Mining Non-redundant Information-Theoretic Dependencies between Itemsets

Michael Mampaey*

Dept. of Mathematics and Computer Science, University of Antwerp
michael.mampaey@ua.ac.be

Abstract. We present an information-theoretic framework for mining dependencies between itemsets in binary data. The problem of closure-based redundancy in this context is theoretically investigated, and we present both lossless and lossy pruning techniques. An efficient and scalable algorithm is proposed, which exploits the inclusion-exclusion principle for fast entropy computation. This algorithm is empirically evaluated through experiments on synthetic and real-world data.

1 Introduction

The discovery of rules from data is a popular task in data mining. Mining association rules in transactional datasets has received a lot of attention especially [2–4]. The objective of association rule mining is to find highly confident rules between sets of items frequently occurring together. This has been generalized to, among others, relational tables with categorical or numerical attributes [5]. In this context, much attention has gone to the discovery of (approximate) functional dependencies in relations [6, 7]. A functional dependency $A \Rightarrow B$ between two sets of attributes is said to hold, if any two tuples agreeing on the attributes of A also agree on the attributes of B. Often it is desirable to also find rules that 'almost' hold. Typically, an error is associated with a functional dependency, which describes how well the relation satisfies that dependency, commonly this is the minimum relative number of tuples that need to be removed from the relation for the dependency to hold (known as g_3 [6]). These tuples can be thought of as being the exceptions to the rule. However, the fact that $A \Rightarrow B$ has a low error, does not necessarily imply that B strongly depends on A, in fact, A and B might even be independent.

Therefore, in this paper we take an information-theoretic approach to mining dependencies in binary data. We will describe the dependence of a rule based on the mutual information between consequent and antecedent. Furthermore, we use the entropy of a rule or itemset to describe its complexity. We present an algorithm called μ-Miner, which efficiently mines rules with a high dependence and a low complexity.

On top of this, we investigate what kinds of redundancy can occur in the collection of all low entropy, high dependence rules. For traditional association rules, several types of redundancy have been presented [8, 9]. We look at lossless closure-based redundancy in the context of this paper, as well as lossy pruning methods based on some information theoretical properties. These techniques are then integrated into our algorithm.

* Michael Mampaey is supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

** An extended version of this paper is available as a technical report [1].

2 Related Work

The discovery of exact and approximate functional dependencies from relations has received a lot of attention in the literature. The TANE algorithm proposed by Huhtala et al. [7] finds exact and approximate functional dependencies in a relation, which have a low g_3 error. TANE is a breadth-first algorithm that works with tuple partitions induced by attribute sets, which can be constructed in linear time with respect to the size of the relation. If the partition induced by XY does not refine the partition induced by X, then $X \Rightarrow Y$ is a functional dependency. The error of an approximate dependency can also be computed using these partitions. The main difference with our work is the way that the strength of a dependency is measured, but also that TANE only mines minimal rules, i.e. rules of the form $X \Rightarrow Y$ for which $|Y| = 1$ and there is no $X' \subsetneq X$ such that $X' \Rightarrow Y$ is an (approximate) functional dependency. On top of this, we also consider the complexity of dependencies.

Dalkilic and Robertson [10] use conditional entropy to determine the strength of dependencies in relational data. Their work examines several of their properties and information inequalities from a theoretical viewpoint, without focussing on rule discovery.

Heikinheimo et al. mine all low entropy sets from binary data, as well as trees based on these sets [11]. A breadth-first mining algorithm is proposed that exploits the monotonicity of entropy, after which a Bayesian tree structure is imposed on the itemsets. Its nodes correspond to the items, and the directed edges express the conditional entropy between them. The paper also discusses high entropy sets, argued to be potentially interesting due to the lack of correlation among their items.

Jaroszewicz and Simovici use information theoretic measures to assess the importance of itemsets or association rules on top of the traditional support/confidence based mining framework [12]. They use Kullback-Leibler divergence to determine the redundancy of confident association rules. Given the supports of some subsets of an association rule, its most likely confidence is computed (using a maximum entropy model); if the actual confidence is close to the estimate, the rule is considered to be redundant.

