Mining Itemsets in the Presence of Missing Values

Toon Calders
Eindhoven Technical University
t.calders@tue.nl

Bart Goethals
University of Antwerp
bart.goethals@ua.ac.be

Michael Mampaey
University of Antwerp
michael.mampaey@ua.ac.be

ABSTRACT

Missing values make up an important and unavoidable problem in data management and analysis. In the context of association rule and frequent itemset mining, however, this issue never received much attention. Nevertheless, the well known measures of support and confidence are misleading when missing values occur in the data, and more suitable definitions typically don't have the crucial monotonicity property of support. In this paper, we overcome this problem and provide an efficient algorithm, XMiner, for mining association rules and frequent itemsets in databases with missing values. XMiner is empirically evaluated, showing a clear gain over a straightforward baseline-algorithm.

1. INTRODUCTION

Association Rule Mining (ARM) [1] is the problem of finding rules $X \Rightarrow Y$ in a database $\mathcal{D}$ of sets, such that when a set in $\mathcal{D}$ contains X, the probability is high that Y is a subset as well. Traditionally two measures for association rules are defined, *support* and *confidence*, and mining algorithms are typically divided into two parts: first find all frequent sets, and then from these, generate all confident rules. The subset lattice spanning the exponentially large search space of itemsets is traversed using the monotonicity of the support measure. Doing so allows large sublattices with an infrequent set as root to be pruned completely.

In the original definitions of association rule mining, missing data are not considered. In practice, however, missing data is a very prominent and non-trivial problem that needs to be handled adequately. The usual approach is to either impute the missing data by using, *e.g.*, the mean of the attribute values that are present, or to simply remove tuples with nulls. These techniques, however, can severely distort the distribution of the data, which can result in misleading or meaningless output, *e.g.* the loss of good or fabrication of bad rules.

The approach proposed in this paper is based on previous work of *Ragel and Crémilleux* [5] in which the notions of support and confidence are redefined, as to deal more adequately with missing data. Support is no longer measured against the *complete* database, but instead, the support of a set is defined w.r.t. the *subset* of the database having no missing values in the attributes of the itemset. Furthermore, they introduce the notion of a minimal *representativity* of a set, as to avoid sets in the output that are frequent solely because there are very few tuples that do not have missing values in the attributes of the set. The new support measure, though adequate, is no longer monotone with the size of the itemset. Hence, most well known algorithms can no longer be applied when these definitions are used. In their paper, however, *Ragel and Crémilleux* only focus on the properties of these measures, showing that they are very suitable to recover the original rules after randomly introducing missing values. They do not focus on efficient algorithms.

For a probabilistic approach to missingness, we refer to *Kryszkiewicz* [4], where pessimistic, optimistic and expected estimations of support and confidence are defined, and are compared to the measures of Ragel and Crémilleux.

The contributions of this paper can be summarized as follows. First, in order to deal with the non-monotonicity of the support measure, we introduce and study the theoretical notion of *extensibility* of itemsets. Extensibility subsumes frequency but does have the monotone property. Secondly, based on the properties of extensibility that we show, an algorithm, XMiner, is given. Finally, through experimentation on real-life datasets with a prototype of the algorithm and a straightforward baseline algorithm, it is shown that XMiner (for extensibility miner) has a significant performance improvement w.r.t. the baseline approach.

2. MISSING VALUES

This section describes the problem with missing values for association rules, gives definitions of the measures we will use, and defines the extensibility property for itemsets.

