Essay Pattern Set Mining

Tamara Florijn (5856442)

June 2020

1 Introduction

We, humans, experience a never satisfied hunger to knowledge and insight about the world around us. We try to model the world, save data in databases and discover interesting information and patterns about the data. In a data set, a lot of patterns are hidden. But not all of them are equally interesting or useful. A first obstacle is the usually huge amount of time it takes to generate all the patterns. But if that succeeds, the second obstacle involves the programmer, who cannot possibly process hundreds, thousands or millions of patterns in order to understand them or distillate information out of them to take action. Therefore, a screening needs to take place. But how? You can raise the bar of interestingness, with the risk of just revealing common knowledge. And next to that, "interestingness" in itself is difficult to define. One can use constraints to define this. Or you might condense the representation, such that less patterns actually represent more information about the data. Instead of looking at the interestingness of each pattern individually, one might aim at returning the most interesting *set* of patterns. These do not necessarily consist of the individually best patterns, for one pattern may cover (almost) all the information that another pattern covers as well, making it (almost) redundant. This is called *pattern set mining.*

In this essay, we will discuss a myriad of approaches to pattern (set) mining. How can we efficiently return the set of patterns that is most useful to the user?

2 Introduction on FPM

Before we can move on to the big problems and solutions, we will first give an introduction on important concepts.

To explain the concepts necessary for understanding thoroughly, we will use a metaphor of a supermarket throughout this section. Suppose you work for a supermarket chain, and your boss would like to gain more insight in the purchases of its clients, for example what combinations of products often occur (Kale and smoked sausages? Apples and pears? Face masks and antibacterial soap?). He keeps record of every transaction (purchase). You start to wonder that the patterns your boss is looking for might have some similarities with rules: if ... [a customer bought face masks], then ... [he likely also bought antibacterial soap]. Your first idea, just simply listing all possible combinations and calculating how often each pattern occurs, is quickly put aside: the number of patterns is way too large to compute in reasonable time. Next to that, how would you pick out a valuable pattern to return to your boss? There are way too many! This is known as *pattern explosion.* How should you deal with this?

First, we formalise as following. A *transaction database* D is a set of transactions, where a transaction is a subset of all items $\mathcal{I}$ for sale. Note, the transaction only shows whether or not a certain product is bought, not how many copies. An *itemset* I is a set of items $I \subseteq \mathcal{I}$, and transaction t contains itemset I if $I \subseteq t$. Suppose, we would like to know often itemsets appear in transactions. We define this number as the *support* of an itemset, and we call it a *frequent* or *large* itemset if the itemset satisfies our frequency demand *minSup* (minimal support), or a *small* itemset otherwise. An *association rule* is an implication of the form $X \Rightarrow Y$, with disjoint itemsets X (antecedent) and Y (consequent). This rule has *confidence* c if $c\%$ of the transactions that contain X also contain Y. The *support* s of a rule is the percentage of transactions in D that contain $X \cap Y$.

Now, we might be able to define more formally what the boss wants you to achieve. First, it would be interesting to know which items are frequently bought together. We can phrase this as:

1. Generate all itemsets with support above threshold minSup. This is called *frequent itemset mining.*

What we also mentioned, was the discovery of rules. Can we find such patterns in the transaction database of the supermarket? We define this as:

2. For a given large itemset Y with k items, generate all rules that use the items in Y. We will look for rules where the antecedent has $k-1$ items. The consequent exactly has one: the one item in Y that is not in the antecedent of the rule.

These goals were formulated before in paper [1].

Now, a light bulb lights above your head. If very little people ever buy apples, we know for sure that the combination of apples and pears will not occur often, not more often than the support of apples or the support of pears. This is called the *A Priori* property, formally expressed as

$$X \subseteq Y \Rightarrow supp_D(X) \geq supp_D(Y)$$

Here, $supp_D(X)$ is the support of itemset X in database D.

2.1 AIS algorithm

In paper [1], an algorithm, from now on called the AIS algorithm, has been introduced to tackle this problem, using the A Priori property. AIS makes multiple passes over the databases, while candidate itemsets (expected large itemsets) are generated. For each transaction encountered, one checks if this transaction contains large itemsets, determined in the previous pass. These large itemsets are extended with (large) items from the transaction, and then added to the set of candidate itemsets. Note, small items don't need to be added, given the A Priori property. Still, (too) many candidate itemsets are generated.

2.2 Apriori algorithm

Paper [2] introduces a faster algorithm (for large scale problems) for finding association rules between items in a database, called *Apriori*, with fewer redundant generated candidates. First, the large itemsets L_{k-1} of size $k-1$ found in pass $k-1$ are used to generate the candidate itemsets C_k. In contrast with the previous article, not all 1-extensions are generated. Instead, a self-join rule is used, where the itemsets A and B are merged if first $k-2$ items are identical, after which item $k-1$ of A and item $k-1$ of B are concatenated. To avoid duplicates, item $k-1$ of A should be lexicographical smaller than item $k-1$ of B. Then, one deletes all generated itemsets that have small $k-1$-subsets, i.e. subsets that are not in C_{k-1}. Note, still some small itemsets will be generated. Both AIS and Apriori algorithms focus on the sub-problem 1, and direct for sub-problem 2 to other papers.

2.3 Frequent Episode Mining

Pattern set mining can be applied to many different fields (so not only supermarket transaction, even though it appears that way sometimes), of which an example is researched in paper [3].

The authors look at event sequences, which are sequences of events, within a time frame. An interesting *episode* is a series of events that happens within a *time window*. First they compute all frequent episodes, which are episodes with a minimal occurrence in a time window. Then, they compute all the rules using that set, where an episode rule is an implication $\beta \Rightarrow \gamma$, where β and γ are episodes where β is a subepisode of γ. Doesn't this sound familiar?

This shows that pattern set mining can be used in broad area by adapting the techniques accordingly.

3 Constraints

In this section, we will dive into the world of constraints. By formulating what patterns you would want, you can limit the number of patterns returned.

3.1 FP-growth algorithm

In this section, the FP-growth [4] algorithm to mine the complete set of frequent patterns will be explained. The improvements of FP-growth over Apriori include no need for generating candidate itemsets and no need for multiple scans over the database to check the support of each itemset.

First, we discuss the proposed storage structure. Each transaction is represented by the set of frequent itemsets included, which is ordered on decreasing support. The concise representation as frequent pattern (FP) tree is based on the fact that transactions will often share the prefix. Itemsets that occur most frequently will appear at the beginning of the prefixes of many transactions, and thus they will have a shared prefix.

Second, we will explain how the frequent patterns are obtained. The search through the tree is very elegant and fast, by node-links that can identify an associated frequent itemset quickly. It applies a pattern growth method, such that you can concatenate frequent 1-itemsets, without generating useless candidate sets.