3 Preliminaries and Notation

Below we establish some notation and introduce some concepts that are used later on. We are given a set of items $\mathcal{I}$. A dataset $\mathcal{D}$ is a bag of transactions t, which are subsets of $\mathcal{I}$. The collection of all transactions is denoted $\mathcal{T}$. We write single items as x, y, z and itemsets as X, Y, Z. For the sake of brevity, we use the shorthands xyz for a set $\{x, y, z\}$, XY for the union of sets $X \cup Y$, and X-Y for set difference $X \setminus Y$. A rule between two itemsets X and Y is written as $X \Rightarrow Y$, where both sets are assumed to be either disjoint ($X \cap Y = \emptyset$) or inclusive ($X \subseteq Y$), depending on the context. The support *supp* and frequency *fr* of the pair $(X = v)$, with $v \in \{0, 1\}^{|X|}$, are the absolute and relative number of transactions $t \in \mathcal{D}$ for which $t_X = v$, respectively.

The notion of entropy as a measure of information was introduced by Shannon [13]. Given a discrete random variable X with values v in a domain $dom(X)$, and a probability distribution p, the entropy of X is defined as $H(X) = -\sum_{v \in dom(X)} p(v) \log_2 p(v)$. For brevity, if $dom(X) = \{0, 1\}$ we also write $H(f)$ where $f = fr(X = 1) \in [0, 1]$.

The entropy expresses the amount of information that is contained in a random variable, expressed in bits. Alternatively, it can be seen as a measure of its complexity or of its uncertainty. The mutual information between two discrete random variables X and Y is measured as the divergence of the joint distribution $p(v, u)$ from the product distribution $p(v)p(u)$; $I(X, Y) = \sum_{v \in dom(X), u \in dom(Y)} p(v, u) \log_2 \frac{p(v,u)}{p(v)p(u)}$. If X and Y are independent then the mutual information is zero and vice versa. Mutual information can conveniently be expressed in terms of entropy, as $I(X, Y) = H(Y) - H(Y \mid X) = H(X) + H(Y) - H(X, Y)$, where $H(Y \mid X)$ is the conditional entropy of Y given X, and $H(X, Y)$ is the joint entropy of X and Y.

4 Strong Dependence Rules

In this section we define our interestingness measures for itemsets and rules, and explore some of their properties.

4.1 Definitions

In the following, an itemset X is seen as a discrete random variable with $dom(X) = \{0, 1\}^{|X|}$. For the probability distribution p, the frequency distribution *fr* is taken.

Definition 1 (Rule Entropy). *Let X and Y be two disjoint itemsets. The entropy h of the rule $X \Rightarrow Y$ is defined as the joint entropy of X and Y: $h(X \Rightarrow Y) = H(X \cup Y)$.*

It is easy to see that for any set X it holds that $0 \leq h(X) \leq \log_2 |dom(X)| = |X|$.

Definition 2 (Rule Dependence). *Let X and Y be two disjoint itemsets. We define the dependence μ of the rule $X \Rightarrow Y$ as*

$$\mu(X \Rightarrow Y) = \frac{I(X, Y)}{H(Y)}.$$

By dividing by $H(Y)$ we obtain a normalized, asymmetric variant of mutual information ranging between 0 and 1. When X an Y are independent then $\mu(X \Rightarrow Y) = 0$, this means that X tells us nothing about Y. On the other hand, $\mu(X \Rightarrow Y) = 1$ if and only if X fully determines Y; in this case the rule is called exact.

4.2 Properties

We describe some useful properties of h and μ which we exploit to construct an efficient set and rule mining algorithm later on. Due to space restrictions proofs are omitted but can be found in the technical report [1].

Theorem 1 (Monotonicity of Entropy). *Let X and X' be two itemsets. If $X \subseteq X'$, then $h(X) \leq h(X')$.*

Using the monotonicity of h, it is possible to efficiently traverse the search space of all itemsets in a typical Apriori-like breadth-first algorithm, or a memory-efficient depth-first algorithm as μ-Miner does.

Theorem 2 (Antecedent Monotonicity). *Let X, X' and Y be itemsets with $X \subseteq X'$, then $\mu(X \Rightarrow Y) \leq \mu(X' \Rightarrow Y)$.*

Theorem 2 implies that rules containing all of the items in $\mathcal{I}$ have the highest dependence. However, the entropy of such rules is also very high, and hence they will be pruned. Furthermore, some of the items might be independent of the other ones, and it is quite likely that such large rules display some sort of redundancy as described in Section 5.

Theorem 3 (Partial Monotonicity). *Let $X \Rightarrow Y$ be a rule, where X and Y are disjoint. There exists an item $y \in Y$ such that $\mu(X \Rightarrow Y) \leq \mu(Xy \Rightarrow Y\text{-}y)$.*

This last theorem allows us to systematically and efficiently construct all rules with a high dependence from a given low entropy set, in a levelwise fashion. This can be achieved without having to consider the exponential number of possible rules that can possibly be constructed from that itemset.