2.1 The problem

Typically, the support of an itemset is measured against the complete database. Hence, if a certain attribute-value occurs in 25% of the tuples, then it's support is said to be 25%, even if a value of the attribute is missing in 50% of the tuples. Similarly, the confidence of an association rule is measured as the number of tuples satisfying all attribute-value combinations present in the rule, divided by the number of tuples satisfying only the antecedent, no matter how many values of the consequent were actually missing in those tuples. For example, consider the two small databases in

t	A	B	C
1	a_1	b_2	c_1
2	a_1	b_2	c_2
3	a_2	b_2	c_2
4	a_1	b_1	c_1

t	A	B	C
1	a_1	b_2	c_1
2	**?**	b_2	c_2
3	a_2	b_2	**?**
4	a_1	**?**	c_1

Figure 1: Identical databases, without and with missing values

figure 1, of which the second one is simply the first one, but with some values missing. Even in this small example with only a few nulls, the effects are visible. Using the classic definitions of support and confidence, the rule $(A = a_1) \Rightarrow (B = b_2)$ has a support of $\frac{2}{4} = 50\%$, and confidence $\frac{2}{3} = 66\%$ in DB1. The rule $(A = a_1) \Rightarrow (C = c_1)$ has exactly the same support and confidence. Due to the missing values in DB2, the support of rule 1 has dropped to $\frac{1}{4} = 25\%$, while that of rule 2 has remained the same. The confidence of rule 1 has also dropped, to $\frac{1}{2} = 50\%$, while the confidence of rule 2 has become $\frac{2}{2} = 100\%$. In general support can drop, and confidence can either rise or drop. Depending on *minsup* and *minconf*, and the missingness itself, this can lead to the loss of rules that are frequent and confident, as well as the creation of rules that aren't in the original database.

2.2 Definitions

Firstly, some notation. We assume that we are working in a relational database $\mathcal{D}$, since it is in these databases (unlike transactional ones), that the problem of missing values occurs more naturally. Each tuple t has a transaction/tuple identifier *tid*. Attributes $A, B, \ldots$ and their values $a, b, \ldots$ will be in upper- and lowercase respectively. An item is an attribute-value pair, written as $(A = a_i)$. The missingness- and attribute item are denoted $(A = ?)$ and $(A = *)$. They are used when a value is resp. either missing or observed for the attribute A. (Note that although these notations are similar to those of a proper item, we cannot just treat them as such.) Itemsets, denoted $X, Y, \ldots$ are sets of these pairs. We will assume that itemsets are consistent, in that no two items of a same attribute - *e.g.* $(A = a_1)$ and $(A = a_2)$ - occur in an itemset together. The only trivial exception to this is $(B = b_1)$ and $(B = *)$, but since the former implies the latter, it may be omitted. By *count* we mean the absolute number of transactions in $\mathcal{D}$ that support an itemset: $\text{count}(X) := |\{t | X \subseteq t.\textit{items}\}|$. Finally, the attribute set of an itemset X is defined as: $X^* := \{(A = *) | (A = a_i) \in X\}$. For example, $\{(A = a_1), (C = *), (D = ?)\}^* = \{(A = *)\}$.

As already argued by Ragel and Crémilleux [5], it is more sensible to use support and confidence measures that are defined relative to the non-missingness of the items in the itemset or association rule. Specifically, they will be measured against the sample of the database that has no missing values for the itemsets in question. However, if this sample becomes very small but nevertheless yields strong rules, its influence on the results should be restricted, since this output is not representative.

Support. We will use the redefined notion of the support of an itemset X as the number of tuples supporting X, relative to the number of tuples that have no missing values *for the attributes of* X. The nulls in the database may or may not hide complete transactions that support X. Note that the tuples we do count, still might have missing values for other attributes. Statistically this is expressed as $Pr(X|X^*)$, hence the new support measure becomes:

Definition 1. $\textit{supp}(X) := \dfrac{\textit{count}(X)}{\textit{count}(X^*)}$

Unfortunately this improved measure is not monotone. This means that in the exponential search space which forms a lattice, the frequent and infrequent sets can no longer be separated by a simple border. This leads to 'infrequent enclaves' among frequent sets and vice versa. At first sight it seems we have to traverse the lattice entirely.