3.2 Push constraints into FP-growth

Because of runtime issues, it would be better to apply constraints directly instead of using a filter after having computed all patterns. This is is called *constraint pushing.* In [5], they propose a method to push anti-monotone constraints into the FP-growth algorithm.

Constraints are anti-monotone if and only if whenever an itemset S violates the constraints, so does any superset of S. A well-known example is the constraint being frequent: If an itemset is not frequent, nor will any of its supersets! This saves a lot of work in searching. The beauty of the proposal in this article, is that more constraints (*convertible anti-monotone constraints*) can be converted to a anti-monotone constraint using a specific order on the items. Consider a harder constraint than frequency, such as the $avg(S) \leq v$, where each item in S is labeled with a value. If the items are sorted in decreasing order, the average of an itemset is not more than of its prefix (try to think of a counter-example; there is none). These ideas can be readily incorporated into FP-growth, the previously explained algorithm, where experimental results showed the effectiveness of the algorithm developed.

3.3 ExAnte

In contrast with the previous article, the ExAnte algorithm tackles the problem of dealing with monotone constraints.

If a monotone constraint holds for itemset X, then it holds for any superset of X. For a long time, this seemed to be harder to deal with than anti-monotone constraints, since you need to look until the last superset of a superset to establish whether or not the constraints holds, even if you encounter itemsets for which is does not hold. But, in [6], a preprocessing algorithm is introduced that can be coupled with any constraint pattern mining algorithm. The constraints prune the search space and the input database, while containing the exact support of each solution itemset. Two reductions are used. The $\alpha-$reduction prunes the items in X that do not satisfy the frequency constraint, and removes them from the corresponding transactions. Then, there are more opportunities for $\mu-$reduction, which prunes the transactions that do not satisfy the given monotone constraint. By removing those transactions, some items may have become infrequent, which induces more opportunities for an $\alpha-$reduction. The $\alpha-$reduction and $\mu-$reduction are called alternatively until neither of them can be applied anymore; then they are passed on a main algorithm.

It has been theoretically proven that it is always profitable to start any constrained pattern computation with the ExAnte algorithm, which is a pretty cool result.

3.4 Pushing constraints?

The previous section already showed a way to avoid filtering, without pushing constraints. [7] argues that pushing constraints should not be used at all, for it would be outdated. Instead, the authors propose, why not produce all the patterns? This is a costly process, they also admit, but it only needs to happen once. After that, the user can do query mining on this initial result set. In other words, use the data with the weakest constraints for exploration. But, we need to keep in mind, even if producing all the patterns once will only take time once, the problem of the space it takes is not taken into account.

To conclude the short review on (pushing) constraints, not much is written about it currently in the academic site. Let's move on to the next approach.

4 Condensed representation

For your quest, a colleague tipped you to look at condensed representations. What if you could think of ways to subset the set of all frequent itemsets in such a way that all the information is there?We will take a look at closed itemsets and non-derivable itemsets in the following two sections.

4.1 Closed itemsets

In [8], Pasquier et al. introduce a new algorithm *Close* for mining association rules in very large databases, based on the pruning of the closed itemset lattice.

A itemset S is *closed* if there exists no superset that has the same support as S. For example, if item A always co-occurs with item B, then A and B are not closed, since the itemset AB has the same support. In the paper, the authors constructed a wonderful proof that stated that the set of maximal frequent itemsets M is identical to the set of maximal *closed* itemsets MC. This is awesome! If we want to find the set of all frequent itemsets, we only have to look at the set of all closed itemsets. Note, from the set of maximal frequent itemsets, the set of all frequent itemsets can be easily derived by taking all the subsets of the found itemsets.

Close works as following. First, the set of frequent closed itemsets is obtained. This can be done in the same sort of level search as Apriori. This is the case, since all sub-closed itemsets (closed subsets) of a frequent closed itemset are frequent. Next to that, all sup-closed itemsets (closed supersets) of an infrequent closed itemset are infrequent. Finally, the support of an itemset I is equal to the support of the smallest closed itemset containing I. This allows us to use the same threshold minSup in the set of closed itemsets as we would in the complete data set.

Second, we can construct the complete set of frequent itemsets, and derive their supports. Then, the generation of association rules can be done efficiently in a straightforward manner.

4.2 Non-derivable itemsets

The main goal of paper [9] is to present several methods to identify redundancies in the set of all frequent itemsets and to exploit these redundancies, resulting in a condensed representation of all frequent itemsets and significant performance improvements of a mining operation. The authors give several deduction rules to deduce the lower bound and upper bound of the support for the itemsets. The bounds they give are proven to be non-redundant, which means that any omission of a rule would give a less tight interval. Moreover, the rules are complete, which means that the derived bounds are always tight. This results in the set of non-derivable itemsets (NDI), which is the set of itemsets whose supports cannot be derived. They prove that the set of frequent NDIs allows to compute the support of all other frequent itemsets, so NDIs can be used as a condensed representation. A nice property of NDIs is that their size is bounded by the logarithm of the database size, which means that the compression is high. The authors provide an efficient algorithm to derive all frequent itemsets.

5 Redefine interestingness

Still, many patterns are returned with a condensed representation. Moreover, would your boss be able to interpret the information hidden, or would we need to reconstruct everything before we can make use of it? Making sense of the database would then again require work and time. We go further. Remember, your boss is looking for interesting patterns in the data. But what is “interesting? In the following sections we redefine this notion and explore the possibilities.

5.1 Miki

In the end, a set of patterns is returned. This set might contain individually very interesting patterns, but the quality of patterns should also be considered in the context of the other patterns found. In paper [10], the authors introduce a new framework for mining binary data. Items are selected by their distinctive power, relative to other items. The quality of itemsets is measured by the use of *joint entropy*. Joint entropy is a measure for the amount of information conveyed by an itemset. Assume a itemset with just one item a. If item a is in (almost) all transactions in the database, item a is not very informative: the chance that this item is in a transaction is very high. Then the joint entropy will be low. What if a almost never occurs in the database? Then again the joint entropy is low. But, the joint entropy of an itemset is high if a occurs in half of the transactions, but in the other half it doesn't: then it can be of distinguishable value in a database! An itemset of size k that induces the highest joint entropy compared to all other itemsets of size k is called a *maximally informative k-itemset* (miki). Miki's can be used as a means of filtering results obtained by rule discovery. In the next section, different intuitions and according measures are discussed as filtering methods.

