of unexpectedness support to capture the part of support that is responsible for violating the user knowledge. In this sense, our pruning is based on the "novelty" of rules.

5.3 The final phase

The final phase (line 22-29) computes $sup(r)$ and $sup(b(r))$ for rules r in $\cup U_k$ produced in the rule phase, and outputs the rules r with $Ustr(r) \geq \sigma_2$. This time, we scan T instead of T', and for each tuple t, we update $sup(r)$ and $sup(b(r))$ for all rules r such that $t \in SAT(r)$ and $t \in MAT(r)$, respectively. After computing $sup(r)$ and $sup(b(r))$ for all rules r in $\cup U_k$, we output those rules r such that $Ustr(r) \geq \sigma_2$.

A note on implementation. First, at lines 15 and 24, for every tuple t in T' or T, we need to find all rules r such that $t \in SAT(r)$. We can borrow the *subset function* [2] for these operations. Second, currently we compute $Usup(r)$ using the pruned database T' in one phase, compute $sup(r)$ and $sup(b(r))$ using the unpruned database T in another phase. Alternatively, we can compute all three using the unpruned T in a single phase. We distinguish these strategies by UMINE-Pruned and UMINE-Unpruned. Third, the above description of the rule generating phase is based on the Apriori-like breadth-first generation of rules [2]. What is essentially required is just the anti-monotonicity of the measure pushed, which in our case is $Usup$. Therefore, any implementation based on the anti-monotonicity, such as the depth-first generation, can be adopted as well.

Algorithm 2 USELECT: the unexpected rule selection algorithm

Input: a database T, a set of data rules $\mathcal{R}$, user knowledge $\mathcal{K}$, threshold σ_1, an integer k, and $f(Usup, Ustr)$
Output: as in Definition 4.4

```
1: /* initialization */
2: R_1 = ∅; i = 1; s = |T|;
3: for all tuple t in T do
4:    compute C_t = {R_1, ..., R_d} and replace t with <
      t, v(t, R_1), ..., v(t, R_d) > in T;
5:    for all rule r in R such that t ∈ SAT(r) do
6:       Usup(r) = Usup(r) + v_{<K,R_1>}(t)/s;
7:       sup(r) + +;
8:       sup(b(r)) + +;
9:    end for
10: end for
11: while i ≤ k and R ≠ ∅ do
12:    /* select the next most unexpected rule */
13:    select the rule r_i from R such that Usup(r_i) ≥ σ_1 and
       f(Usup(r_i), Ustr(r_i)) is maximum;
14:    /* update the information */
15:    for all tuple < t, v(t, R_1), ..., v(t, R_d) > in T such
       that t ∈ MAT(r_i) do
16:       update the tuple with v(t, r_i);
17:       compute Δv_{<K,R_i>}(t) due to the update;
18:       for all r in R − {r_i} such that t ∈ SAT(r) do
19:          Usup(r) = Usup(r) + Δv_{<K,R_i>}(t)/s;
20:          if Usup(r) < σ_1 then
21:             delete r from R
22:          end if;
23:       end for;
24:       if v_{<K,R_i>}(t) = 0 then
25:          delete t from T
26:       end if;
27:    end for;
28:    output r_i;
29:    R_i = R_i ∪ {r_i};
30:    R = R − {r_i};
31: end while;
32: output the rules in R_i in the order added;
```

6. THE SELECTION ALGORITHM

We present the algorithm for selecting a specified number of unexpected rules, called USELECT, in Algorithm 2. First, the initialization (line 1-9) computes $Usup(r)$, $sup(r)$, $sup(b(r))$ for all rules r in $\mathcal{R}$ in one database scan. It also computes the violation bag $V_t = \{v(t, R_1), \cdots, v(t, R_d)\}$ and stores it with each tuple t, denoted by $< t, V_t >$, where $\mathcal{C}_t = \{R_1, \cdots, R_d\}$ is the covering knowledge of t with re-

spect to $< \mathcal{K}, \mathcal{R}_i >$. d is usually a small number like 1 or 2 or 3. Note that $v_{<\mathcal{K},\mathcal{R}_i>}(t) = agg(V_t)$. The reason for keeping the violation bag V_t rather than $v_{<\mathcal{K},\mathcal{R}_i>}(t)$ is for updating the covering knowledge $\mathcal{C}_t$ when rules are added to $\mathcal{R}_i$. See below.

