Mining Top-K Patterns from Binary Datasets in presence of Noise

Claudio Lucchese*   Salvatore Orlando†   Raffaele Perego‡

Abstract

The discovery of patterns in binary dataset has many applications, e.g. in electronic commerce, TCP/IP networking, Web usage logging, etc. Still, this is a very challenging task in many respects: overlapping vs. non overlapping patterns, presence of noise, extraction of the most important patterns only.

In this paper we formalize the problem of discovering the Top-K patterns from binary datasets in presence of noise, as the minimization of a novel cost function. According to the Minimum Description Length principle, the proposed cost function favors succinct pattern sets that may approximately describe the input data.

We propose a greedy algorithm for the discovery of Patterns in Noisy Datasets, named PANDA, and show that it outperforms related techniques on both synthetic and real-world data.

1 Introduction

Many popular applications – for example, in the electronic commerce, TCP/IP networking, Web usage logging, or mobile communications domains – generate daily huge amounts of transactions. Such huge datasets can be stored as binary matrices, where each row is a transaction, and the 0-1 bits corresponds to the presence/absence of a given binary attribute (item) in each transaction. The binary attributes are in many cases not independent, as the transactions to which they belong generally model real-life phenomena and events. *Patterns*, i.e., collections of items whose occurrences are somehow related to each other [15], can be mined from these binary datasets, in order to understand common behavior and derive interesting knowledge about the people/phenomena/events that generated them. Finding significant patterns may be very complex, and algorithms for efficiently and effectively solving this problem attracted a lot of attention in the Data Mining community. In particular the problem of detecting the most "important" patterns embedded in data, often known as *Top-K patterns*, is a particularly challenging task.

In Figure 1 we show a toy binary dataset made of 50 items (columns) and 1000 transactions (rows). Thanks to a suitable re-ordering of rows and column, at first glance it is easy to observe the presence of three patterns (corresponding to the three dark rectangles).

---
*I.S.T.I.-C.N.R., Pisa, Italy.
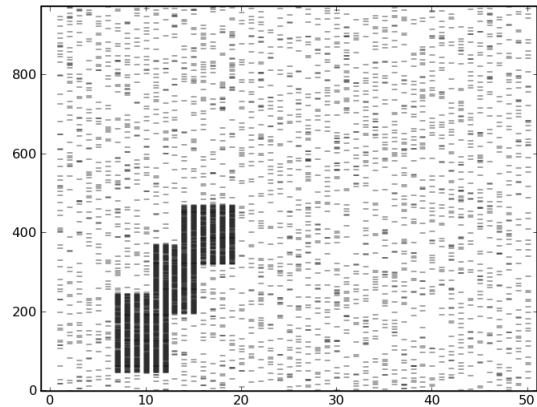†Dept. of Computer Science, Univ. of Venice, Italy.
‡I.S.T.I.-C.N.R., Pisa, Italy.

Figure 1: A synthetic binary dataset: rows correspond to transactions, columns correspond to items.

These Top-3 patterns consist of three overlapping sets of items occurring together in three overlapping sets of transactions. Nevertheless, the translation of our simple intuition into an actual pattern mining algorithm is very challenging. The overlapping of patterns and the presence of noise in real-world data (which is simulated by random flips in Figure 1) may hide these patterns. Indeed, no one of the algorithms proposed in literature is able to discover that the dataset in Figure 1 simply contains three patterns plus some noise.

Frequent itemset mining algorithms are not able to find those patterns due to the presence of noise: not all the items of a pattern may always occurs simultaneously in the set of supporting transactions/observations. A way to solve this issue is to relax the definition of support, as for Error Tolerant Itemsets (ETIs) [3, 16, 17]. But also ETIs have a number of drawbacks: they are too many, and they contain a lot of spurious itemsets containing only noisy occurrences.

Among the *dataset tiling* techniques, a recent proposal is HYPER+ [19], which tries to cover with at most K pattern every item occurring in the dataset. Since the algorithm assumes that no noise is present, it tries to cover every noisy bits as well, and thus fails in finding the Top-3 patterns in Figure 1.

Other interesting approaches for solving the prob-

lem of detecting the Top-K patterns in a noisy binary dataset are based on *matrix decomposition*. In this regards, it is worth discussing the Discrete Basis Problem [13], which has partially inspired the novel framework proposed in this paper. The algorithm proposed to solve that problem is called ASSO [13], and yields a set of K basis vectors, i.e., sets of correlated attributes, but also determines how these vectors must be combined to express each rows/transactions of the binary dataset. The output of the algorithm can also be interpreted a set of K patterns, i.e. itemsets and related transactions, which can approximately cover the input data. The parameter K forces ASSO to introduce some errors, i.e. sets of bits correctly or incorrectly covered by the mined patterns. Therefore ASSO greedily discovers a set of K basis that minimizes such error, but unfortunately fails in discovering the Top-3 patterns of Figure 1, as discussed in the Section 4.

In this paper we thus introduce a new *Top-K Pattern Discovery Problem*, whose formulation is based on the minimization of a cost function, which takes into account both the *error* and the *complexity* of the patterns being discovered. We claim that the previous proposals, like ASSO, may generate over-fitted models. In fact ASSO minimizes only the error, and this leads to the extraction several patterns that try to cover noisy occurrences.