5.2 Pattern teams

In [11], the authors propose filtering the returned set of patterns based on a number of quality measures. The small subset that optimizes such a measure is called a *pattern team.*

What quality measures should be used? A myriad of intuitions is used. An example is (approximate) mutual exclusivity, which means the patterns should not describe the same subset of items. Moreover, no patterns should cover (approximately) the same set of examples. Also, no patterns should cover (approximately) the complement of another pattern. Did you note something odd? The first and the last intuition are actually competing to some extent! Therefore, the user is usually only interested in one or a few intuitions. Several measures are compared, including the previously introduced joint entropy and exclusive coverage, where patterns sets that reduce the amount overlap between patterns are favored. What measure is best for what application, is dependent on the context, aim and underlying intuitions.

5.3 Chosen few

In [12], the author propose a general heuristic approach for selecting a small subset of patterns, a similar goal as the previous section. That set of patterns should be small enough to be easily processed, and it should show little redundancy while retaining as much information as possible encoded in the full pattern set. In machine learning algorithms, a subset $\mathcal{S}^* \subset \mathcal{S}$ that induces the same partition for the complete set $\mathcal{S}$ will be of the same usefulness, is the intuition. The advantage is that a human can then far more easily process the small subset (with size around 30). The idea of the algorithm is as following. Given a set of patterns, one can (binary) point out which transactions contain a pattern, and which don't. In this way, a pattern partitions the transactions. From the two subgroups, the next patterns can again split up the subgroups in more subgroups. Note that not each pattern will split up each subgroup in two. If there is no change, then the pattern is rejected and not included in the subset. The ordering of the processing of the patterns works best by support descending or length ascending.

This technique is a post-processing step, used to reduce the number of patterns without losing information with respect to the partitioning power.

5.4 Tiling

Tiling databases [13] is a new way of looking at characterizing and mining a database, just like frequency or constraints rules. While there are too many frequent itemsets returned as results, there are less results for tiles.

A *tile* consists of a block of ones in a 0/1 database. Frequent itemsets do not necessarily tell you something about the database. Using area of ones as measure, you balance the number of items (not too low) and the number of transactions (not too low). A (large) tile contains information about the database. If you only know there are n possible items and m rows, there are 2^{nm} possibilities to fill the database. A tiling implicitly determines an upper bound on the number of different databases that are consistent. A complete tiling covers exactly the database, and it can be seen as a form of a (condensed) representation.

Several quests in the context of tiling can be formed, for example the *maximum k-tiling problem*, which asks for a tiling consisting of at most k tiles that have the largest possible area, or the *minimum tiling problem* which asks for a tiling with minimum number of tiles that covers all ones in the database.

In [13], the authors proof the minimum $k-$tiling and maximum k-tiling problem are NP-hard. They provide a greedy approximation algorithm for $k-$tiling that ensures an approximation within a constant factor of the optimal solution, which is really nice.

5.5 Using sampling

In the next few sections, we will move on to a different way to determine the interestingness of a set of patterns, namely using statistics.

Instead of exhaustively search the pattern space (and filter it), algorithms exist that are based on Markov Chain Monte Carlo methods (MCMC). A downside of this is the so called "burn in period", that usually, MCMC needs to run for a long time before it approximates the real distribution accurately. In paper [14], the proposed procedures produce exactly k patterns proportional to either frequency, squared frequency, area or discriminativity. We will shortly discuss the frequency-based sampling. First, select a transaction proportional to the size of its powerset. Then return a uniformly sampled subset of that transaction. The intuition is that a random transaction is likely to contain a pattern that is supported by many transactions. Using the size of the powerset makes sure that the sampling is not biased towards small transactions.

We summarize [15] that introduces the so called *FastEst* algorithm that estimates the number of patterns for a given threshold minSup. Consider a tree T rooted at $\emptyset$, where each node is a frequent itemset. The children of a node X are all of its frequent expansions $X \cup u$, with u some item not in X. The leaves correspond to the maximal frequent itemsets we came across before. The tree does not need to be materialized; instead paths will be sampled from it. A sampled path is constructed by "walking" through the tree until it hits a maximal frequent itemset. Then it will return the sequence of encountered out-degrees. The average of the sampled paths is the final estimate. Moreover, the *pattern frequency spectrum* gives an estimate for the total number of patterns for all possible frequency thresholds. It can be obtained by a number of estimates for random values of thresholds using the FastEst algorithm, and then fit a nonlinear regression line through these.

5.6 Self-sufficient itemsets

Instead of searching for interesting association rules, paper [16] looks for interesting itemsets. If two products co-occur more often than expected, than two rules will be found, both *hand gel* $\rightarrow$ *face masks* and *face masks* $\rightarrow$ *hand gel*. Then, the essential discovery is {*face masks, hand gel*}, the authors argue. In the paper, interestingness of itemsets is defined as *self-sufficiency*. Itemsets are self-sufficient if their frequency is greater than can be accounted for by either the frequency of their subsets or of their supersets alone. But when is that?

We define an itemset *productive* if the frequency of the itemset is greater than that which would be expected from any partition of the items into two independent itemsets. Suppose item a, b and c are independent. Then, $supp(\{a,\ b\}) = supp(\{a\})supp(\{b\})$, so $supp(\{a,b,c\})= supp(\{a,b\}supp(\{c\})$. Hence, $\{a,b,c\}$ is not productive. We need to add another criterion, namely the itemset should not be redundant. The transactions that contain the item *mother* will all also contain the item *female*. We know *female* is a generalization of *mother*. An itemset S is redundant if there is a proper subset R of S that has identical support to one $R's$ subsets.

The last extra requirement we set is that an itemset S should also be productive to its exclusive domain. Its exclusive domain is the set of transactions that is not covered by any productive non-redundant superset of S. Then we would find that superset (instead of S) more interesting. The supposed interestingness of the subsets would then be caused or explained by its superset.

A note of caution, this is not a condensed representation, since it does not contain enough information to derive the support of all frequent itemsets.

5.7 Statistical significance testing

In paper [17], the authors propose a very general approach to find the smallest set of patterns that captures what is statistically significant about the data, in terms of a global p value. The proposed framework is shown to be applicable in a large area, including time series segmentation, clustering and frequent itemset mining.

Given a null model, a set of predefined patterns, and a test statistics, the goal is to identify the smallest set of patterns that, when imposed as constraints to the null model, leads to the data no longer being significant. Each time a pattern is added as a constraint, the space of possible binary databases shrinks (remember, we have seen this concept before in section 5.4), so then the constraints contain more information. The null hypothesis is defined by a probability function over the sample space, a "neutral" null reference.

First they show that the problem of finding a set of size K that maximizes the p-value is of serious complexity. It is NP-complete, and no algorithm can approximate it by a finite ratio. Anything else than exhaustive search may be arbitrarily far from the optimum. Next, they provide a greedy algorithm for the problem, with a practically small (20) number of returned itemsets in experiments.