Confidence. Although this paper is mainly concerned with itemset mining, for the sake of completeness the definition of the confidence measure is also given here. Similar to support, the confidence of a rule $X \Rightarrow Y$ is also redefined, as the number of tuples that support them, relative to the number of tuples that support the antecedent *and* have no missing values for the attributes of the consequent (and trivially the antecedent). Statistically this is expressed as $Pr(Y|X \cup Y^*)$.

Definition 2. $\textit{conf}(X \Rightarrow Y) := \dfrac{\textit{count}(X \cup Y)}{\textit{count}(X \cup Y^*)}$

Note that with this new definition of confidence, it is no longer easy to construct the confident rules from the frequent sets. The confidence of a rule $X \Rightarrow Y$ is not equal to $\text{supp}(X \cup Y)/\text{supp}(X)$. In fact, $Pr(Y|X \cup Y^*)$ only equals $Pr(Y|X)$ if X and Y are independent.

Representativity. This is a completely new measure. It is needed to restrict the influence of itemsets that are not observed a lot (*i.e.* all tuples in $\mathcal{D}$ have missing values for some of the attributes) on confidence and support. In other words the sample of $\mathcal{D}$ that has no missing values for the attribute set of X must be a representative sample.

Definition 3. $\textit{rep}(X) := \dfrac{\textit{count}(X^*)}{|\mathcal{D}|}$

Whether this definition is absolute or relative to the number of tuples in the database is not that important, just like with the traditional definition of support. Below we will use representativity either way for the sake of brief notation. It will always be clear from the context which is meant.

Extensibility. Finally, we define the notion of *extensibility.*

Definition 4. *An itemset X is called extensible, if it has a frequent and representative superset,* i.e.

$$\exists Y : \begin{cases} \textit{supp}(X \cup Y) \geq \textit{minsup} \\ \textit{rep}(X \cup Y) \geq \textit{minrep} \end{cases}$$

It is clear that extensibility is monotone (a superset of an inextensible itemset is never extensible). Note that Y may be empty, so all *representative frequent itemsets are extensible.*

Remarks. All of these definitions are backwards compatible with the traditional ones, when missingness is absent. In that case, representativity is 100%, and extensibility is the same as frequency.

In its absolute form, representativity is the denominator of support, so checking it first is already half of the work of computing support (plus, it is monotone). The relative form is also similar to traditional support, as well as the new version (but on the level of attributes): $\text{rep}(X) = \text{count}(X^*)/\text{count}((X^*)^*)$, since $(X^*)^* = \emptyset$. This implies a hierarchy among the items, where for an attribute A, $A > (A = ?)$ and $A > (A = *) > (A = a)$. However, using a generalized itemset mining algorithm (as in [6]) is not appropriate, because more general itemsets do not necessarily have higher support.

2.3 Properties

The redefined support measure is no longer monotone, but since extensibility subsumes frequency (*i.e.* inextensible sets are infrequent or unrepresentative), we will mine for these sets instead. For this we need a numerical test to check the extensibility of an itemset.

THEOREM 1. *For an itemset X, define the set S_X as $S_X := \{Z \supseteq X | Z \text{ is representative and frequent}\}$,* i.e. *all frequent and representative supersets of X. Let among their local maxima, $m(X)$ be the lowest representativity, $m(X) := \min\{rep(Z)|Z \in S_X\}$, or $m(X) := +\infty$ if $S_X = \emptyset$. Then*

$$X \text{ is extensible} \Leftrightarrow \frac{count(X)}{m(X)} \geq minsup$$

PROOF. If X is extensible, $S_X \neq \emptyset$, so $\text{count}(X)/m(X) \geq \text{supp}(X) \geq minsup$, since $minrep \leq m(X) \leq \text{rep}(X)$. If X is inextensible, $S_X = \emptyset$, so $\text{count}(X)/m(X) = 0 < minsup$. $\square$

We now have a tangible way of checking the monotone extensibility property. The only remaining obstacle might be the counting of extensible yet infrequent sets. However, depending on the algorithm used, it is usually necessary to compute them anyway, since without them it would be impossible to compute the count of their supersets, some of which are indeed frequent.