Our work contains several original contributions. First, our problem formulation exploits a novel cost model used to evaluate the goodness of the set of patterns extracted. By minimizing our cost function we are able to detect the Top-K patterns, i.e. itemsets and their supporting transactions, according to the Minimum Description Length (MDL) principle [14]. Second, we propose PANDA[1], a new efficient mining algorithm for the discovery of Patterns in Noisy Datasets. In particular, the PANDA robustness is due to its ability to effectively deal with both *false positives*, i.e. pairs of item/transaction present in the cover but not in the data, and *false negatives*, i.e. pairs of item/transaction present in the data but not in the cover. Finally, we show that PANDA outperforms ASSO in terms of the quality of the extracted patterns. Several tests were conducted, by exploiting both synthetic and real-world datasets, with the aim of validating the effectiveness of our proposed algorithm.

The paper is structured as follows. Section 2 introduces our cost function and formalize the Top-K Pattern Discovery Problem in noisy binary datasets. Section 3 presents PANDA, a robust and efficient algorithm for solving our problem. The experimental settings and the

results of the tests conducted are analyzed in Section 4. Finally, Section 5 discusses the related works, while Section 6 draws some conclusions.

2 Problem Statement

DEFINITION 1. (TRANSACTIONAL DATASET $\mathcal{D}$) *Let* $\mathcal{D} \in \{0,1\}^{N \times M}$ *be the* binary *representation of a transactional dataset, composed of N transactions* $\{t_1, \ldots, t_N\}$, $t_i \subseteq \mathcal{I}$, *where* $\mathcal{I} = \{a_1, \ldots, a_M\}$ *is a set of M items. We have that $\mathcal{D}(i,j) = 1$ iff $a_j \in t_i$, $\mathcal{D}(i,j) = 0$ otherwise.*

We argue that any observed data $\mathcal{D}$ can be *explained* by the occurrence of a set of patterns Π, plus some noise. A pattern $P \in \Pi$ is defined as a set of items occurring in a set of transactions, and can be represented by a couple $P = \langle P_I, P_T \rangle$, where $P_I \in \{0,1\}^M$ and $P_T \in \{0,1\}^N$. The observed dataset $\mathcal{D}$ is indeed obtained as the *or*-ing of the various patterns Π, plus some noise. If this is the case, we say that Π is a *pattern model* of $\mathcal{D}$.

DEFINITION 2. (PATTERN MODEL)
Let $\mathcal{D}$ be the binary representation of the input data, a set of patterns Π is called a pattern model *of $\mathcal{D}$, if:*

$$\mathcal{D} = \bigvee_{P \in \Pi} (P_T \cdot P_I^\mathsf{T}) \quad \oplus \quad \mathcal{N}$$

where $\bigvee$ is the logical or *operation, $P_T \cdot P_I^\mathsf{T}$ is the outer product of the two binary vectors (thus $(P_T \cdot P_I^\mathsf{T}) \in \{0,1\}^{N \times M}$), $\oplus$ is the element-wise* xor *operation, and $\mathcal{N} \in \{0,1\}^{N \times M}$ models the* noise *which acted on the data by flipping some uncorrelated bits.*

The above model has many similarities with other proposals in related fields such as probabilistic *latent semantic analysis*, *factor analysis*, and *topic models*, where the keyword *pattern* is replaced by *aspect*, *unobserved class variable*, *factor*, or *latent topic*, etc. In Section 5 we illustrate in detail differences and commonalities with our model. Also in our model the set of patterns Π is latent. It is unobserved by a user (and even obfuscated due to the presence of noise), but it is crucial for understanding the data.

Our goal is thus to discover the set of patterns Π that better explains a dataset $\mathcal{D}$. Unfortunately, there are several different pattern sets and noise matrices that can explain the data according to Definition 2. We choose to pick the best pattern set among all the possible candidates according to the the Minimum Description Length (MDL) principle [14]. When choosing among different models, which are somehow encoded, the MDL principle suggests to adopt the model with the minimum code length. To this end, we define two

---

[1]The code is available at `http://hpc.isti.cnr.it/~claudio`

encoding cost functions $\gamma_P : \{0,1\}^N \times \{0,1\}^M \to \mathbb{R}$ and $\gamma_{\mathcal{N}} : \{0,1\}^{N \times M} \to \mathbb{R}$, respectively for patterns $P \in \Pi$ and for $\mathcal{N}$, and require that the selected pattern set Π, and the resulting noise matrix, to have the minimum costs. Finally, we force Π to have cardinality at most K.

In the following we also assume the existence of a (hidden) true pattern set Ω that actually generated the data, and we argue that the pattern set Π, selected according to the MDL principle, closely approximates Ω.

We can finally introduce the *Top-K Pattern Discovery Problem*.