5.8 Statistical Modelling

In the previous section, a given null model was used as a reference model against which the interestingness or unexpectedness of patterns was contrasted. But what could we use as a null model? The following two papers integrate prior knowledge to serve as model. The challenge here is to formalize this prior information in a way that it can be used to define subjective interestingness in a meaningful and practical way.

The prior information is considered as constraints on the probabilistic null model, the maximum entropy [18] principle is used for this. The problem is finding a probability distribution that satisfies a set of linear constraints implied by prior information. These will not be sufficient to uniquely determine a distribution. Therefore, a common strategy is to search for the distribution with the largest entropy subject to these constraints, called the *MaxEnt distribution*. The MaxEnt distribution makes the fewest assumptions about the true distribution of the data, and is therefore the safest bet to represent the prior knowledge. Besides that, it is relatively easy to compute the MaxEnt model.

Subjective interestingness of a pattern can be quantified by contrasting it with the prior information in the form of a MaxEnt distribution. You would want to compute some measure of unexpectedness of the pattern. An example is *self-information*, which relies on the probability of a pattern under MaxEnt model. The smaller this probability, the more surprising the pattern (you can take for example the $-\log$ of the probability). Self-information does not take the complexity of the communication of a pattern into account. Therefore, also the *information compression ratio* could be used as measure, which represents how much information is compressed in a pattern. Or, the *p-value*, as has been introduced in the previous section, with the MaxEnt distribution as reference model.

Much research tries to objective interestingness. This is odd, since an 'interesting' projection is not easily universally agreed on. Paper [19] explores the problem where the user is interested in exploring without a clear anticipation of what to expect or what to do with the patterns found. Can we develop a mathematical method to rate the patterns based on some user criteria, so that the patterns that should be interesting for the user will be more likely to be returned?

A user cannot define all prior believes at once. Instead, the user can incrementally update its prior beliefs. The goal should be to pick those patterns that will result in the best updates of the user's belief state, with a small descriptional complexity.

5.9 Krimp

The main reasons for mining a data set, is to gain insight. Therefore, the goal of paper [20] is to find the set of patterns that describes the data the best. To make this precise, the Minimal Description Length (MDL) principle is used. The MDL principle provides a way to balance the complexities of the compressed database and the encoding. If the encoding is too simple, the database it hardly compressed, but if the encoding is too complex, this is also not desirable. The best model is $H \in \mathcal{H}$ minimizes $L(H) + L(\mathcal{D}|H)$, where $L(H)$ the length of the description of H is, and $L(\mathcal{D}|H)$ the description of the data when encoded with H. How does the MDL principle work for itemsets? The idea is to use a code table, which is a two-column table with itemsets and their corresponding code. Transactions can be covered by a disjoint set of itemsets from the code table, so a transaction can be encoded by the codes of its cover. They are covered according to the *Standard Cover Order*, first by decreasing itemset size, second by decreasing support and third lexicographically.

If the codes are chosen wisely, the database can be compressed much. $\text{usage}_D(X|CT)$ denotes the usage of itemset X in D given CT. We consider relative usages:

$$\mathbb{P}_D(X \mid CT) = \frac{\text{usage}_D(X \mid CT)}{\sum_{Y \in CT} \text{usage}_D(Y \mid CT)} \quad (1)$$

The intuition that we should use shorter codes the more an itemset is used, expresses that the length of X is $-\log(\mathbb{P}_D(X|CT))$. From now on, we assume all code tables are code-optimal.

Also the choice of the itemsets for in the code table, the coding set, is important. The problem can be defined as to find the smallest coding set CS such that the total compressed size of the database $L(\mathcal{D}, CT)$ with the corresponding code table CT is minimal. The proposed Krimp algorithm works as following.

First, you start with standard code table ST containing only singleton itemsets. How to find the optimal? Trying all options is simply not viable. Therefore, the algorithm makes use of a heuristic.

Add the candidate itemset one by one, by the *Standard Candidate Order*. This is an order first decreasing on support, second descending on the size of the itemset, and third lexicographically. The transactions are covered using the *Standard Cover order*. A candidate itemset is kept only if the encoding results in a smaller compressed size.

In addition, pruning can be used. When a candidate code table is accepted, we consider all its valid subsets for pruning. Only a few candidate subsets are accepted, so not the whole pruning search space needs to be considered. Also, only the itemsets that have an increased usage are considered for removal.

Krimp was tested as classifier to value the interestingness of the found set, which gave good results. Also the reduction on the number of returned sets were very promising.

The advantages of the approach include that Krimp is lossless, shows to be noise resistant and does not need user-defined parameters.

Second part

"All human knowledge is uncertain, inexact and partial," Bertrand Russell once said. When we try to make sense of the world around us, uncertainty is part of that process. In general, the data in databases is assumed to be certain. This might be realistic for the modelling in supermarkets (an item is there or not). But, sometimes this is not the case. In some databases, noisy data is present, such as the data collected by satellite images. Next to that, uncertainty can be used to protect the privacy of users, when artificial noise is added. It is then more challenging to find patterns. Moreover, transactions can be aggregated per customer. Probabilities express the estimated likelihood that a customer purchases an item. Then patterns can be mined across customers instead of transactions.

In the previous part, we discussed the problem of (deterministic) pattern set mining and discussed solutions proposed over the years. In this part, we will introduce the problem of probabilistic pattern set mining, and discuss some approaches to this problem. Probabilistic or not, we still aim to gain insight in the data.

First, we will discuss how to mine probabilistic frequent itemsets using the methods of [21]. Then, we shortly address the problem of finding probabilistic association rules following the idea of [22]. Finally, we explore characterizing uncertain databases using compressing, according to the approach of [23].

Definitions

Uncertainty in a database can be expressed in a myriad of different ways. In section 6 and 8, we look at databases where a probability is attached to whether an item i is present in transaction j. However, the article in section 7 assumed an existential probability for each transaction, which specifies the chance that the transaction exists. Note that the probabilities are all assumed to be independent. These approaches are different (one cannot express one in terms of the other). Next to these, another type of uncertainty is attribute-level uncertainty. Instead of an item being present (1) or absent (0), each attribute can have n different values, but we are not sure which one. We will not address attribute-level uncertainty further.

All articles adopt the widely used *possible world semantics*. Conceptually, an uncertain database can be viewed as a set of possible deterministic instances (called *possible worlds*). Reality is one of the possible worlds, but our observations of it contain uncertainty reflected by the probabilities in probabilistic database P. There exist $2^{n \times m}$ possible worlds in a database with n transactions and m items.