In the iterative part (lines 11-31), each iteration selects the most unexpected rule r_i from remaining rules $\mathcal{R}$ and adds it to $\mathcal{R}_i$, until either k rules are selected or there is no rule to select. Also, we update the violation bag V_t for affected tuples t to account for the new rule r_i added to $\mathcal{R}_i$. The See-and-Know preference implies that if r_i matches a tuple t and $v(t, r_i) < max(V_t)$, $v(t, r_i)$ should replace one occurrence of $max(V_t)$ in V_t. This is done at line 16 with the detail omitted. The induced (negative) change $\Delta v_{<\mathcal{K},\mathcal{R}_i>}(t)$ is propagated to $Usup(r)$ for the remaining rules r such that $t \in SAT(r)$ at lines 17-19. At lines 20-21 we prune any remaining rule r with $Usup(r) < \sigma_1$, and at lines 24-25 we prune any tuple t with $v_{<\mathcal{K},\mathcal{R}_i>}(t) = 0$ from the database. These pruning optimizations follow from the following theorem.

THEOREM 6.1. (1) $v_{<\mathcal{K},\mathcal{R}_i>}(t)$ is anti-monotone with respect to $\mathcal{R}_i$, (2) $Usup(r, < \mathcal{K}, \mathcal{R}_i >)$ is anti-monotone with respect to $\mathcal{R}_i$.

Proof: By adding a rule r_i to $\mathcal{R}_i$, the See-and-Know preference prefers r_i if $v(t, r_i)$ is smaller than $max(V_t)$. Therefore, $V_t' \leq V_t$, where V_t is the violation bag before adding r_i to $\mathcal{R}_i$ and V_t' is the violation bag after adding r_i to $\mathcal{R}_i$. So we have $agg(V_t') \leq agg(V_t)$ for well-behaved agg. This shows that $v_{<\mathcal{K},\mathcal{R}_i>}(t)$ is anti-monotone with respect to $\mathcal{R}_i$. (2) follows from this anti-monotonicity and the definition of $Usup(r, < \mathcal{K}, \mathcal{R}_i >)$. □

Two pruning strategies follow from Theorem 6.1. First, if $v_{<\mathcal{K},\mathcal{R}_i>}(t) = 0$ for some iteration i, then $v_{<\mathcal{K},\mathcal{R}_j>}(t) = 0$ for all subsequent iterations j because $\mathcal{R}_j$ is a superset of $\mathcal{R}_i$. Therefore, such tuples t can be pruned in subsequent iterations without affecting the result. This is done at lines 24-25. Second, for any remaining rule r, $Usup(r, < \mathcal{K}, \mathcal{R}_j >) \leq Usup(r, < \mathcal{K}, \mathcal{R}_i >)$ for $j \geq i$ because $\mathcal{R}_j$ is a superset of $\mathcal{R}_i$. Therefore, if $Usup(r, < \mathcal{K}, \mathcal{R}_i >) < \sigma_1$, r can be pruned immediately. This is done at lines 20-21.

A note on implementation. The search of all tuples that match a given rule at line 15 requires the *superset function* instead of the subset function. This can be done by reversing the roles of tuples and rules in the subset function implementation discussed earlier, that is, storing tuples t, instead of rules, in a hash-tree and using the given rule to descend the hash-tree. The subset/superset function implementations and pruning rules and tuples at lines 20-21 and 24-25 make the algorithm much more efficient than its worst-case complexity. Typically, the selection algorithm is applied to the unexpected rules $\mathcal{R}$ found by the algorithm in Section 5. Hence, the number of rules in $\mathcal{R}$ is much smaller than the number of rules found without pushing user knowledge.

7. EVALUATION

In this section, we evaluated UMINE and USELECT with two objectives. The first is to evaluate the effectiveness of dealing with the computational bottleneck by pushing the user knowledge. The second objective is to evaluate the effectiveness of finding interesting rules in terms of unexpectedness.