PROBLEM 1. (TOP-K PATTERN DISCOVERY PROBLEM) *Given a binary dataset $\mathcal{D} \in \{0,1\}^{N \times M}$, and an integer K, find the pattern set Π, $|\Pi| \leq K$, that defines a pattern model of $\mathcal{D}$ and minimizes the following cost:*

$$\gamma(\Pi, \mathcal{D}) = \sum_{P \in \Pi} \gamma_P(P_T, P_I) \; + \; \gamma_{\mathcal{N}}(\mathcal{N})$$

where $\gamma_P(x,y) = \|x\|_F + \|y\|_F$ and $\gamma_{\mathcal{N}}(z) = \|z\|_F$. The Frobenius norm $\| \cdot \|_F$ when applied to either a binary vector or a binary matrix simply counts the number of 1 bits it contains.

The rationale of the two cost functions γ_P and $\gamma_{\mathcal{N}}$ is to penalize items that are not clustered together. Suppose that a large "rectangle" of 1's is present in $\mathcal{D}$. The mining algorithm has to decide whether to consider this rectangle being noise or an actual pattern. In the first case, the algorithm incurs in a cost equal to the number of bits, i.e. the *area* of the rectangle. In the second case, the rectangle is described just using its items and transactions, and the cost is thus equal to its *semiperimeter* (plus 1). Being the second option more cost effective, i.e. with a smaller encoding cost, the rectangle is promoted to a pattern. From an information theoretical point of view, patterns in data are useful in generating a compressed description, the remainder is supposed to be noise.

The pattern model allows both *false positives* and *false negatives* to occur in the pattern set extracted Π. A false positive occurs when an item which is not present in $\mathcal{D}$ is covered by a pattern in Π, while a false negative occurs when an item present in $\mathcal{D}$ is not covered by Π.

More formally, we call *expected ground truth* how the dataset would look like if noise was not present. The expected ground truth deriving from the discovered pattern set Π is:

$$\overset{\triangle}{\Pi} \; = \; \bigvee_{P \in \Pi} (P_T \cdot P_I^{\mathsf{T}})$$

Note that, according to Definition 2, the noise matrix $\mathcal{N}$ can be computed in terms $\overset{\triangle}{\Pi}$ as $\mathcal{N} = \mathcal{D} \oplus \overset{\triangle}{\Pi}$, and accounts of both false positives and negatives:

False Positives if $\mathcal{D}(i,j) = 0$ and $\overset{\triangle}{\Pi}(i,j) = 1$

False Negatives if $\mathcal{D}(i,j) = 1$ and $\overset{\triangle}{\Pi}(i,j) = 0$

Therefore, in both the false positives and false negative cases, we have that $\mathcal{N}(i,j) = 1$.

2.1 Boolean matrix decomposition.

The problem of discovering a suitable pattern model Π according to Definition 2 can also be seen as a *Boolean matrix decomposition problem*.

If $|\Pi| = K$, let $\mathcal{P}_T \in \{0,1\}^{N \times K}$ and $\mathcal{P}_I \in \{0,1\}^{K \times M}$ be two Boolean binary matrices, where $\mathcal{P}_T$ is obtained by juxtaposing the K *column* vectors P_T, whereas $\mathcal{P}_I$ is obtained by juxtaposing the K *row* vectors P_I^{T}. More specifically, for each $P = \langle P_I, P_T \rangle \in \Pi$, if P_I^{T} appears in the h-th row of $\mathcal{P}_I$, then $\mathcal{P}_T$ has to appear in h-th column of $\mathcal{P}_T$. Then the following equation holds:

$$\mathcal{D} \; = \; \bigvee_{P \in \Pi} (P_T \cdot P_I^{\mathsf{T}}) \; \oplus \; \mathcal{N} = \mathcal{P}_T \circ \mathcal{P}_I \; \oplus \; \mathcal{N}$$

where $\circ$ denotes the *Boolean product* of two matrices, i.e. a matrix product with addition and multiplication being replaced by the logical *or* and *and* operations respectively.

The above equation holds because, according to Definition 2, each element (i,j) of matrix $\overset{\triangle}{\Pi} = \bigvee_{P \in \Pi}(P_T \cdot P_I^{\mathsf{T}})$ is obtained by *or*-ing the corresponding binary entries (i,j) of all the matrices $(P_T \cdot P_I^{\mathsf{T}}) \in \{0,1\}^{N \times M}$. Therefore, we can write:

$$\overset{\triangle}{\Pi}(i,j) \; = \; \bigvee_{P \in \Pi} P_T(i) \cdot P_I(j)$$

Note that the above equation exactly corresponds to the Boolean dot-product that involves the i-th row of $\mathcal{P}_T$ and the j-th row column of $\mathcal{P}_I$, and thus produces the entry (i,j) of the Boolean matrix product $\mathcal{P}_T \circ \mathcal{P}_I$.

The problem of finding suitable matrices $\mathcal{P}_T$ and $\mathcal{P}_I$ is also discussed in [13], where the Discrete Basis Problem is introduced. The rows of $\mathcal{P}_I$ are called basis, since every transaction in $\mathcal{D}$ can be obtained by *or*-ing a subset of those basis. Each row of matrix $\mathcal{P}_T$ encodes which basis are needed for any given transaction.

Indeed, the same pattern model described in Definition 2 is adopted, but with a different cost function.

The authors aim at minimizing the noise $\mathcal{N}$ only, without taking into account any cost associated with the extracted pattern set Π. Their formulation can be thus reduced to Definition 1, with $\gamma_P(\cdot) = 0$. Note that the decision version of the Discrete Basis Problem was proven to be NP-Complete.