6 Probabilistic Frequent Itemset Mining

A frequent itemset has a support higher than a threshold minSup. However, in probabilistic databases the support of an itemset is uncertain. Therefore we define the frequentness probability, that is the probability that an itemset is frequent. A probabilistic frequent itemset (p-FI in short) is an itemset with a frequentness probability of at least τ.

Some discussed previous work has been based on the expected support of an itemset. The expected support $E[supp_{\mathcal{P}}(X)]$ of an itemset X in probabilistic database $\mathcal{P}$ is the sum of existential probabilities of X, defined as

$$E[supp_{\mathcal{P}}(X) = \sum_{t \in \mathcal{P}} P(X \subseteq t) \tag{2}$$

Here, $P(X \subseteq t)$ is the existential probability that transaction t contains itemset X.

The problem with this is that two itemsets with the same expected support can have very different frequentness probabilities. Let $t_1 = \{\{A, 0.5\}; \{B, 1.0\}\}$ and $t_2 = \{A, 0.5\}$. Suppose minSup = 1. The expected support of both item A and B is 1. But the frequentness probability of A is 0.75, while the frequentness probability of B is 1. We can be certain that B is frequent, in contrast to A. Frequentness probability can differentiate between those two.

The authors propose an algorithm to find all itemsets that are frequent with a frequentness probability of at least τ. To do this, they first present a method for computing the frequentness probability in $O(|T|)$ time, with $|T|$ the number of transactions. They propose an algorithm based on Apriori, and an additional algorithm that outputs the uncertain itemsets in order of decreasing frequentness probability.

We first need to calculate support probability distribution called the *probabilistic support* (note, this is not the expected support). $P_i(X)$ denotes the probability that itemset X has support i. The naive implementation would materialize all possible worlds and count their support. But this takes too long, and is not necessary. It can also be computed directly, as proposed by the authors. The frequentness probability of X can be expressed as $P_{\geq \text{minSup}}(X)$. Nevertheless, the complexity of computing this is still exponential w.r.t. the number of transaction $|T|$, so we will not go into this basic algorithm in detail. Because, the authors propose to use dynamic programming to reduce this to linear time.

Dynamic programming splits up the problem in sub-problems, solves them in a smart order and combines it efficiently. Let $P_{\geq i,j}(X)$ denote the probability that at least i of j transactions contain itemset X, which is

$$P_{\geq i,j}(X) = \sum_{S \subseteq T_j : S \geq i} \Big(\prod_{t \in S} P(X \subseteq t) \cdot \prod_{t \in T_j - S} \big(1 - P(X \subseteq t)\big)\Big) \tag{3}$$

The summation is over all subsets S of T_j with size larger than i, where T_j is the set of the first j transactions. The product is taken over all transactions t in S. Because all the probabilities are assumed to be independent, we multiply the probability $P(X \subseteq t)$, that X is contained in transaction t (this is in the data), with the complement of that probability $1 - P(X \subseteq t)$. Note, $P_{\geq i,|T|}(X)$ denotes the probability that X is contained in at least i transactions of the entire database.

Dynamic programming can make use of recursion, splitting up the problem, as we mentioned before. We construct a matrix with the support on the y-axis, and the number of transactions on the x-axis. Each cell (i, j) represents the value of $P_{\geq i,j}$. The recursive formula is given by

$$\begin{aligned} P_{\geq i,j}(X) = \\ & P_{\geq i-1,j-1}(X) \cdot P(X \subseteq t_j) + \\ & P_{\geq i,j-1}(X) \cdot P(1 - P(X \subseteq t_j)) \end{aligned} \tag{4}$$

Here, $P_{\geq 0,j} = 1 \ \forall .0 \leq j \leq |T|$, because it is certain that an itemsets is contained at least 0 transactions. We also know that $P_{\geq i,j}(X) = 0 \ \forall .i \leq j$, because in the first j transactions, X can be contained in at most j transactions. A visualization of the matrix can be seen in figure 1.

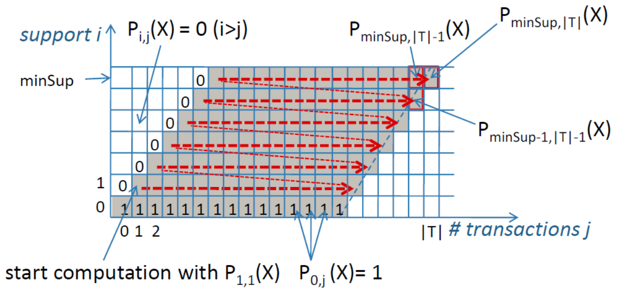

Figure 1: [21] A visualization of the DP matrix

Here can be seen that once the cell to left $[P_{\geq i,j-1}(X)]$ and the cell to the lower left $[P_{\geq i-1,j-1}(X)]$ is known, then $P_{\geq i,j}(X)$ can be computed. The authors proof that the running time is $O(|T|)$ and the required space can also be reduced to $O(|T|)$.

Filter strategies

In addition, filter strategies can be used. We can use the monotonicity lemma that $P_{\geq i,j} \geq P_{\geq i+1,j+1}$. In the best case, the added transaction $j + 1$ will certainly contain the corresponding itemset, which makes $P_{\geq i,j} = P_{\geq i+1,j+1}$. In all other cases, the probability will be less. Therefore, we can prune X if any of the probabilities $P_{\text{minSup}-k,|T|-k}(X)$, with $1 \leq k \leq |T|$ is lower than the user specified threshold τ.

Next to that, 0-1 optimization can be used. If $P(X \subseteq t_j) = 0$, transaction t_j can be ignored. If a transaction t_j contains X with probability 1, t_j can be omitted if we decrease minSup by 1. This speeds up the computation if the database contains certain items (0/1).

Algorithms

The first proposed algorithm is based on the Apriori algorithm. Frequentness probability is anti-monotone, which means that all subsets of a p-FI are also p-FIs. This can be used to adjust the Apriori algorithm as following. Each iteration is performed in two steps: a join step for generating new candidates and a pruning step with calculating the frequentness probabilities and extracting the p-FIs from the candidates.

The second proposed algorithm is called incremental probabilistic FIM algorithm. The handy aspect of this algorithm is that the parameter τ is not necessary any more, since the algorithm returns the results in order of decreasing frequentness probability. The itemsets can be returned one by one (incremental ranking queries), or be used to find the k itemsets that have the highest frequentness probability (top-k p-FIs query). The algorithm works as following. First, all the items are added to the Active Itemset Queue (AIQ) in decreasing order of frequentness probability. The first itemset X of the AIQ (the most probable) is taken out and outputted. Then, the 1-extensions of X are considered, and added to the AIQ with their respective frequentness probabilities. All of X's supersets cannot have a higher frequentness probability, due the anti-monotonistic property. Continuing to retrieve the next first itemset from the AIQ can be done until k is reached or all itemsets are returned.