7.1 The KDD-CUP-98 dataset

The experiments were conducted on the well known KDD-CUP-98 dataset [1] about the result of the 1997 Paralyzed Veterans of America fund raising mailing campaign. We chose this dataset mainly for the consideration of obtaining the user knowledge based on common sense in this application. The dataset contains 191,778 tuples over 482 attributes about donors' demographic information and donation history. We picked $NK97$ as the target attribute, which represents the donation amount (in dollars) in response to the 1997 mailing campaign. We discretized the dollar amount of $NK97$ into five donation scales: $c_0 = 0, c_1 = (0, 5], c_2 = (5, 10], c_3 = (10, 20], c_4 = (20, \infty)$ (note that most donations are small). We picked up the following 23 non-target attributes, as they seem to be related to the target attribute and their meanings are easier to understand than most of the other attributes.

A_1 NOEXCH: whether it can be exchanged for list rental
A_2 RECINHSE: whether donor has given to PVA's in house program
A_3 RECP3: whether donor has given to PVA's P3 program
A_4 RECPGVG: whether it is a planned giving record
A_5 RECSWEEP: whether it is a Sweepstakes record

MDMAUD: RFA status for donors who have given a $100+ gift at any time in their giving history:
A_6 Recency of Giving;
A_7 Frequency of Giving;
A_8 Amount of Giving;
A_9 Blank/ meaningless/filler

DOMAIN:
A_{10} Urbanicity level of donor's neighborhood;
A_{11} Socio-Economic status of neighborhood

A_{12} AGE: Overlay Age
A_{13} HOMEOWNR: Home Owner Flag
A_{14} NUMCHLD: Number of Children
A_{15} INCOME: Household Income
A_{16} GENDER: Gender
A_{17} MAJOR: Major ($$) Donor Flag
A_{18} WEALTH2: Wealth Rating
A_{19} Percent Households with Income, replaced with the majority income as obtained from IC6 to IC14
A_{20} Percent Families with Income, replaced with the majority income as obtained from IC15 to IC23

ADATE_2: RFA status as of 97NK promotion date:
A_{21} Recency based on date of the last gift;
A_{22} Frequency based on the period of recency;
A_{23} Amount of the last gift

We need to have some user knowledge about how donors would respond (in donation) to the 1997 mailing campaign in terms of non-target attributes. This information is not available from the dataset, nor is it easily available elsewhere. Our approach is to "simulate" the user knowledge using some rules found in the data, which is reasonable because in real applications often the user knowledge does come from the analysis of the data. In particular, we simulated the following user knowledge.

The user knowledge: *People tend to remain unchanged in donation behaviors, that is, those who were active/inactive*

in the past are likely to remain so this time (i.e., for NK97). This knowledge is represented by the 4 rules R_1 to R_4 below that were extracted from the data. We have included the support and confidence of these rules, which shows their strong existence in the data.

- $R_1 : A_{17} = Not\_major\_donor \rightarrow NK97 = Non\_donor$: a non-major donor is unlikely to donate this time. The user value Non_donor corresponds to c_0. (sup=78.09% and conf=86.90%)
- $R_2 : A_{21} = Lapsing\_donor \rightarrow NK97 = Non\_donor$: a lapsing donor (i.e., "a donor who made their last donation between 13-24 months ago") is unlikely to donate this time. (sup=73.25% and conf=97.35%)
- $R_3 : A_{21} = Active\_donor \rightarrow NK97 = Donor$: an active donor (i.e., "a donor who made their first donation more than 12 months ago and has made a donation in the last 12 months") is likely to donate this time. The user value $Donor$ corresponds to any one of the non-zero donation scales c_1, c_2, c_3, c_4. In implementation, we replicated R_3 into four knowledge rules with $Donor$ instantiated to each of c_1, c_2, c_3, c_4, and they are called R_3 collectively. (sup=9.52% and conf=67.67%)
- $R_4 : A_{21} = Star\_donor \rightarrow NK97 = Donor$: a star donor (i.e, "a donor who has given to 3 consecutive card mailings") is likely to donate this time. We replicated R_4 into four knowledge rules with $Donor$ instantiated to each of c_1, c_2, c_3, c_4. (sup=3.07% and conf=68.58%)

For non-target attributes, we use the binary match degree measure $m(v, v')$: $m(v, v') = 1$ if $v = v'$, or $m(v, v') = 0$ otherwise. For the target attribute, we use the distance based measure $m(c_i, c_j) = 1 - |i - j|/4$, i.e., the further away the two scales c_i and c_j, the less the match degree between them. We use the average avg for the aggregate in Definition 3.1, and consider the covering depth of 1.