The authors propose a greedy algorithm, called Asso for approximating the best solution for the Discrete Basis Problem. First, a set of candidate basis is created by using global statistics. The i-th candidate basis is composed of item a_i plus any other item a_j having correlation with a_i greater than a user defined threshold τ. Then, K basis are iteratively selected by greedily minimizing the cost function.

In the present work, we generalize the framework proposed in [13] with a novel cost function, which takes into account the *complexity* of the discovered pattern set. We claim that minimizing $\mathcal{N}$ is not a sufficient criteria, since trying to cover every bit in $\mathcal{D}$ means to cover also occurrences introduced by noise, and it is thus likely to lead to over-fitting issues. We argue that the exploitation of the MDL principle may solve such issues, thus supporting the extraction of an improved pattern set.

3 Algorithm PANDA

The solution space of Problem 1 is extremely large. There are 2^M times 2^N candidate patterns – where M is the number of items $\mathcal{I}$, and N is the number of transactions – out of which only a few must be selected.

To tackle such a large search space, we propose a greedy algorithm named PANDA. It adopts two heuristics. First the problem of discovering a pattern is decomposed into two simpler problems: discovering a *core pattern* [4] and extending it to form a good approximate pattern. Second, rather than considering all the 2^M possible combination of items, these are sorted and processed one by one without backtracking.

Similarly to AC-CLOSE [4], we assume that, even in presence of noise, a true pattern $P \in \Omega$ occurs in $\mathcal{D}$ with a smaller *core pattern*. That is, given a pattern $P = \langle P_I, P_T \rangle$, there exists $C = \{C_I, C_T\}$ such that $C_T(i) = 1 \Rightarrow P_T(i) = 1$, $C_I(j) = 1 \Rightarrow P_I(j) = 1$ and $C_T(i) = 1 \wedge C_I(j) = 1 \Rightarrow \mathcal{D}(i,j) = 1$. Then, given C, PANDA adds items and transactions to C, until the largest extension of C that minimizes the overall cost γ is found.

Algorithm 1 gives an overview of the PANDA algorithm. It iterates two main steps at most K times, where K, provided by the user, is the maximum number of patterns to be extracted. During each iteration, first a core pattern C is extracted (line 4), and then it is extended to form a new pattern C^+ (line 5). If C^+

reduces the cost of the model, it is added to the current pattern set Π. These two steps are described in detail in the following sections. Regardless the user-provided parameter K, PANDA stops generating new patterns if they do not improve the cost of the model (line 6).

The algorithm uses a particular view $\mathcal{D}_R$ of the dataset $\mathcal{D}$ for the discovery of a new core pattern (line 10). This view is called *residual dataset* and it is computed by setting $\mathcal{D}(i,j) = 0$ for any i,j such that $P_T(i) = 1$ and $P_I(j) = 1$ for some previously discovered pattern P. The rationale is that we want to discover new patterns that *explain* a portion of the database that was not already covered by any previous pattern.

3.1 Extraction of dense cores

The procedure FIND-CORE extracts a core pattern C from the residual dataset $\mathcal{D}_R$, and returns an *ordered* list E of items that are later used to extend C.

The procedure is described in Algorithm 2. The extension list E is initialized to the empty set. In order to reduce the search space, items in $\mathcal{D}_R$ are sorted (line 3). We discuss a number of possible orderings in Section 3.3. The core pattern C is initialized with the first item s_1 in the ordered list S, and its supporting transactions in $\mathcal{D}_R$. The remaining items in S are processed one by one. The current item s_h is used to create a new candidate pattern $C*$ (lines 8-10). Since in this phase we do not allow noise, when we add an item to a pattern (line 9), as a consequence we can have a reduction in the number of supporting transactions (line 10). If C^* reduces the cost of the pattern set with respect to C, then C^* is promoted to be the new candidate (line 12) and it is used in the subsequent iteration. Otherwise, s_h is appended to the extension list E (line 14), and it can be used later to extent the extracted dense core.

Eventually, every item occurring in $\mathcal{D}_R$ has been processed only once, and either it was added to the dense core C, or it was appended to the extension list E.

Algorithm 1 PANDA algorithm.

1: $\Pi \leftarrow \emptyset$ ▷ the current collection of patterns
2: $\mathcal{D}_R \leftarrow \mathcal{D}$ ▷ the residual data yet to be explained
3: **for** $iter \leftarrow 1, \ldots, K$ **do**
4:      $C, E \leftarrow$ FIND-CORE$(\mathcal{D}_R, \Pi, \mathcal{D})$
5:      $C^+ \leftarrow$ EXTEND-CORE$(C, E, \Pi, \mathcal{D})$
6:      **if** $\gamma(\Pi, \mathcal{D}) < \gamma(\Pi \cup C^+, \mathcal{D})$ **then**
7:         **break** ▷ cost cannot be improved any more
8:      **end if**
9:      $\Pi \leftarrow \Pi \cup C^+$
10:     $\mathcal{D}_R(i,j) \leftarrow 0 \ \ \forall i,j$ s.t. $C_T^+(i) = 1 \wedge C_I^+(j) = 1$
11: **end for**

Algorithm 2 Core Pattern discovery.