In conclusion, in [21], a method is proposed to find itemsets in an uncertain transaction database that are likely to be frequent. Using dynamic programming techniques, the frequentness probability can be computed in linear time. Moreover, filtering techniques were used to speed up the computation. Then, an implementation based on Apriori was discussed, and an iterative itemset mining framework was introduced which outputted the most likely frequent itemsets first.

7 Probabilistic association rules

In [22], the authors study the discovery of frequent patterns and association rules from probabilistic data under the possible world semantics. The authors assume that each transaction has an existence probability. As has been said before, this is different from the previous article, and one cannot be trivially converted into the other. The article first discusses an algorithm to mine frequent patterns (itemsets), but we will not go into this. At this point, we are more interested in their ideas to mine probabilistic association rules.

A deterministic association rule has been defined in the previous part. An association rule is an implication in the form $X \Rightarrow Y$, which has a support *supp(XY)* and a confidence *c*. $X \Rightarrow Y$ is a probabilistic association rule (p-AR in short) if

$$P(X \Rightarrow Y) \geq \text{minProb} \tag{5}$$

So the probability $X \Rightarrow Y$ should be larger than a threshold *minProb*. This probability is given by

$$P(X \Rightarrow Y) = \\ P\big[\mathit{supp(XY)} \geq \text{minSup} \wedge \mathit{conf}(X \Rightarrow Y) \geq \text{minConf}\big] \tag{6}$$

Given a minSup, minConf, minProb and the set of probabilistic frequent patterns, the problem can be defined to find all p-AR and their probabilities. A naive way of solving this, would be to materialize all possible worlds, determine the association rules per world, and compute the probability of each rule over all the worlds. This is of course not viable.

Before we can move on, we need to be able to compute the probability of a p-AR efficiently. Let f_X denote the support probability distribution of X, which denotes the probability that X has support i for $0 \leq i \leq |T|$. Given that X and XY are probabilistic frequent itemsets, and given f_X and f_{XY}, we can efficiently evaluate $P(X \Rightarrow Y)$.

First, find $f_{X\overline{Y}}$. Because $\mathit{supp}(X) = \mathit{supp}(XY) + \mathit{supp}(X\overline{Y})$, f_X is the convolution of f_{XY} and $f_{X\overline{Y}}$. The other way around, $f_{X\overline{Y}}$ is the deconvolution of f_X and f_{XY}. We can use some math magic called the Fast Fourier method (we will not go into detail here), and implement the computation of the probability of a p-AR in $O(n \log n)$. For the details, we direct you to [21].

At this point, we can use the anti-monotonicity trick again. Let X be a p-FI, and X' and X'' non-empty itemsets, with $X'' \subset X' \subset X$. Then, if $X - X' \Rightarrow X'$ is a p-AR, then also is $X - X'' \Rightarrow X''$.

The anti-monotonicity trick is married with Apriori, so we adopt that framework. Let X be a p-FI, and X_i a sub-pattern of X of size i. We check whether $X - X_1 \Rightarrow X_1$ is a p-AR for all X_1s. If it is not, we can stop examining all rules with a superset of X_1 as the consequent. Otherwise, we check the validity of $X - X_2 \Rightarrow X_2$ for all supersets X_2. We continue this checking until a rule is encountered that is not a p-AR, or until $X_i = X$.

On our way

The last two sections both showed nice results. However, some of the problems of the first part of the essay are again prevalent here, for example the pattern explosion, the questioned definition of “interesting”, and the overlap and redundancy in the set of itemsets returned. If all probabilistic frequent itemsets or all probabilistic association rules are returned, will we gain any knowledge or insight in the database, other than that there exist many? Also, because the algorithm is based on Apriori, it suffers from some of the same caveats, such as generating useless candidates.

To continue our quest, we will go and discuss the next article.

8 Characterizing using Compression

Now, we have come to the moment supreme (maybe dependent on what you were hoping for in the first place, but still), where we look at the value or interestingness of the whole set returned. In contrast, the previous two articles looked at all interesting (frequent) patterns and all probabilistic association rules. In [23], the authors study the problem of discovering characterizing patterns in uncertain data through information theoretic lenses. Using the possible worlds semantics and the MDL principle (remember from section 5.9 in the previous part?), the aim is to discover patterns that compress well in expectation.

In this setting, a probabilistic transactional database $\mathcal{P}$ attaches a probability p_{ij} to indicate the probability that item i is present in transaction j. Reality is one of the possible worlds, but our observations of it contain uncertainty reflected by the probabilities in probabilistic database $\mathcal{P}$. The frequency of a pattern X is defined as the expected support over all the possible worlds. This is different from the approach in [21] as explained in section 6. Then, we wanted to know how often the support exceeded a threshold minSup to be frequent. Here, we would like to find patterns that compress well in *expectation.*

Recall the minimum description length (MDL) that can be used to find code tables on deterministic data. We would like to find the best model $H \in \mathcal{H}$ for database D, that minimizes $L(H) + L(D|H)$, where $L(H)$ is the length of the description of H, and $L(D|H)$ the length of the description of the data in D when encoded with H. In the context of probabilistic databases, we need to reformulate the problem to finding the code table CT with *minimum expected coding length.*

Analogous to the deterministic case, we construct the probability distribution on itemsets in the code table by normalizing, here using expected usages:

$$\mathbb{P}_{\mathcal{P}}(X \mid CT) = \frac{E[\text{usage}_D(X \mid CT)]}{\sum_{Y \in CT} E[\text{usage}_D(Y \mid CT)]} \quad (7)$$

The optimal code length can defined as $L(\mathit{code}_{CT}(X)) = -\log(\mathbb{P}_P(X \mid CT))$. Itemsets that are expected to be used often are assigned short codes. From now on, if we talk about a code table, we assume that we use these optimal code lengths.

The aim is to find the code table CT that tries to minimize $E[L(D, CT)] = E[L(D|CT)] + L(CT)$. As always, the naive approach would be to enumerate all possible worlds of $\mathcal{P}$, but we will put this approach quickly aside again.

Exact computation

For databases with a small number of items, we can compute $E[L(D|CT)]$ exact by materializing a sort of "expected database", we will call this method MED (Materialize Expectation in Data). Minimizing the expected coding length of D is equivalent to finding the optimal code table on the deterministic matrix $M(\mathcal{P})$, if we keep track of the weights of its rows that are their expected counts $E[\text{count}_D(t)]$. We can compute the expected count $E[\text{count}_D(t)]$ of a transaction t without summing over all possible worlds. Using $E[\text{count}_D(t)]$, we can compute the $E(L(D|CT)]$ and the expected usages directly on the matrix $M(\mathcal{P})$. For the details of the proof, we direct you to [23].