4.3 Closedness

Due to the explosion of patterns commonly encountered in classic frequent itemset mining, one often turns to mining a condensed representation of a collection of frequent itemsets. Such pattern collections are typically much smaller in magnitude, can be discovered faster, and it is possible to infer other patterns from them. One example are maximal frequent itemsets [14, 15]. Two other popular condensed representations are closed and non-derivable frequent itemsets, which we extend to our framework.

The concept of closedness is well-studied for support based itemset mining [16]. An itemset is closed with respect to support if all of its proper supersets have a strictly smaller support. A closure operator can be defined that maps an itemset to its (unique) smallest closed superset, i.e. its closure. Similarly, we can consider itemsets that are closed with respect to entropy. We formally do this as follows. The set inclusion relation ($\subseteq$) defines a partial order on the powerset $\mathcal{P}(\mathcal{I})$ of all itemsets. Furthermore, a partial order, i.e. refinement ($\sqsubseteq$), can be defined on the set $\mathcal{Q}(\mathcal{T})$ of all transaction partitions. A given itemset $X \in \mathcal{P}(\mathcal{I})$ partitions $\mathcal{T}$ into equivalence classes according to the value of X in all transactions, and conversely a partition in $\mathcal{Q}(\mathcal{T})$ corresponds to an itemset in $\mathcal{P}(\mathcal{I})$. (The entropy of an itemset is computed using the sizes of the equivalence classes in its corresponding partition.) Let us call these two mapping functions i_1 and i_2. It can be shown that i_1 and i_2 form a Galois connection between $(\mathcal{P}(\mathcal{I}), \subseteq)$ and $(\mathcal{Q}(\mathcal{T}), \sqsubseteq)$. The composition $cl := i_2 \circ i_1$ defines a closure operator on $\mathcal{P}(\mathcal{I})$, which satisfies the following properties for all itemsets X.

$$\begin{cases} X \subseteq cl(X) & \text{(extension)} \\ cl(X) = cl(cl(X)) & \text{(idempotency)} \\ X \subseteq X' \Rightarrow cl(X) \subseteq cl(X') & \text{(monotonicity)} \end{cases}$$

Definition 3. *We call an itemset $X \subseteq \mathcal{I}$ closed if $X = cl(X)$. Conversely, the set X is called a generator if for all $X' \subsetneq X$ it holds that $cl(X') \neq X$.*

It holds that all proper supersets of a closed itemset have a strictly higher entropy and $h(X) = h(cl(X))$. All proper subsets of a generator have strictly lower entropy. Note that the rule $X \Rightarrow Y$ is exact if and only if $X \subseteq XY \subseteq cl(X)$. Furthermore, if an exact rule $X \Rightarrow Y$ is minimal, then X is a generator.

4.4 Derivability

The idea of itemset derivability was introduced by Calders and Goethals [17]. Given the supports of all proper subsets of an itemset $(X = 1)$, it is possible, using the inclusion-exclusion principle, to derive tight lower and upper bounds on its support. If these bounds coincide we know $supp(X = 1)$ exactly, and $(X = 1)$ is called derivable (with respect to support). The set of all frequent itemsets can thus be derived from the set of all non-derivable frequent itemsets. Similarly, we can define the derivability of the entropy of an itemset.

Definition 4. *We call X h-derivable if its entropy can be determined exactly from the entropies of its proper subsets.*

The set of all non-derivable itemsets is downward closed. Interestingly, an itemset X is h-derivable if and only if it is derivable with respect to support.

5 Rule Redundancy

Mining all low entropy, high dependence rules can yield a very large set of patterns, which is not desirable for a user who wants to analyze them. Typically, this collection contains a lot of redundant rules. In this section we investigate how we can characterize and prune such rules. We define two types of redundancy, one that is based on the closure of itemsets, and one that is based on the superfluous augmentation of the antecedent or consequent of a rule.

5.1 Closure-based Redundancy

As mentioned in the previous section, rules of the form $X \Rightarrow cl(X)$ are always exact. It should be clear that combining an arbitrary rule with an appropriate exact rule yields a new rule with identical entropy and dependence. For instance, if $A \Rightarrow B$ is exact, then $\mu(A \Rightarrow C) = \mu(AB \Rightarrow C)$.