7.2 Efficiency of the unexpectedness mining

UMINE was implemented based on an adoption of the depth-first based BUC for mining frequent itemsets [5]. We compared UMINE with the method of not using user knowledge, denoted by UMINE(NULL), which we implemented as UMINE with only the default rule as the user knowledge. UMINE-Pruned denotes UMINE exactly as described in Algorithm 1, and UMINE-Unpruned denotes UMINE that merges the rule phase and the final phase into one phase but scans the unpruned database.

Figure 1(a) and (b) shows the execution time versus the unexpectedness support threshold, for the minimum violation preference and the maximum confidence preference, respectively. After the unexpectedness support threshold is reduced to 10%, we have to abort UMINE(NULL) because the execution time is too long. UMINE-Pruned and UMINE-Unpruned, however, can go to a much lower threshold thanks to the focus on unexpectedness support. This indicates that, without pushing the user knowledge, the threshold has to be set very high, in which case many unexpected rules will not pass the threshold, therefore, will not be found by the post-analysis approach. Figure 1(a') and (b') shows that the number of rules generated, which measures the search space explored, is much smaller after

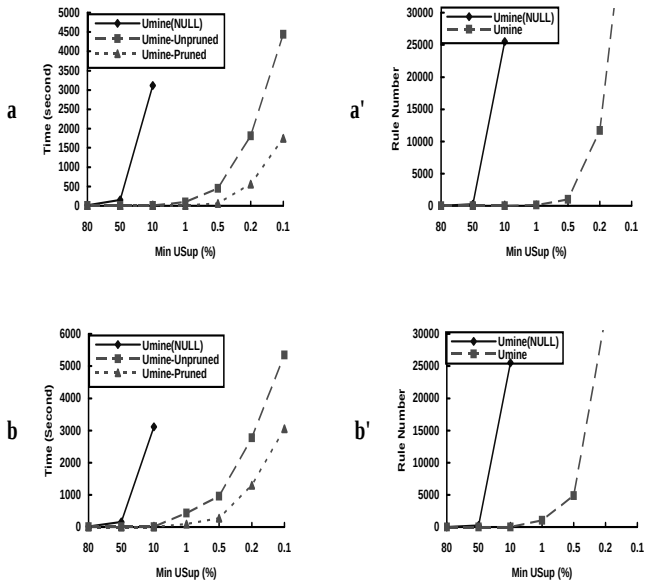

Figure 1: Efficiency

pushing the user knowledge. This suggests that, as far as the unexpected rule mining is concerned, many useless rules are generated if the user knowledge is not pushed. UMINE-Pruned is more efficient than UMINE-Unpruned, due to the pre-pruning of up to 85% of the database. This experiment clearly demonstrated the effectiveness of the proposed methods in focusing search effort.

In the rest of the experiment, the minimum violation preference is used as the preference model, and the covering depth is 1.

7.3 Effectiveness of the unexpectedness mining

Instead of listing all rules found, which are too many, we examine a few top rules to convey the idea about the effectiveness. We consider each rule independently. The interaction between rules will be considered in Subsection 7.4. We applied UMINE at the minimum unexpectedness support of 0.2% and found 11,722 rules. The 20 rules of highest $Unexp$ are presented below. For each rule r presented, we also present the *violation distribution* $R_i = (x, y)$, where the knowledge rule R_i covers x tuples satisfying r, with y being the total violation by these tuples.