So, we materialize all possible transactions and compute the expected count in $\mathcal{P}$ for every such vector, and can then apply it to the standard Krimp algorithm. We can materialize the matrix $M(\mathcal{P})$ in $O(2^{m+1}n)$ time, with m the number of items and n the number of transactions. It is clear this quickly becomes infeasible as m grows. Therefore, we then will resort to the next approach: sampling.

Sampling

For larger databases, we approximate $E[L(D|CT)]$ by sampling. A basic sampling approach would be sample k possible worlds from $\mathcal{P}$ according to their existential probabilities, and then concatenate them into one database D^k. Too simple? Maybe not. The authors provide a proof on the bounds of the estimation, which we will shortly discuss here.

First, observe that the sample mean based estimator for $E[L(D|CT)]$ is defined as

$$\hat{E}[L(D|CT)] = \frac{1}{k}\sum_{i=1}^{k} L(D_i|CT) \quad (8)$$

This formula doesn't need scare you too much, since it is just the average over the k sampled worlds. Since $L(D|CT) = \sum_{t \in D} L(t|CT)$, we can rewrite it as

$$\hat{E}[L(D|CT)] = \frac{1}{k}\sum_{t \in D^k} L(t|CT) \quad (9)$$

This sums all the code lengths of the transactions in D^k and averages over the k possible worlds. If these code lengths $L(t|CT)$ would be accurate, we would obtain an accurate estimate of $E[L(D|CT)]$. $L(t|CT)$ is the sum of the code lengths of the codes that belong to the cover of transaction t, denoted by *cover(CT, t)*. So, the problem has shifted to determining (approximately) accurate code lengths. Recall that $L(\mathit{code}_{CT}(X)) = -\log(\mathbb{P}_P(X))$, and that $\mathbb{P}_P(X)$ and thus the code lengths are dependent on the expected usage of X and the expected usages of all other itemsets in CT.

Just for presentation, let μ_X and $\hat{\mu}_X$ denote $E[\text{usage}_D(X|CT)]$ and $\hat{E}[\text{usage}_D(X|CT)]$. Γ and $\hat{\Gamma}$ denote respectively the sums $\sum_{Y \in CT} \mu_Y$ and $\sum_{Y \in CT} \hat{\mu}_Y$. We can rewrite equation 7 as

$$\mathbb{P}_{\mathcal{P}}(X) = \frac{\mu_X}{\Gamma} \quad (10)$$

We consider sample mean based estimators and let

$$\hat{\mu}_X = \sum_{t \in D^k} \frac{\mathbb{I}\{X \in \mathit{cover(CT,\ t)}\}}{k} \quad (11)$$

$$\hat{\Gamma} = \sum_{Y \in CT} \sum_{t \in D^k} \frac{\mathbb{I}\{Y \in \mathit{cover(CT,\ t)}\}}{k} \quad (12)$$

Again, no need to be scared of these formulas! $\hat{\mu}_X$ is the expected usage of X, and the formula above just counts how often X used in the cover of any of the transaction of k. $\mathbb{I}$ is the indicator function, which is 1 if X is in the cover, and 0 otherwise. Because we sample k times, the sum is divided by k. $\hat{\Gamma}$ sums over all itemsets Y in CT to add up $\hat{\mu}_Y$.

Okay, this is the moment we will do some math magic again, which means formally that we use *Hoeffding's inequality bound.* Using this, we can bound the difference between the real mean and the sampled mean. We have

$$\Pr(|\mu_X - \hat{\mu}_X| \geq \epsilon n) \leq 2e^{-2nk\epsilon^2} \quad (13)$$

and

$$\Pr(|\Gamma - \hat{\Gamma}| \geq \Gamma S \epsilon) \leq 2e^{-2nk\epsilon^2} \quad (14)$$

for a small ϵ, with k the number of samples, n the number of transactions, and S the maximum number of items in a sampled transaction.

Wow, this is awesome! This means we can say something about the performance of our sampling method. And not just something. The main result is that for itemsets X with a reasonably high expected usage ($\mu_X \geq n/c$), the estimator $\mathbb{P}_{\mathcal{P}}(X)$ is concentrated around its expectation with high probability.

Algorithms

The first algorithm MED is only viable for small databases, as already had been explained. After constructing the deterministic matrix $M(\mathcal{P})$, the expected optimal code table can be easily approximated by applying Krimp. This method is the most accurate, since the only approximation is caused by the heuristics of Krimp. The downside is that it has the largest memory requirements, and is infeasible for large m.

The second algorithm is called "Sample, Merge and Mine" (SMM), based on the sampling approach explained before. After sampling k worlds and merging then, Krimp can be applied. The found compressed size needs to be adjusted by a factor $\frac{1}{k}$ to account for the k samples.

The third algorithm is called is called "Sample & Mine Until Convergence" (SMUC). It is parameter free and does not need to maintain all sampled worlds. This last method is the least accurate of the three, but requires also the least memory. It starts with an empty code table, and continues sampling till the code table no longer changes. It works as follows.

SMUC first computes the expected standard code table, which contains all singleton items. Then, a world is sampled and frequent itemsets are mined from this world (note, these are deterministic). The found frequent itemsets are added one by one using the standard candidate order of Krimp. If the compression is improved, the new CT table is adopted. This repeats until the CT table does not change anymore and is converged.

The previously seen worlds do not need to be saved explicitly. Instead, the usages aggregated over all previously seen worlds are maintained in U, for each itemset in CT separately. For each sampled world D, for each itemset X in CT, U is updated $U[X] = U[X] + \text{usage}_D(X)$. Now, the lossless compression over all worlds can be ensured, by defining

$$L(CT, U, D) = L(CT) + \frac{L(U \mid CT) + L(D \mid CT)}{k} \tag{15}$$

In addition, pruning can be used. If we want to test if removing an itemset would improve the expected coding length, we would want to compute a new cover of the data over all sampled worlds. But we do not maintain them, so that is not possible. The solution is simple: we remove itemset X and cover X using the current CT as many times as its previous usage, and adjust the usages of other itemsets accordingly. The pruning is accepted if it improves the expected coding length. Post-accepted pruning considers all itemsets for removal one by one, and can be used for both MED, SMM and SMUC. Pre-accepted pruning considers all supersets of a candidate itemset for removal, for these (more specific) supersets may be in the way of (better) generic candidates. Pre-accepted pruning is only used with SMUC, since it would not make a difference in MED and SMM because of the standard candidate order.