Theorem 4. *Let $X \Rightarrow Y$ and $X' \Rightarrow Y'$ be two rules, where $X \subseteq X' \subseteq cl(X)$ and $Y \subseteq Y' \subseteq cl(Y)$. Then $h(X \Rightarrow Y) = h(X' \Rightarrow Y')$ and $\mu(X \Rightarrow Y) = \mu(X' \Rightarrow Y')$.*

Since the entropy and dependence of such larger rules can be inferred using the smaller rule and the closure operator, we call them redundant. These minimal rules are constructed using generators.

Definition 5 (Closure-based Redundancy). *A rule $X \Rightarrow Y$ is redundant with respect closure if*

$$\begin{cases} X \text{ is not a generator or } |Y| > 1 & \text{if } \mu(X \Rightarrow Y) = 1, \\ XY \text{ is not a generator} & \text{if } \mu(X \Rightarrow Y) < 1. \end{cases}$$

Note that if XY is a generator, then X and Y are also generators, since the set of all generators is downward closed.

5.2 Augmentation Redundancy

Here we define a stricter kind of redundancy that prunes rules which have items unnecessarily added to their antecedents or consequents.

Antecedent Redundancy Suppose we have two rules with a common consequent, $X \Rightarrow Y$ and $X' \Rightarrow Y$, with $X' = X \cup \{x\}$. Theorem 2 tells us that $\mu(X \Rightarrow Y) \leq \mu(X' \Rightarrow Y)$. Even though the latter rule has a higher dependence, it might be redundant if x does not make a real contribution to the rule. For instance, if X and x are independent, then $\mu(X' \Rightarrow Y)$ is simply the sum of $\mu(X \Rightarrow Y)$ and $\mu(x \Rightarrow Y)$. To characterize this type of redundancy we use the chain rule of mutual information, $I(Xx, Y) = I(X, Y) + I(x, Y \mid X)$, where the last term is the conditional mutual information (which can be written as $H(x \mid X) - H(x \mid XY)$). It is known that I does not behave monotonically with conditioning. In the case where X and x are independent, we have $I(x, Y \mid X) = I(x, Y)$. If X already explains part of the dependency between x and Y, then $I(x, Y \mid X) < I(x, Y)$, meaning there is an "overlap" between X and x. Otherwise, if $I(x, Y \mid X) > I(x, Y)$, this means that under knowing X, the mutual information between x and Y increases. Intuitively, it means that Xx tells us more about Y than the sum of X and x separately. This motivates the following definition.

Definition 6 (Antecedent Redundancy). *A rule $X \Rightarrow Y$ is redundant with respect to antecedent augmentation, if there exists an item $x \in X$ such that*

$$\begin{cases} \mu(X \Rightarrow Y) \leq \mu(X\text{-}x \Rightarrow Y) + \mu(x \Rightarrow Y), \textit{or} \\ X\text{-}x \Rightarrow Y \textit{ is redundant.} \end{cases}$$

It follows that $\mu(X \Rightarrow Y) > \sum_{x \in X} \mu(x \Rightarrow Y)$ if the rule $X \Rightarrow Y$ is non-redundant.

Consequent Redundancy Consider the rule $X \Rightarrow Y$ and an item $y \notin XY$. Unlike in the previous section, μ is not monotonic with respect to augmentation of the consequent, so in general the dependence of $X \Rightarrow Yy$ can either be higher or lower that that of $X \Rightarrow Y$. An increase in μ means that adding y to Y increases the mutual information $I(X, Y)$ more than it increases the entropy $H(Y)$. Put differently, the relative increase in uncertainty from $H(Y)$ to $H(Yy)$, is surpassed by the increase of the amount of information X gives about Y and Yy. X gives relatively less information about Yy than it does about Y.

Definition 7 (Consequent Redundancy). *A rule $X \Rightarrow Y$ is redundant with respect to consequent augmentation, if there exists an item $y \in Y$ such that*

$$\begin{cases} \mu(X \Rightarrow Y) \leq \mu(X \Rightarrow Y\text{-}y), \textit{or} \\ X \Rightarrow Y\text{-}y \textit{ is redundant.} \end{cases}$$

It follows that if $X \Rightarrow Y$ is non-redundant, then $\forall Y' \subset Y : \mu(X \Rightarrow Y') < \mu(X \Rightarrow Y)$.

Relation to Closure-based Redundancy It turns out that augmentation redundancy is strictly stronger than closure-based redundancy, as stated in the theorem below.