Unfortunately, this does not completely solve the problem, since the aforementioned property is rather paradoxical. In order to prune the supersets of a set, we must first compute all of them to obtain the value $m(X)$. Hence, the theorem cannot be directly applied in an algorithm. However we really only need one direction of the equivalence, so we can use a lower bound for $m(X)$, as described in this corollary.

COROLLARY 1. *For an itemset X, let $k \leq m(X)$. Then $\frac{count(X)}{k} < minsup \Rightarrow X$ is not extensible.*

3. XMINER

Baseline algorithm. First, we propose a straightforward solution which will serve as a baseline for comparison. This algorithm is based on the observation that if an itemset X is representative, its support is at most $\text{count}(X)/minrep$. Otherwise checking this fraction is unnecessary. Using the corollary of theorem 1, *minrep* plays the role of lower bound. Formally, the algorithm first checks if $\text{rep}(X) \geq minrep$, and if true $\text{count}(X)$ is counted. If $\text{count}(X)/minrep < minsup$ the algorithm concludes X is inextensible. Otherwise the algorithm must continue with the supersets of X.

The fact that the $m(X)$ lower bound is a fixed constant for all itemsets, facilitates implementation of the baseline.

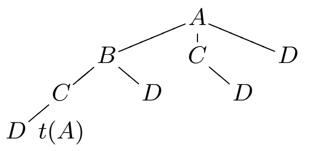


Figure 2: Partial subtree with tail of A marked

We choose to adapt Eclat [7] for its simplicity and speed. It does a depth first traversal, for each itemset maintaining a tid list of transactions supporting that itemset. At each step itemsets with a common prefix are combined to obtain larger ones (this is where the monotonicity of extensibility is used). Our implementation also uses diffsets as an optimization [8].

The baseline is equivalent to (albeit more efficient than) finding all representative itemsets whose count is larger than $minsup \times minrep$, and filtering out the frequent ones in a postprocessing step.

Main algorithm. Now we introduce XMiner, along with some specific improvements. We approximate $m(X)$ by a lower bound better than *minrep*, which is dependent of X (or, more accurately, X^*). Since $m(X)$ denotes a representativity, we simplify our search for it somewhat, by dropping the frequency constraint in S_X. This is actually equivalent to finding $\min\{\text{count}(Z^*) \geq minrep | Z^* \supseteq X^*\}$. We do this by making intersections, quickly looking ahead to the itemset at the the so-called tail, the longest path below the current itemset in the traversal tree (figure 2). This itemset has minimal representativity among all local supersets of the current itemset. If the tail is not representative we replace it with a better bound, namely *minrep*.

We add two optimizations. Firstly, the ordering of the itemsets. In regular depth-first itemset mining it is often beneficial to order the itemsets on support, putting less frequent ones at the roots of larger subtrees, yielding better pruning. However, XMiner orders the itemsets by their representativity instead, and not support, because of the erratic behavior of the support measure, and the impossibility of ordering attributes with several possible values. This not only improves pruning based on *minrep*, but also returns a better approximation for $m(X)$. The second is rather trivial. When, while computing the tail $\bigcap_j X_j^*$, we find that at step j we have $\text{rep}(\bigcap_{i<j} X_i^*) \leq minrep$, we can stop early. Note that together with ordering, this happens sooner.

In stead of intersecting itemsets to obtain the representativity of the tail, it is also possible to approximate it using basic set theory, using simple subtractions (similar to the MaxMiner algorithm [2]). Although this approximation is much faster, our experiments have shown this lower bound is far too weak to approximate $m(X)$, such that the overall effect is negative on both the efficiency (number candidate itemsets versus frequent and representative itemsets) and execution time of XMiner.

As a last remark, we point out that XMiner is equally efficient as the Eclat algorithm in complete datasets *i.e.* the number of candidate extensible or frequent itemsets is the same. Through the use of diffsets in the implementation, the execution time is also only marginally higher. (At depth greater than one in the subset lattice, all diffsets for representation become empty, resulting in negligible overhead.)