1. $A_{21} = Lapsing\_donor \rightarrow NK97 = c_4$ (Usup=0.38%; Unexp=100%). $R_1 = (724, 724), R_2 = (11, 11)$.
2. $A_1 = Can\_be\_exchanged \rightarrow NK97 = c_4$ (Usup=0.38%; Unexp=100%). $R_1 = (722, 722), R_2 = (11, 11)$.
3. $A_4 = Not\_planned\_giving \rightarrow NK97 = c_4$ (Usup = 0.38%; Unexp =100%). $R_1 = (723, 723), R_2 = (10, 10)$.
4. $A_{17} = Not\_major\_donor \rightarrow NK97 = c_4$ (Usup=0.38%; Unexp=100%). $R_1 = (724, 724)$.
5. $A_5 = Not\_sweepstakes \rightarrow NK97 = c_4$ (Usup = 0.38%; Unexp=100%). $R_1 = (713, 713), R_2 = (11, 11)$.
6. $A_9 = Not\_major\_donor \rightarrow NK97 = c_4$ (Usup=0.38%; Unexp=100%) $R_1 = (724, 724)$.

7. $A_3 = Not\_P3 \rightarrow NK97 = c_4$ (Usup=0.37%; Unexp=100%). $R_1 = (697, 697), R_2 = (9, 9)$.

8. $A_2 = Not\_in\_house \rightarrow NK97 = c_4$ (Usup = 0.35%; Unexp = 100%). $R_1 = (673, 673)$.

9. $A_{13} = Home\_owner \rightarrow NK97 = c_4$ (Usup=0.26%; Unexp=100%). $R_1 = (485, 485), R_2 = (8, 8)$.

10. $A_{22} = One\_gift\_in\_theperiod\_of\_recency \rightarrow NK97 = c_4$ (Usup=0.24%; Unexp=100%). $R_1 = (460, 460), R_2 = (8, 8)$.

11. $A_{23} = \$25.00\_and\_above \rightarrow NK97 = c_4$ (Usup = 0.21%; Unexp = 1) $R1 = (396, 396)$, $R2 = (11, 11)$.

12. $A_1 = Can\_be\_exchanged, A_{21} = Lapsing\_donor \rightarrow NK97 = c_4$ (Usup= 0.38%; Unexp=100%). $R1 = (722, 722)$, $R2 = (11, 11)$.

13. $A_4 = Not\_planned\_giving, A_{21} = Lapsing\_donor \rightarrow NK97 = c_4$ (Usup= 0.38%; Unexp=100%). $R_1 = (723, 723)$, $R2 = (10, 10)$.

14. $A_1 = Can\_be\_exchanged, A_4 = Not\_planned\_giving \rightarrow NK97 = c_4$ (Usup= 0.38%; Unexp=100%). $R_1 = (721, 721)$, $R_2 = (10, 10)$.

15. $A_9 = Not\_major\_donor, A_{21} = Lapsing\_donor \rightarrow NK97 = c_4$ (Usup= 0.38%; Unexp=100%). $R_1 = (724, 724)$.

16. $A_9 = Not\_major\_donor, A_{17} = Not\_major\_donor \rightarrow NK97 = c_4$ (Usup= 0.38%; Unexp=100%). $R_1 = (724, 724)$.

17. $A_{17} = Not\_major\_donor, A_{21} = Lapsing\_donor \rightarrow NK97 = c_4$ (Usup= 0.38%; Unexp=100%). $R_1 = (724, 724)$.

18. $A_5 = Not\_sweepstakes, A_{21} = Lapsing\_donor \rightarrow NK97 = c_4$ (Usup= 0.38%; Unexp=100%). $R_1 = (713, 713)$, $R_2 = (11, 11)$.

19. $A_4 = Not\_planned\_giving, A_9 = Not\_major\_donor \rightarrow NK97 = c_4$ (Usup= 0.38%; Unexp=100%). $R1 = (723, 723)$.

20. $A_4 = Not\_planned\_giving, A_{17} = Not\_major\_donor \rightarrow NK97 = c_4$ (Usup= 0.38%; Unexp=100%). $R_1 = (723, 723)$.

Generally speaking, these rules are found because their bodies are either the same as or correlated with the bodies of R_1 or R_2, but their heads say very different things from the heads of R_1 or R_2. For example, the body of rule No 3, $A_4 = Not\_planned\_giving$, tends to be correlated with a non-major donor (i.e., $A_{17} = Not\_major\_donor$), and that is why the minimum violation preference (which acts on behalf of the user) chose R_1 to cover 723 tuples satisfying this rule. Rule No 3 is unexpected to the extent that it summarizes many tuples that violate the user preferred knowledge on them. This example also illustrates that our method finds not only "directly unexpected rules" that use attributes used by the user, such as rule No 1, but also "indirectly unexpected rules" that use attributes correlated with those used by the user, such as rule No 3. Since correlation does not require syntax similarity, "indirectly unexpected rules" cannot be found by the syntax based [8, 9] or the logic based [12]. If the user knows these correlations, the unexpectedness of "indirectly unexpected rules" is immediate to the user. Otherwise, "indirectly unexpected rules" together with the violated knowledge rules (i.e., R_1 here) provides new information to the user.