Theorem 5. *If a rule $X \Rightarrow Y$ is redundant with respect to closure, then it is also redundant with respect to antecedent augmentation or consequent augmentation.*

Algorithm 1 μ-Miner

Input: Binary dataset $\mathcal{D}$, thresholds $h_{\max}$ and $\mu_{\min}$
Output: Non-redundant, low entropy, high dependence rules
1: $\mathcal{P} \leftarrow \{\{x\} \subset \mathcal{I}; h(x) \leq h_{\max}\}$
2: SetMine($\mathcal{P}$, $h_{\max}$, $\mu_{\min}$)

6 The μ-Miner Algorithm

In this section we present μ-Miner (see Algorithm 1). As input it expects a dataset $\mathcal{D}$, a maximum entropy threshold $h_{\max}$, and a minimum dependence threshold $\mu_{\min}$. The algorithm efficiently mines low entropy itemsets, and from these sets strong dependence rules are constructed. Further, μ-Miner prunes rules that are closure redundant or augmentation redundant. Computation of entropy and dependence, and the checking of redundancy is performed by doing some simple arithmetic operations and lookups, and only one scan of the database is required.

6.1 Mining Itemsets

In the SetMine function (Algorithm 2), itemsets with a low entropy are mined. This recursive function takes a collection of itemsets with a common prefix as input, initially this is the set of all low entropy singletons. The search space is traversed in a depth-first, right-most fashion. This is less memory-intensive than a breadth-first approach, and the right-most order ensures that when an itemset is considered, all of its subsets will already have been visited in the past (lines 1&3), a fact we exploit later. This implies that we need to impose an order on the itemsets, e.g. a simple lexicographical ordering. In our implementation of μ-Miner we use a heuristic ordering based on the entropy of the items, such that large subtrees of the search space are rooted by sets which are expected to be have a high entropy, allowing us to prune larger parts of the subspace.

Algorithm 2 SetMine

Input: Itemset collection $\mathcal{P}$, thresholds $h_{\max}$ and $\mu_{\min}$
1: **for** X_1 in $\mathcal{P}$ in descending order **do**
2: $\quad\mathcal{P}' \leftarrow \emptyset$
3: $\quad$**for** $X_2 < X_1$ in $\mathcal{P}$ **do**
4: $\quad\quad X \leftarrow X_1 \cup X_2$
5: $\quad\quad$Compute and store $fr(X = 1)$
6: $\quad\quad h(X) \leftarrow$ EntropyQIE(X)
7: $\quad\quad$**if** X is not a generator **then**
8: $\quad\quad\quad$Report corresponding exact rule(s)
9: $\quad\quad$**else if** $h(X) \leq h_{\max}$ **then**
10: $\quad\quad\quad\mathcal{P}' \leftarrow \mathcal{P}' \cup \{X\}$
11: $\quad\quad\quad$RuleMine(X, $\mu_{\min}$)
12: $\quad$SetMine($\mathcal{P}'$)

Algorithm 3 EntropyQIE

Input: Candidate itemset $X \subset \mathcal{I}$
Output: $h(X)$, the entropy of X
1: **for all** $X' \subseteq X$ **do**
2: $p(X') \leftarrow fr(X' = 1)$
3: **for all** $x \in X$ **do**
4: **for all** $X' \subseteq X$ with $x \in X'$ **do**
5: $p(X'\text{-}x) \leftarrow p(X'\text{-}x) - p(X')$
6: **return** $h(X) = -\sum_{X' \subseteq X} p(X') \cdot \log_2\ p(X')$

6.2 Efficiently Computing Entropy

A straightforward method to compute $h(X)$ is to perform a scan over the database to obtain the frequencies of $(X = v)$ for all values $v \in \{0,1\}^{|X|}$. In total there are 2^k such frequencies for $k = |X|$, however, at most $|\mathcal{D}|$ of them are nonzero and hence this method requires $O(|\mathcal{D}|)$ time. If the database fits in memory this counting method is perfectly feasible, otherwise it becomes too expensive, since database access is required for each candidate itemset. Another option is to use the partitioning technique used by TANE [7]. For each itemset a partition of the transactions is explicitly computed in $O(|\mathcal{D}|)$ time, and then the sizes of the sets in the partition can be used to compute $h(X)$.