Algorithm 1 XMiner(set of itemsets P)

```
Require: $P$ is a set of representative and possibly extensible ordered itemsets with a common (omitted) prefix
 1. for all $X_i \in P$ do
 2.   compute $m'(X) = \max(\text{count}(\cap_{j \geq i} X_j^*), minrep)$
 3.   for all values $x_i$ of $X_i$ with
      $\text{count}(X_i = x_i)/m'(X_i) \geq minsup$ do
 4.     for all $X_j \in P$ with $X_j > X_i$ do
 5.       if $\text{count}(X_{ij} = *) \geq minrep$ then
 6.         for all values $x_j$ of $X_j$ do
 7.           $(X_{ij} = x_{ij}).tids =$
              $(X_i = x_i).tids \cap (X_j = x_j).tids$
 8.           if $\text{count}(X_{ij} = x_{ij})/m'(X_i) \geq minsup$ then
 9.             $P^i = P^i \cup \{(X_{ij} = x_{ij})\}$
10.           if $\text{count}(X_{ij} = x_{ij})/\text{count}(X_{ij} = *) \geq$
              $minsup$ then
11.             report $(X_{ij} = x_{ij})$ as frequent
12.     XMiner($P^i$)
```

Local Monotonicity The approximation $m'(X)$ only uses local information (in the traversal tree), which yields better (local) pruning. However, this also means that information about the inextensibility of X cannot be extrapolated to elsewhere in the lattice. This unfortunately prevents us from maximizing the use of the monotonicity of extensibility (otherwise possible by *e.g.* maintaining a trie of extensible itemsets while doing a right-most depth-first traversal).

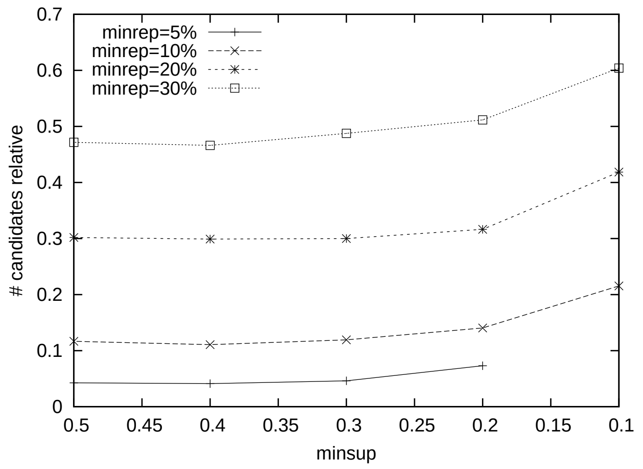
4. EXPERIMENTS

We ran experiments with XMiner and the baseline algorithm on two datasets: the results of the Eurovision song contest from 1957 to 2005, and the census income dataset from the UCI KDD Archive [3]. We implemented XMiner and the baseline algorithm in C++ using diffsets to optimize for speed [8], and compiled them with gcc 3.4.6 (all with -O3) on a machine with a 2.2 GHz Opteron CPU and 2GB of RAM, running Linux.

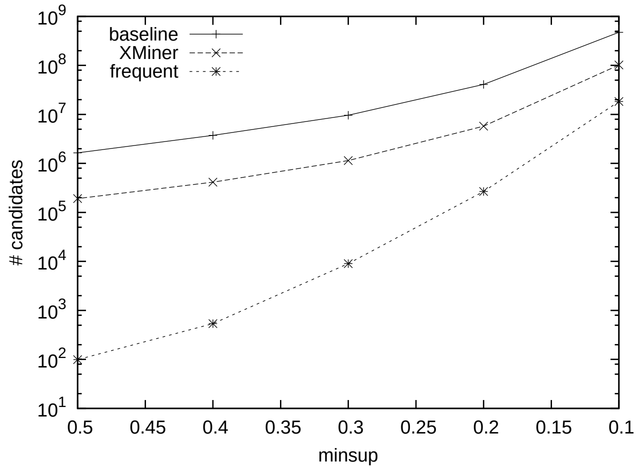
Eurosong dataset. In the Eurovision Song Contest a limited number of countries can enter and give points to the others. The countries that perform worst cannot enter in the next edition. Over the years more countries could join, recently a semi-final was introduced and countries that don't enter can cast votes, which previously wasn't possible. Hence the database has a lot of missing values. Each tuple represents a year-country combination, and has 2*43 attributes for points given to and received from other countries.