1: **function** FIND-CORE($\mathcal{D}_R$, Π, $\mathcal{D}$)
2:     $E \leftarrow \emptyset$ ▷ Extension list
3:     $S = \{s_1, \ldots, s_M\} \leftarrow$ SORT-ITEMS-IN-DB($\mathcal{D}_R$)
4:     $C \leftarrow \langle C_T = 0^N, C_I = 0^M \rangle$
5:     $C_I(s_1) \leftarrow 1$
6:     $C_T(i) \leftarrow 1 \quad \forall i$ s.t. $\mathcal{D}_R(i, s_1) = 1$
7:     **for all** $h \leftarrow 2, \ldots, M$ **do**
8:         $C^* \leftarrow C$ ▷ create a new candidate
9:         $C_I^*(s_h) \leftarrow 1$
10:        $C_T^*(i) \leftarrow 0 \quad \forall i$ s.t. $\mathcal{D}_R(i, s_h) = 0$
11:        **if** $\gamma(\Pi \cup C^*, \mathcal{D}) \leq \gamma(\Pi \cup C, \mathcal{D})$ **then**
12:           $C \leftarrow C^*$
13:        **else**
14:           E.append(s_h)
15:        **end if**
16:     **end for**
17:     **return** C, E
18: **end function**

Algorithm 3 Extension of a Core Pattern.

1: **function** EXTEND-CORE(C, E, Π, $\mathcal{D}$)
2:     **while** $E \neq \emptyset$ **do**
                            ▷ add a new item
3:         $e \leftarrow E$.pop()
4:         $C^* \leftarrow C$
5:         $C_I^*(e) \leftarrow 1$
6:         **if** $\gamma(\Pi \cup C^*, \mathcal{D}) \leq \gamma(\Pi \cup C, \mathcal{D})$ **then**
7:           $C \leftarrow C^*$
8:         **end if**
                     ▷ add new transactions
9:         **for** $i \in \{1, \ldots, N\}$ s.t. $C_T^*(i) = 0$ **do**
10:           $C^* \leftarrow C$
11:           $C_T^*(i) \leftarrow 1$
12:           **if** $\gamma(\Pi \cup C^*, \mathcal{D}) \leq \gamma(\Pi \cup C, \mathcal{D})$ **then**
13:             $C \leftarrow C^*$
14:           **end if**
15:         **end for**
16:     **end while**
17:     **return** C
18: **end function**

The procedure returns C and E for further processing.

Being a greedy approach, the procedure may not find the best core pattern, i.e. the one minimizing the cost. However, since the procedure is invoked K times, this risk is amortized and the probability of getting a good set of core patterns out of K runs will be sufficiently large. In addition, we will consider in Section 3.4 a randomization strategy to avoid local minima. This makes it possible to successfully exploit a greedy and simple strategy as the one we just described.

A prefix-tree based data structure is used to store the transaction of $\mathcal{D}_R$, which are sorted according to descending items frequency. A prefix-tree allows to easily test the goodness of a new candidate, i.e. when a new item is added, by processing only a subset of its branches.

3.2 Extension of dense cores

Given a dense core $C = \langle C_I, C_T \rangle$, the procedure EXTEND-CORE tries to add items to C_I and transactions to C_T, possibly introducing some noise, as long as the overall representation cost is reduced. The procedure is described in Algorithm 3.

An extension list $E \subset \mathcal{I}$ is given. The first item e is removed from the list E, and it is added to C_I thus forming a new pattern C^* (line 5). If this pattern does not improve the cost of representing Π and $\mathcal{N}$ (line 6), then item e is disregarded.

The set C_T is extended analogously. A transaction t_i, which has not yet been considered in C_T, is used to create a new candidate pattern C^* (line 11). If the new pattern carries any improvement over the previous representation (line 12), that C^* is promoted and considered for further extensions in place of the old C.

The above two steps are repeated as long as there is a new item in E.

There are some interesting subtleties in the evaluating process of new pattern C^*. First C^* may introduce some noise. If the item e is added to C_I^*, then e may not occur in every transaction C_T^*, and similarly when extending C_T. The noise cost thus increases by the number of missing items, i.e. false positives. At the same time, the extension may cover a number of items that were already considered as noise by some previous patterns, and this noisy bits are not accounted again in the cost function $\gamma_{\mathcal{N}}$. If the balance between items covered and noise introduced justifies the increased representation cost of C^*, then the new pattern C^* is accepted.

Second, the convenience of C^* depends on the amount of data it covers, that was not already covered by other patterns in Π. For example, suppose that C^* does not introduce any noise, and that all the items covered thanks to the extension were already covered by another pattern in Π. Then, the only variation in the cost function occurs because of the cost of representing Π, which increases thus making the extended pattern C^* non convenient.

The algorithm stores a vertical representation of the dataset, where tid-lists are stored for each item, i.e. the list of the transactions containing a given item. This allows to quickly check the cost incurred when adding a new item or a new transaction to C^*. This data

structure is updated after a new pattern is discovered, in order to distinguish items that were already covered by the current pattern set Π.