If we go further down the list, some rules violate two or more knowledge rules. Here is an example:

- $A_4 = Not\_planned\_giving, A_{17} = Not\_major\_donor \rightarrow NK97 = c_1$ (Usup = 1.05%, Unexp = 10%). $R_1 = (7510, 1877.5)$, $R_2 = (555, 138.75)$, $R_3 = (9633, 0)$, $R_4 = (1695, 0)$.

The subpopulation that satisfies this rule violates two knowledge rules, R_1 and R_2. This is different from previous approaches where the unexpectedness is always measured with respect to a single knowledge rule.

7.4 Effectiveness of the selection problem

In the above experiment, many rules were found because of some conditions correlated with the bodies of knowledge rules. Such rules are related in the sense of capturing the same or similar violating subpopulation, i.e., $Usup(r)$ coming from the same or similar violating subpopulation. We can treat all such rules as one cluster and present only one representative of the cluster to the user. The other rules in a cluster can still be computed, but will not be presented to the user, unless the user requests them. This strategy is implemented in the selection algorithm USELECT by pruning all satisfying tuples once a data rule is selected. This pruning is also a consequent optimization of the See-and-Know preference dealing with the unexpectedness dynamics, as shown in Section 6.

To evaluate the effectiveness of the selection algorithm, we applied USELECT to the 11,722 rules found in Subsection 7.3 to select 5 most unexpected rules, with $f(Usup, Ustr) = Unexp$ as the selection criterion. The following 5 rules[1] are found in that order (recall that the minimum unexpectedness support is 0.2%):

- R_5 : $A_{21} = Lapsing\_donor \rightarrow NK97 = c_4$ (Usup = 0.38%; Unexp = 100%). $R_1 = (724, 724), R_2 = (11, 11)$.

- $R_6 : A_{21} = First\_time\_donor \rightarrow NK97 = c_1$ (Usup = 0.50%; Unexp=26%). $R_1 = (3641, 910.25), NULL = (39, 39)$.

- R_7 : $A_{21} = New\_donor \rightarrow NK97 = c_1$ (Usup = 0.35%; Unexp=25%). $R_1 = (2589, 647.25), NULL = (16, 16)$.

- $R_8 : A_{21} = Star\_donor \rightarrow NK97 = c_0$ (Usup = 0.25%; Unexp=18%). $R_4 = (1900, 475)$.

- $R_9 : A_{21} = Active\_donor$, $A_{22} = One\_gift\_in\_the\_period\_of\_recency \rightarrow NK97 = c_0$ (Usup = 0.35%; Unexp=13%). $R_3 = (2656, 664)$.

As discussed in Section 6, each time a data rule is selected, it is made a new knowledge denoted R_i $(5 \leq i \leq 9)$ in the subsequent selection to prevent similar rules from being selected again. As we can see, the above 5 rules selected this

[1]Despite their syntactic similarity, rules R_6 and R_7 summarize disjoint subpopulations in the data!

way provide more insights than simply reporting the top 5 unexpected rules in Subsection 7.3. For example, from R_8 and R_9, the user learns unexpected rules about star and active donors not donating this time. Many of the top 20 rules in Subsection 7.3 are not in this list, therefore, not presented to the user (who only wants to see 5 rules in our example!). However, as mentioned above, each presented rule R_i $(5 \leq i \leq 9)$ in fact represents a cluster of rules whose $Usup$ mainly comes from the satisfying subpopulation of R_i and which are not presented because of the pruning of this subpopulation. USELECT can be modified without difficulty to compute this cluster for each selected rule. Therefore, the user still has the choice of knowing the rules not presented if she wishes to do so.