We fixed the minimum representativity at 5%, 10%, and 30%, and varied minsup, to look at the number of candidate sets and the execution time of XMiner, relative to the baseline algorithm. For the baseline algorithm with parameters minrep=5% and minsup=10%, the process was killed after 4000 seconds, so no comparison with XMiner could be made, and this point is not plotted (3(a)). XMiner's course doesn't follow the number of representative and frequent itemsets very tightly here, but still outperforms the baseline algorithm by almost an order of magnitude in generated itemsets (3(b)), and a factor of eight in execution time (3(c)).

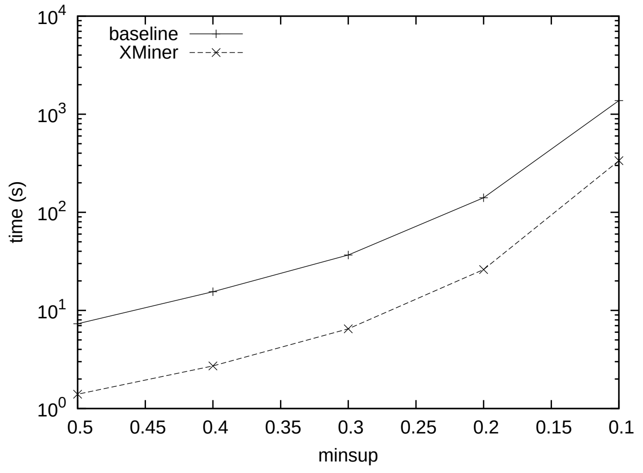
Some interesting results are listed here. We used both this and another version of the set with 50 tuples and 1849



(a)