3.3 Sorting items

The ordering of the items chosen in Alg. 2 at line 3 affects the whole algorithm. Indeed, items are associated with a *score*, and then ordered by descending score. Such ordering is a greedy choice that may hurt the quality of the extracted patterns. We consider four scoring strategies:

- **Frequency.** The score of an item is given by its frequency in $\mathcal{D}_R$.

- **Couples frequency.** As in [20], items are sorted according to their contribution to the frequency of the 2-itemsets in $\mathcal{D}_R$. The score of item a_j is given by the sum of the supports of every itemset $\{a_j \cup a_k\}$ occurring in $\mathcal{D}_R$.

- **Correlation.** The score is given by the correlation with the *current* candidate pattern C.

- **Prefix child.** The itemset C is identified by a set of paths on the prefix-tree starting from its root. Given the deepest nodes in those paths, the support of their children is cumulated to compute a score.

The rationale of the frequency strategy is clear: the most frequent item has the highest probability of being part of a core pattern. This is also the least expensive ordering to compute.

The other three strategies aim at grouping together items with high correlation. The couples frequency based order is slightly more expensive, since the whole prefix-tree must be processed. The correlation based order must be re-computed every time the itemset C is updated (i.e. after line 10 of Alg.2), and a processing is required of those portions of the prefix-tree supporting C. Finally, the prefix child based strategy approximates the correlation one by reducing the number of nodes of the prefix-tree visited to re-calculate a new ordering. In the last two strategies, when $C = \emptyset$ the ordering is given by the items' frequency.

3.4 Randomizing PANDA

To reduce the risks of a greedy strategy we introduce some randomization. The lines 4 and 5 of Alg. 1 are repeated R times, where R is the number of randomization rounds. During each round the ordering of items is perturbed as follows. Each item is associated with a *randomized score* computed by drawing a random number between 0 and its original score raised to the power of τ. Then, items are re-ordered according to their randomized score.

The parameter τ works as an *computational temperature*, controlling the amount of randomization allowed. The larger τ, the larger the probability of having the largest score for an item with high support. The parameter τ is decreased exponentially at each randomization round.

After that R candidate patterns have been found, only the best is evaluated for its inclusion in the pattern set Π.

4 Experiments

4.1 Experiments on synthetic data

Evaluating the goodness of the patterns discovered by a mining process is a very difficult task. The patterns present in a given dataset are unknown, and therefore there is no *ground truth* that we can use as a benchmark. However, Gupta, et al. [8] proposed a collection of synthetic datasets, along with a set of evaluation measures to evaluate the goodness of the patterns extracted in presence of noise. The idea is to synthesize a dataset by embedding a given set of patterns – i.e., the *ground truth* Ω – and then introduce noise by randomly flipping a given percentage of bits in the corresponding binary representation. An example of these synthetic datasets is shown in Figure 1.

Since the embedded true patterns occurring in the dataset are known, it is possible to compare the patterns extracted by any mining algorithm. Due to space constraints, we only use three out of the eight datasets proposed, namely the datasets identified by numbers 5, 6, and 8. Respectively, a number of two, three, and four patterns are embedded in these datasets. Moreover, there are overlaps between the embedded patterns. All the datasets have 50 items and 1000 transactions. These datasets allow for repeatability of experiments, and they can be used to detect and understand the weaknesses of an algorithm.

The authors of [8] propose four evaluation measures that only consider the items of a discovered pattern, i.e. only P_I for each discovered pattern $P \in \Pi$. We are instead interested in evaluating an improved quality measure, which takes into account both the sets P_I and P_T.

Given the true patterns Ω and the discovered ones Π, we measure the goodness Π on the basis of how well Π matches Ω. We say that an occurrence is correctly *retrieved* iff:

$$\overset{\triangle}{\Pi}(i,j) = 1 \text{ and } \overset{\triangle}{\Omega}(i,j) = 1$$

Note that if this holds, we cannot derive that $\mathcal{D}(i,j) = 1$, since this bit might be flipped due to the noise.

We adapt the usual precision and recall measures to this setting, and compute the *pattern-based* F-measure
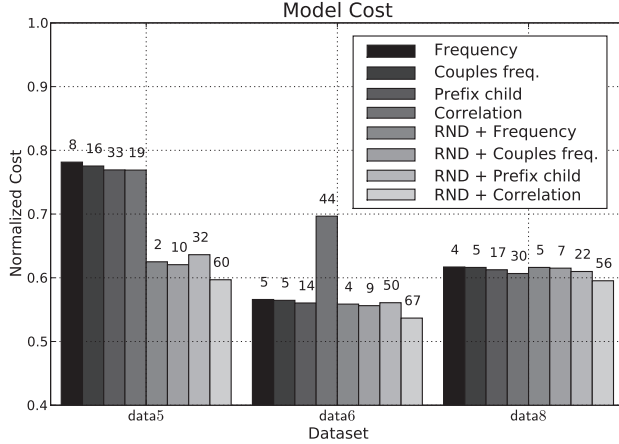
Figure 2: Model cost of the several variants of the PANDA algorithm. On top of each bar, the number of discovered patterns.

Figure 3: Patterns extracted by PANDA from data6.