The methods are tested on two synthetic databases. Because they were small, MED could be tested and be used as "gold standard" for SMM and SMUC. SMM approximates the gold standard MED very well, and SMUC only slightly worse. When pruning is used, all methods improve their performance, and then achieve better compression ratio's.

Furthermore, the methods were tested on two real databases. For SMM, neither code table size nor compression ratio stabilizes. SMUC converges nicely, but the achieved compression ratio is not as good. Remember, the absolute compression ratios are not that interesting since the goal is not to compress, but compression is a means to select patterns. The interestingness of the returned patterns of the real databases is judged by domain experts. For the biological real database, it is said "For a biologist, these are typical, not surprising but still meaningful patterns". We are not entirely sure how to interpret this, for "not surprising, but still meaningfull" appears to be contradicting.

Still, the results are promising and thoroughly supported by theory and experiments. This is an approach that values the interestingness of the returned set of patterns, and defines interestingness of a pattern by the containment of much information in a compressed representation. The proposed methods show to have theoretical bounds on the quality and also the experiments have good results.

8.1 Some last words

We have seen several approaches to pattern (set) mining in uncertain databases. In section 6, we discussed the problem of mining all probabilistic frequent itemsets. In section 7, we considered a method to obtain probabilistic association rules. In section 8, we examined the problem of discovering characteristic patterns that compress well in expectation.

Since we are practising science, there are still a lot of problems unsolved. What is "interesting" is always a difficult question. We could think of situations were the probabilities in itself might provide patterns and contain interesting information, and not only "the higher, the better". *Unexpected* might be a key word, new information hidden in the data that you did not know before. Could we make a model with prior knowledge on a probabilistic database, as has been proposed for deterministic databases in section 5.8? Or should we try to find the most representative frequent patterns? Or could we try. . . ? There are a myriad of unexplored possibilities, and there always will be. This has been made clear by the development and adjustment of techniques over the years, as has been presented in the first part of this essay, and by the adaptation of techniques applied in other contexts and fields, as has been presented in the second part of this essay.

It is difficult. . . , no, even impossible to define "interestingness" in such a way that everyone is happy in every context. These individual differences cannot be bridged by new measures, but should be embraced to find and combine measures to broaden our view on pattern set mining. In this way, we will be able identify and tackle more problems in the future, maybe in even broader fields then we can imagine.

References

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases". In: *Proceedings of the 1993 ACM SIGMOD international conference on Management of data.* 1993, pp. 207–216.

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. "Fast algorithms for mining association rules". In: *Proc. 20th int. conf. very large data bases, VLDB.* Vol. 1215. 1994, pp. 487–499.

[3] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. "Discovery of frequent episodes in event sequences". In: *Data mining and knowledge discovery* 1.3 (1997), pp. 259–289.

[4] Jiawei Han et al. "Mining frequent patterns without candidate generation: A frequent-pattern tree approach". In: *Data mining and knowledge discovery* 8.1 (2004), pp. 53–87.

[5] Jian Pei, Jiawei Han, and Laks VS Lakshmanan. "Pushing convertible constraints in frequent itemset mining". In: *Data Mining and Knowledge Discovery* 8.3 (2004), pp. 227–252.

[6] Francesco Bonchi et al. "Exante: Anticipated data reduction in constrained pattern mining". In: *European Conference on Principles of Data Mining and Knowledge Discovery.* Springer. 2003, pp. 59–70.

[7] Jochen Hipp and Ulrich Güntzer. "Is pushing constraints deeply into the mining algorithms really what we want? an alternative approach for association rule mining". In: *ACM SIGKDD Explorations Newsletter* 4.1 (2002), pp. 50–55.

[8] Nicolas Pasquier et al. "Efficient mining of association rules using closed itemset lattices". In: *Information systems* 24.1 (1999), pp. 25–46.

[9] Toon Calders and Bart Goethals. "Non-derivable itemset mining". In: *Data Mining and Knowledge Discovery* 14.1 (2007), pp. 171–206.

[10] Arno J Knobbe and Eric KY Ho. "Maximally informative k-itemsets and their efficient discovery". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining.* 2006, pp. 237–244.

[11] Arno J Knobbe and Eric KY Ho. "Pattern teams". In: *European Conference on Principles of Data Mining and Knowledge Discovery.* Springer. 2006, pp. 577–584.

[12] Bjorn Bringmann and Albrecht Zimmermann. "The chosen few: On identifying valuable patterns". In: *Seventh IEEE International Conference on Data Mining (ICDM 2007).* IEEE. 2007, pp. 63–72.

[13] Floris Geerts, Bart Goethals, and Taneli Mielikäinen. "Tiling databases". In: *International conference on discovery science.* Springer. 2004, pp. 278–289.

[14] Mario Boley et al. "Direct local pattern sampling by efficient two-step random procedures". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining.* 2011, pp. 582–590.

[15] Matthijs Van Leeuwen and Antti Ukkonen. "Fast estimation of the pattern frequency spectrum". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer. 2014, pp. 114–129.

[16] Geoffrey I Webb. "Self-sufficient itemsets: An approach to screening potentially interesting associations between items". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4.1 (2010), pp. 1–20.

[17] Jefrey Lijffijt, Panagiotis Papapetrou, and Kai Puolamäki. "A statistical significance testing approach to mining the most informative set of patterns". In: *Data Mining and Knowledge Discovery* 28.1 (2014), pp. 238–263.

[18] Tijl De Bie. "Maximum entropy models and subjective interestingness: an application to tiles in binary databases". In: *Data Mining and Knowledge Discovery* 23.3 (2011), pp. 407–446.

[19] Tijl De Bie. "Subjective interestingness in exploratory data mining". In: *International Symposium on Intelligent Data Analysis.* Springer. 2013, pp. 19–31.

[20] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. "Krimp: mining itemsets that compress". In: *Data Mining and Knowledge Discovery* 23.1 (2011), pp. 169–214.

[21] Thomas Bernecker et al. “Probabilistic frequent itemset mining in uncertain databases”. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*. Ed. by John F. Elder IV et al. ACM, 2009, pp. 119–128. DOI: 10.1145/1557019.1557039. URL: https://doi.org/10.1145/1557019.1557039.

[22] Liwen Sun et al. “Mining uncertain data with probabilistic guarantees”. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*. Ed. by Bharat Rao et al. ACM, 2010, pp. 273–282. DOI: 10.1145/1835804.1835841. URL: https://doi.org/10.1145/1835804.1835841.

[23] Francesco Bonchi, Matthijs van Leeuwen, and Antti Ukkonen. “Characterizing Uncertain Data using Compression”. In: *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*. SIAM / Omnipress, 2011, pp. 534–545. DOI: 10.1137/1.9781611972818.46. URL: https://doi.org/10.1137/1.9781611972818.46.