(b) minrep=10%



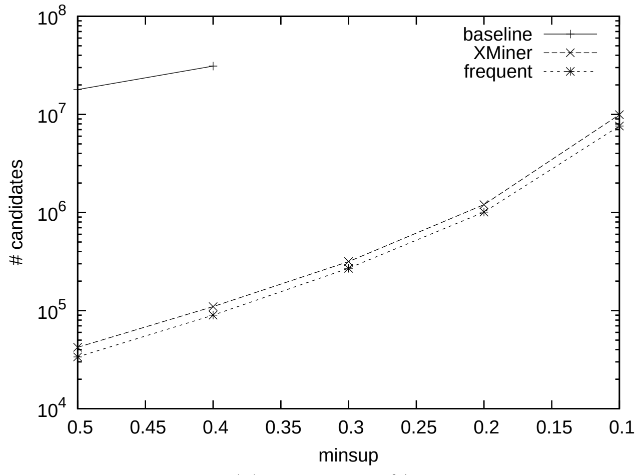
(c) minrep=10%

Figure 3: Eurosong dataset experiments

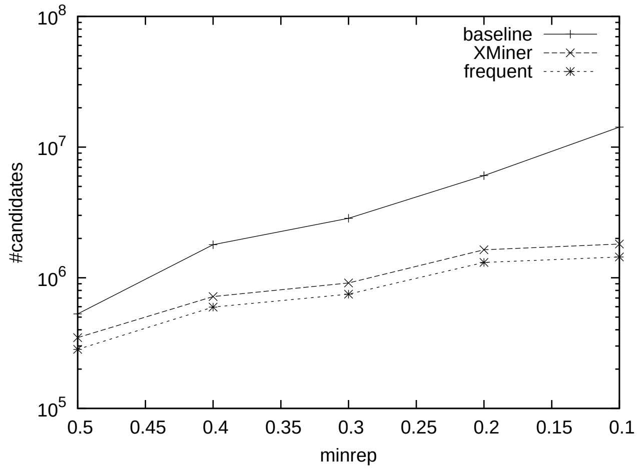
attributes for each country-country combination (for points given). Among others we found that the United Kingdom is rather popular. The itemset corresponding to the UK getting points from another country participating in the same year, or any country after the rules changed in 2004 (notice the need for accounting for nulls) is 67.8%. On the other side we find Finland, for which the itemset corresponding to not getting any points from other participating countries is 70.7%. (Note that our dataset did not yet include the results from the 2006 edition of the contest, when Finland actually won with a 'monster score'.) Examples from the other dataset are the following. If all three Scandinavian countries Denmark, Norway and Sweden enter the the competition in the same year, the support of Denmark giving points to Norway, Norway to Sweden, and Sweden to Denmark, is 30%. The support of Spain and Andorra giving each other points is even 100%. However, as Andorra only started entering the competition in 2004, the representativity is low.

Census income dataset. This is a large database of about 100MB. It has 33 nominal attributes and nearly 200k tuples. Some attributes have no missing values, other attribute values are missing in up to 50% of the tuples.

We mined it for varying minsup, minrep, First, we fixed minrep at 15% and varied minsup between %50 and 10%, using the original database (4(a)). Only for a minsup of 50% and 40% did the baseline algorithm finish within 4000 seconds. Still, this already shows the phenomenal difference with XMiner, and the efficiency of XMiner itself, the number of generated candidate itemsets follows the frequent ones very tightly. Second, we fixed minsup at 15% and ran experiments for different minreps (4(b)). To stay within reasonable time, we used our census 3 database (with the most values removed). For a decreasing minrep the number of frequent and representative itemsets does not increase dramatically. The number of generated candidate itemsets by XMiner increases similarly, while the baseline algorithm generates increasingly more candidates.



(a) minrep=15%



(b) minsup=15%

Figure 4: Census dataset experiments

5. CONCLUSIONS

In this paper we worked with new measures for itemsets and association rules, to be used in incomplete databases, and we implemented them in a novel algorithm, XMiner. Support and confidence account for missingness correctly, the representativity measure relates to the degree of nonmissingness of an itemset. Because support is no longer monotone, we define extensible itemsets, which have at least one frequent representative superset. Extensibility is monotone and since it subsumes frequency is used to traverse the exponential search space of itemsets. XMiner estimates the minimal representativity of the maximal supersets of an itemset to determine extensibility. Experiments show that it is very efficient and that it outperforms the most simple baseline algorithm.

Acknowledgments Thanks to the UCI KDD Archive [3] for the Census-Income Database. We would also like to thank Joost Kok for collecting and preprocessing the Eurovision Song Contest Database. This work was partially funded by the EU IQ project FP6-516169.

6. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD*, pages 207–216, 1993.

[2] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. ACM SIGMOD*, pages 85–93, Seattle, Washington, 1998.

[3] S. Hettich and S. D. Bay. The UCI KDD Archive [http://kdd.ics.uci.edu] Irvine, CA: University of California, Dept. of inf. and comp. science, 1999.

[4] M. Kryszkiewicz. Association Rules in Incomplete Databases. pages 84–93. Springer-Verlag London, UK, 1999.

[5] A. Ragel and B. Crémillieux. Treatment of missing values for association rules. In *Research and Development in Knowledge Discovery and Data Mining*, LNAI, 1998.

[6] R. Srikant and R. Agrawal. Mining generalized association rules. pages 407–419, 1995.

[7] M. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, May/June 2000.

[8] M. Zaki and K. Gouda. Fast vertical mining using diffsets. In *Proc. KDD*. ACM, 2003.