F_p as follows:

$$Prec_p(\Pi) = \sum_{i,j} \overset{\triangle}{\Pi}(i,j) \cdot \overset{\triangle}{\Omega}(i,j) \; / \; \| \overset{\triangle}{\Pi} \|_F$$

$$Rec_p(\Pi) = \sum_{i,j} \overset{\triangle}{\Pi}(i,j) \cdot \overset{\triangle}{\Omega}(i,j) \; / \; \| \overset{\triangle}{\Omega} \|_F$$

$$F_p(\Pi) = \frac{2 \cdot Prec_p(\Pi) \cdot Rec_p(\Pi)}{Prec_p(\Pi) + Rec_p(\Pi)}$$

In the following we use the cost function $\gamma(\Pi, \mathcal{D})$ and the pattern-based F-measure $F_p(\Pi)$ to evaluate the variants of the PANDA algorithm, and to compare the best of those with the ASSO algorithm.

In Figure 2 we report the normalized model costs, i.e. $\gamma(\Pi, \mathcal{D})/\|\mathcal{D}\|_F$, for the four variants of the PANDA algorithm resulting from the four item ordering strategies, and their randomized versions. The test was run on the dataset data5 with a noise level of 4%, i.e. each bit in $\mathcal{D}$ was randomly flipped with probability 4%. The parameter K was set to infinity and the number of randomization rounds R to 20.

The first observation is that the introduction of randomization brings a significant benefit. This is evident in dataset data5 where the cost of the model is reduced from about 0.8 to 0.6. The second observation regards the number of discovered patterns, which are shown on top of each bar. Generally this number increases with the randomized approach. We deduce that randomization helps in finding smaller dense regions, which can be added to the extracted model Π, still decreasing the total cost.

We conclude that the most promising variants of PANDA are: (1) the randomized frequency and (2) the randomized correlation. The former has a very simple
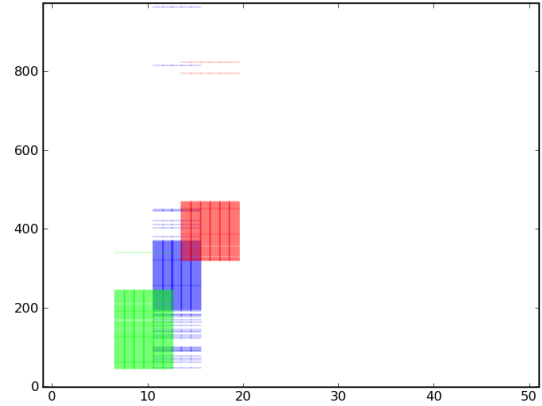
and slightly faster implementation, and the introduction of randomization does not increase the number of patterns. The latter has a better performance in terms of cost of the model, but it always generates the largest number of patterns. Therefore, for the remainder of the paper we focussed on those two variants only.

Next, we compare PANDA against the ASSO algorithm [13]. Even if we know exactly how may patterns have been embedded in each synthetic dataset, this information is not available in real data analysis tasks. So we asked the algorithms to extract the best $K = 15$ patterns from each dataset. The rationale is to avoid any biasing due to an information that usually is not known in advance.

In Figures 4.a-c we evaluate the ability to minimize the cost of the model representation as defined in Problem 1, as a function of the amount of random noise (percentage of flipped bits) embedded in the datasets. The three algorithms have very similar behavior, with no significant differences. The only exception regards dataset data5, where ASSO produces a representation with a much larger cost for small noise levels.

The above comparison may seem unfair, since ASSO optimizes a different cost function, namely the amount of noise $\|\mathcal{N}\|_F$. For this reason, in Figures 4.d-f we report the normalized model error $\|\mathcal{N}\|_F/\|\mathcal{D}\|_F$ produced by the three algorithms on the same datasets. In this case, the ASSO algorithm outperforms PANDA, and the differences appear more significant than in the previous set of plots. Again, ASSO is not able to find a good pattern set for data5 when no error is present.

Recall that our primary goal is to approximate the true pattern set Ω hidden in the datasets. Figures 4.g-i show the pattern-based F-measure F_p as a function of the noise embedded in the datasets. In every experiment, both the two variants of the PANDA algorithm

show consistently better performance than Asso. Indeed, PaNDa reaches a value of F_p about 10% larger in every experiment.

We draw the following conclusion. Asso is able to find a good covering of the given dataset, i.e. to minimize $\mathcal{N}$. On the other hand, PaNDa can discover the *true* patterns embedded in the data.

In Figure 4.1 we show the actual patterns extracted by PaNDa from data6 after 6% random flips. The dataset is same shown in Figure 1. The PaNDa algorithm was able to discover the three embedded patterns and their supporting transaction. Only a few spurious transactions were added, because they contain large portion of patterns' items due to noise.

4.2 Experiments on real data

We conducted a set of experiments on a collection of real-world datasets taken from the UCI Machine Learning Repository [1]. This repository contains labelled binary and non-binary data, which are mainly used to validate classification and clustering tasks.

Indeed, a pattern P identifies a cluster of correlated transactions P_T, and therefore we evaluated the pattern set Π as it was the result of a clustering task. Hereinafter, we use the term *clusters* to address the transaction sets corresponding to the extracted patterns, and the term *classes* to address the partitioning of transactions in the benchmark dataset.