μ-Miner uses a different entropy computation method that does not require direct database access, and has a lower complexity, which is beneficial especially for large datasets. We start from a simple right-most depth-first itemset support mining algorithm similar to Eclat [4], and store the supports in a trie (line 5). When we have mined the support of $(X = 1)$, the frequencies of all $(X = v)$ are computed with the stored supports of all subsets, by using quick inclusion-exclusion (Algorithm 3), which takes $O(k \cdot 2^{k-1})$ time [18]. However, we can again use the fact that at most $|\mathcal{D}|$ frequencies are nonzero, hence this counting method is $O(\min(k \cdot 2^{k-1}, |\mathcal{D}|))$. The advantage of our method is that it is fast and it does not require database access. The disadvantage is that the supports of all mined itemsets must be stored, which may be a problem if memory is scarce and $h_{\max}$ is set rather high. Note that if we were to restrict ourselves to mining only non-derivable itemsets, we know that $k \leq \lceil \log_2 |\mathcal{D}| \rceil$ [17]. In this case the total number of frequencies we need to store is $O(|\mathcal{I}|^{\log_2 |\mathcal{D}|})$ in the worst case, which is polynomial in $|\mathcal{I}|$ for a fixed database size, and polynomial in $|\mathcal{D}|$ for a fixed number of items.

6.3 Mining Non-redundant Dependence Rules

For each low entropy itemset, RuleMine (Algorithm 4) is called to generate high dependence rules. It starts with rules whose consequent is a singleton, and then moves items from the antecedent to the consequent. By using the partial monotonicity property from Theorem 3, not all 2^k possible rules need to be considered. Since we have the entropies of all subsets available to us, we can compute the dependence by performing just a few lookups. If a rule is found to have high dependence, it is checked whether the rule is redundant (line 6). Again, since we have the entropies of all subsets available, these redundancy checks can be performed quite efficiently.

Algorithm 4 RuleMine

Input: Low entropy itemset X; dependence threshold $\mu_{\min}$
Output: Non-redundant strong dependence rules based on X

```
1: L ← {X-x ⇒ x; x ∈ X}
2: while L ≠ ∅ do
3:    for A ⇒ B in L do
4:       Compute μ(A ⇒ B)
5:       if μ(A ⇒ B) ≥ μ_min and A ⇒ B is non-redundant then
6:          Report A ⇒ B
7:    L ← {A-a ⇒ Ba; A ⇒ B ∈ L, using Theorem 3}
```

7 Experimental Evaluation

We perform experiments on several datasets to evaluate the efficiency of μ-Miner. We also investigate the effect of our pruning techniques. The algorithm is implemented in C++[1], and the experiments were executed on a machine with a 2.2GHz CPU and 2GB of memory, running Linux. More experiments can be found in the technical report [1].

7.1 Datasets

First, we have some benchmark datasets taken from the FIMI Repository [19]: CHESS, MUSHROOM and PUMSB, containing 3196, 8124 and 49046 transactions respectively. The original PUMSB dataset contains 2112 items, in our experiments we used the 100 most high entropy items. MUSHROOM originally has 119 items, for our experiments we removed items with frequencies higher than 0.9 or lower than 0.1. These datasets are used to test the efficiency of μ-Miner.

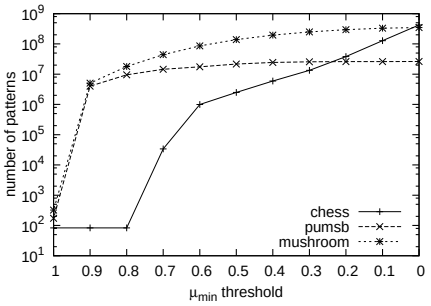
Second, we generated a SYNTHETIC dataset which contains an embedded pattern. We use it to evaluate the scalability of μ-Miner with respect to the size of the database. The dataset consists of 1 000 000 transactions and has 16 items. The 15 first items are independent and have random frequencies between 0.1 and 0.9. The last item equals the sum of the other two modulo 2, i.e. the rule $\{1, \ldots, 15\} \Rightarrow \{16\}$ is an exact one. Note that this dependency is also minimal.