In summary, our evaluation indicated that the proposed methods show a promise in addressing the three issues raised in the introduction. Ultimately, whether a rule is interesting is up to the end user to judge. This is because often the user knowledge specified is incomplete and certain knowledge is not easy to model. It would be unrealistic to expect every rule found to be a surprise to the user. A more realistic goal of our work is to take into account the user knowledge that can be modeled so that a large number of obvious rules can be pruned. However, the user still needs to examine the surviving (but much fewer) rules to identify interesting ones.

8. CONCLUSION

The user knowledge plays a crucial role in determining what is interesting to the user. However, most data mining algorithms do not consider such knowledge and merely let the user choose some parameters and thresholds required by the algorithms. The consequences are that: (1) a large number of rules are found, while a human user can only comprehend a small number; (2) many uninterested rules are found, but many interesting rules are not. These consequences are major obstacles to today's data mining applications. In this paper, we addressed these issues by examining the fundamental question of how the user knowledge should be represented and modeled in mining unexpected rules. An additional novel contribution is that we recognized the human user as a dynamic entity in applying her knowledge to have a say in what is most unexpected to her. Modeling these dynamics can better help capture the intuition that what is unexpected can change as the user starts seeing discovered rules. Another contribution is a general framework for pushing the user knowledge into the search process. Our experiments support our claim that the proposed approach is not only efficient in mining rules, but it is also effective in terms of the quality (i.e., unexpectedness) of the rules produced.

9. REFERENCES

[1] Kdd-cup-98 dataset. http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html.

[2] R. Agrawal and R. Srikant. Fast algorithm for mining association rules. In *VLDB*, pages 487–499, September 1994.

[3] R. Bayardo and R. Agrawal. Mining the most interesting rules. In *SIGKDD*. SIGKDD, 1999.

[4] R. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. In *ICDE*. IEEE, 1999.

[5] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD*, pages 359–370, 1999.

[6] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD*, pages 265–276, 1997.

[7] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonene, and A. Verkamo. Finding interesting rules from large sets of discovered association rules. In *CIKM 94*, 1994.

[8] B. Liu and W. Hsu. Post-analysis of learned rules. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 828–834. AAAI, Aug 1996.

[9] B. Liu, W. Hsu, and S. Chen. Using general impressions to analyze discovered classification rules. In *KDD*, 1997.

[10] R. Ng, L. V. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *SIGMOD*, pages 13–24, 1998.

[11] B. Padmanabhan and A. Tuzhilin. A belief-drievn method for discovering unexpected patterns. In *KDD-98*. SIGKDD, 1998.

[12] B. Padmanabhan and A. Tuzhilin. Unexpectedness as a measure of interestingness in knowledge discovery. In *KDD-98*. SIGKDD, 1998.

[13] B. Padmanabhan and A. Tuzhilin. Small is beautiful: Discovering the minimal set of unexpected patterns. In *SIGKDD*, pages 54–63. SIGKDD, 2000.

[14] G. Piatesky-Shapiro. Discovery, analysis and presentation of strong rules. In *Knowledge Discovery in Databases*, pages 229–248. MIT Press, 1991.

[15] G. Piatesky-Shapiro and C. Matheus. The interestingness of deviations. In *KDD-94*. SIGKDD, 1994.

[16] S. Sahar. Interestingnes via what is not interesting. In *SIGKDD*, pages 332–336. SIGKDD, 1999.

[17] A. Silberschatz and A. Tuzhilin. On subjective measures of interestingnessin in knowledge discovery. In *KDD*, pages 275–281. KDD, 1995.

[18] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. In *IEEE Transactions on Knowledge and Data Engineering*, 1996.

[19] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *KDD*, pages 67–73. SIGKDD, 1997.

[20] P.-N. Tan and V. Kumar. Interestingness measures for association patterns: A perspectives. In *KDD-2000 Workshop on Post-Processing in Machine Learning and Data Mining*, 2000.

[21] K. Wang, Y. He, and J. Han. Pushing support constraints into frequent itemset mining. In *VLDB*, 2000.

[22] K. Wang, Y. Jiang, J. Yu, G. Dong, and J. Han. Pushing aggregate constraints by divide-and-approximate. In *ICDE*. IEEE, 2003.

[23] M. Zaki. Generating non-redundant association rules. In *SIGKDD 2000*. SIGKDD, 2000.