7.2 Experiments

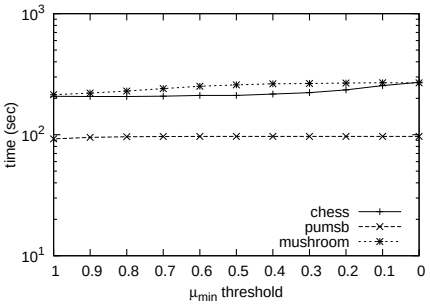
First, we perform some experiments on the benchmark datasets. To begin with, we set the value of $h_{\max}$ to a fixed value (1.5 for CHESS, 2 for PUMSB, and 3.5 for MUSHROOM), and we vary the minimum dependence threshold $\mu_{\min}$ between 0 and 1. As can be seen in Figure 1a, the number of rules increases roughly exponentially when $\mu_{\min}$ is decreased. Noticeably, the execution times stay roughly constant as $\mu_{\min}$ decreases, as shown in Figure 1b. This is not surprising, since most computations are performed in the itemset mining phase, and the computation of μ involves just a few lookups. Next, we set the value of $\mu_{\min}$ to a fixed value (in this case 0.4 for all datasets), and gradually

[1] The source code of μ-Miner is publicly available at http://www.adrem.ua.ac.be

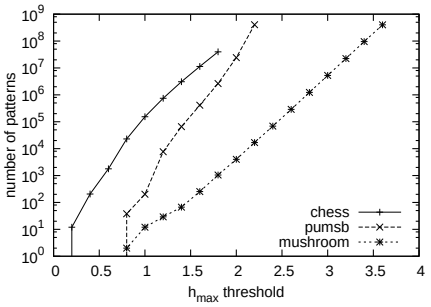
increase the maximum entropy threshold $h_{\max}$ from zero upward. In Figure 1c we see that for very low values of $h_{\max}$ no rules are found. Then, the number of rules increases exponentially with $h_{\max}$, which is to be expected. In Figure 1d we see that this trend also translates to the execution times. For lower thresholds ($h_{\max} \leq 1$) the runtimes stay roughly constant, because they are dominated by I/O time.



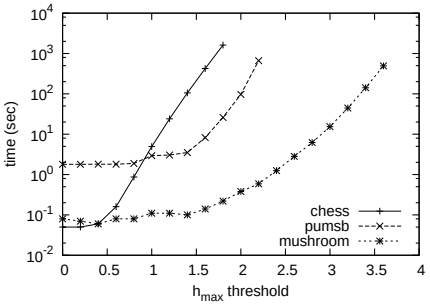
(a) Number of rules for varying $\mu_{\min}$



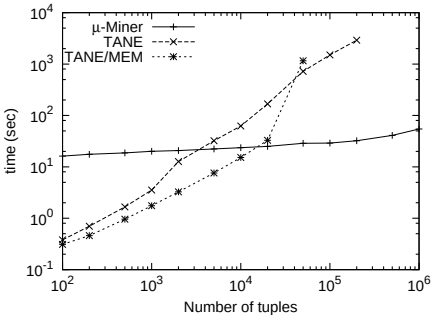
(b) Execution times for varying $\mu_{\min}$



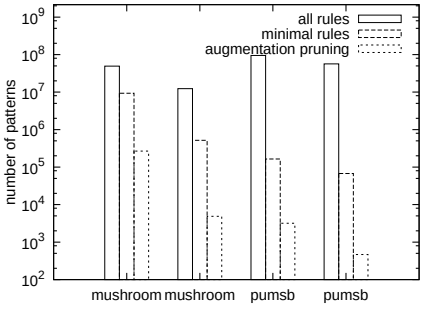
(c) Number of rules for varying $h_{\max}$



(d) Execution times for varying $h_{\max}$



(e) μ-Miner and TANE scalability on SYNTHETIC



(f) Effect of redundancy pruning

Fig. 1: Experimental results.

Secondly, we evaluate the scalability of μ-Miner with respect to the size of the database using the SYNTHETIC dataset. The aim is to discover the embedded functional dependency, in order to do this we set $\mu_{\min}$ to 1, and $h_{\max}$ sufficiently high (say, 16). We compare μ-Miner with the TANE and TANE/MEM implementations from [20]. The main TANE algorithm stores partitions to disk level per level, while the TANE/MEM variant entirely operates in main memory. The number of transactions is gradually increased from 10^2 to 10^6 and the runtimes are reported in Figure 1e. We see that all algorithms scale linearly with $|\mathcal{D}|$, although the slope is much steeper for TANE and TANE/MEM, while the execution time of μ-Miner increases very slowly. At around ± 3000 transactions, μ-Miner overtakes TANE in speed, and at 10^5 transactions our algorithm is already two orders of magnitude faster. The TANE/MEM algorithm is faster up to ± 10000 transactions, but cannot handle any datasets much larger than that due to heavy memory consumption. This observed difference in speed can be explained entirely by the counting method. TANE explicitly constructs a partition of size $O(|\mathcal{D}|)$ for each itemset (and stores these to disk or in memory level per level), while our algorithm computes the required sizes of the partitions without actually constructing them. The increase in the execution time of μ-Miner can be accounted for almost entirely by the increase in time it takes to read the data file.

Next, let us investigate how redundancy pruning affects the size of the output. We experimented on the MUSHROOM and the PUMSB datasets for different values of $h_{\max}$ (3 for MUSHROOM and 1.5 for PUMSB) and $\mu_{\min}$ (0.2 and 0.8 for both datasets). The results are shown in Figure 1f. For the MUSHROOM dataset pruning all non-minimal rules already reduces the output by roughly an order of magnitude. Augmentation pruning reduces the output by an additional two orders of magnitude. For the PUMSB dataset pruning non-minimal rules reduces the output by three orders of magnitude. Here augmentation pruning reduces the output in size even further, by roughly two orders of magnitude. This makes the result collection of rules far more manageable for a user.

8 Conclusions

We proposed the use of information-theoretic measures based on entropy and mutual information to mine dependencies between sets of items. This allows us to discover rules with a high statistical dependence, and a low complexity. We investigated the problem of redundancy in this framework, and proposed two techniques to prune redundant rules. One is based on the closure of itemsets and is lossless, while the other, shown to be stronger, is lossy and penalizes the augmentation of rules with unrelated items. We presented our algorithm μ-Miner, which mines such dependencies and applies the presented pruning techniques. Several experiments showed that μ-Miner is efficient and scalable: it can easily handle datasets with millions of transactions and does not require a large amount of memory. Furthermore, our pruning techniques were shown to be very effective in reducing the size of the output by several orders of magnitude.

References

1. Mampaey, M.: Mining non-redundant information-theoretic dependencies between itemsets. Technical report, University of Antwerp (2010)
2. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. ACM SIGMOD Record **22**(2) (1993) 207–216
3. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. ACM SIGMOD Record **29**(2) (2000) 1–12
4. Zaki, M., Parthasarathy, S., Ogihara, M., Li, W., et al.: New algorithms for fast discovery of association rules. Proceedings of KDD. (1997)
5. Srikant, R., Agrawal, R.: Mining quantitative association rules in large relational tables. ACM SIGMOD Record **25**(2) (1996) 1–12
6. Kivinen, J., Mannila, H.: Approximate inference of functional dependencies from relations. Theoretical Computer Science **149**(1) (1995) 129–149
7. Huhtala, Y., Karkkainen, J., Porkka, P., Toivonen, H.: TANE: An efficient algorithm for discovering functional and approximate dependencies. The Computer Journal **42**(2) (1999) 100–111
8. Zaki, M.J.: Generating non-redundant association rules. In: Proceedings of KDD. (2000) 34–43
9. Balcázar, J.L.: Minimum-size bases of association rules. In: Proceedings of ECML PKDD. (2008) 86–101
10. Dalkilic, M.M., Robertson, E.L.: Information dependencies. In: Proceedings of ACM PODS. (2000) 245–253
11. Heikinheimo, H., Hinkkanen, E., Mannila, H., Mielikäinen, T., Seppänen, J.K.: Finding low-entropy sets and trees from binary data. In: Proceedings of KDD (2007) 350–359
12. Jaroszewicz, S., Simovici, D.A.: Pruning redundant association rules using maximum entropy principle. In: Proceedings of PAKDD. (2002) 135–147
13. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal **27** (1948) 379–423
14. Bayardo Jr, R.: Efficiently mining long patterns from databases. In: Proceedings of ACM SIGMOD (1998) 85–93
15. Gouda, K., Zaki, M.: Efficiently mining maximal frequent itemsets. In: Proceedings of IEEE ICDM. (2001) 163–170
16. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Proceedings of ICDT (1999) 398–416
17. Calders, T., Goethals, B.: Non-derivable itemset mining. Data Mining and Knowledge Discovery **14**(1) (2007) 171–206
18. Calders, T., Goethals, B.: Quick inclusion-exclusion. LNCS **3933** (2006) 86–103
19. Goethals, B.: Frequent itemset mining implementations repository http://fimi.cs.helsinki.fi/data
20. Huhtala, Y., Karkkainen, J., Porkka, P., Toivonen, H.: TANE homepage http://www.cs.helsinki.fi/research/fdk/datamining/tane