Unfortunately, the patterns discovered by both PaNDa and Asso may overlap, i.e. a transaction may belongs to different patterns. Conversely, in the test datasets all the transactions exclusively belong to a single class.

In order to allow a sound comparison between the discovered clusters and the actual classes present in the data, we post-processed the output of the two algorithms, so as to associate each transaction with a single best matching pattern. For each transaction t_i, all patterns $P \in \Pi$ are ranked according to the *overlap score*: $\sum_j P_I(j) \cdot \mathcal{D}(i,j)^2$. In case of ties, the pattern P with the largest $\|P_I\|_F$ is preferred. Finally, transaction t_i is retained by the first ranked pattern, and removed from the others. Eventually, every transaction will be associated with one pattern only, thus obtaining a partitional clustering as required.

The evaluation of the updated pattern set Π^* was carried out by means of two external indices to measure the goodness of clustering [18]. Let $F(i,j)$ be the usual F-measure of the i-th extracted cluster with respect to the j-th actual class embedded in the dataset. The

---

^{2}The cardinality of the set-intersection between the itemset identified by P_I and transaction t_i

clustering F-measure F_c is thus defined as follows:

$$F_c(\Pi^*) = \sum_j \frac{N_j}{N} \max_i F(i,j)$$

where N_j is the number of transactions in the j-th class. The function F_c averages the F-measures of the best extracted clusters for each actual class j.

The second measure is the Jaccard index $J(\Pi^*)$:

$$J(\Pi^*) = J_{11}/(J_{11} + J_{10} + J_{01})$$

where J_{11} is the number of transaction pairs (t_i, t_j) appearing together in the same cluster as well as in a class, J_{10} is the number of pairs appearing together in a cluster but belonging to different classes, and J_{01} is the number of pairs belonging to different clusters but to the same class.

We measured the quality of the clusters extracted from 22 different datasets, which have been discretized using the LUCS-KDD DN software [5]. The class label was removed in advance. Also in this case we asked each algorithm the top 15 patterns.

Table 4.2 reports the average length ($\|P_I\|_F$) and support ($\|P_T\|_F$) of the patterns in the extracted set Π before the post-processing, and the values of the two quality indices F_c and J. Asso performs best only on four datasets, while the randomized correlation variant of PaNDa performs best on average: it provides an F_c value 18% better than Asso, and the improvement is 65% for the J index. We also report the geometric mean, since this is less affected by outliers: the gains are respectively 17% and 83% respectively for the F_c and J measures with respect to Asso.

An interesting observation is that the PaNDa variants extract much smaller patterns, with less than half of the items. This is because the size of the patterns' description is a cost to be minimized. In fact, overlaps are penalized since they cover the same data, thus producing a less redundant pattern set. It is worth underlining that shorter patterns are more human understandable, and possibly have a higher prediction power.

We do not report the running time of the algorithms, since, even if PaNDa is slightly slower, the average running time is below one second on average.

In conclusion, we proved that on both synthetic and real world data, our pattern model may help in finding those true patterns hidden in the data.
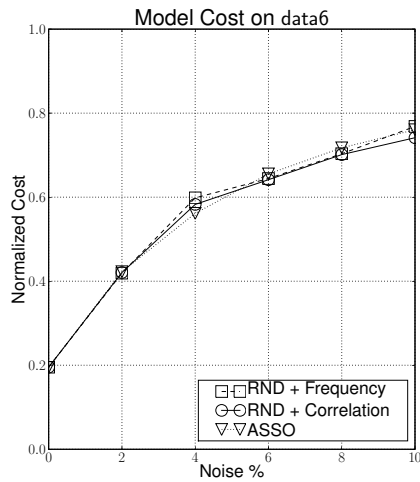
5 Related Work

We classify related works in three large categories: matrix decomposition based, database tiling, frequent itemsets based. All of them have many similarities with the Top-K Pattern Discovery Problem. However, there
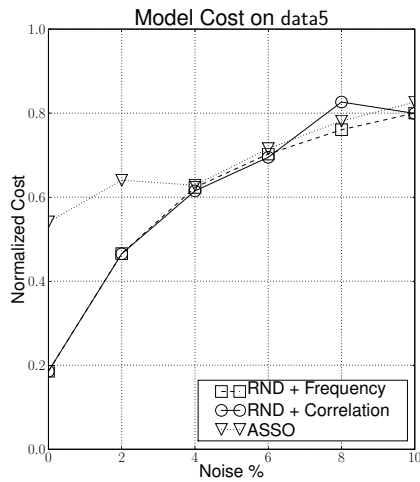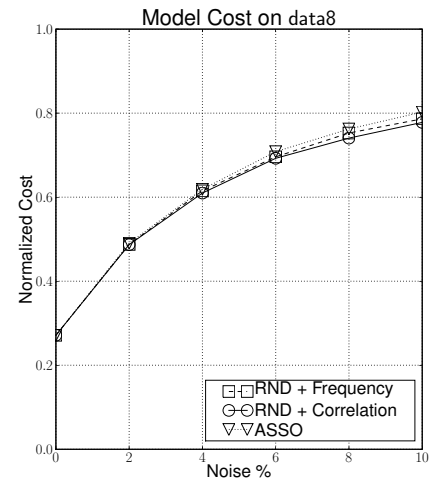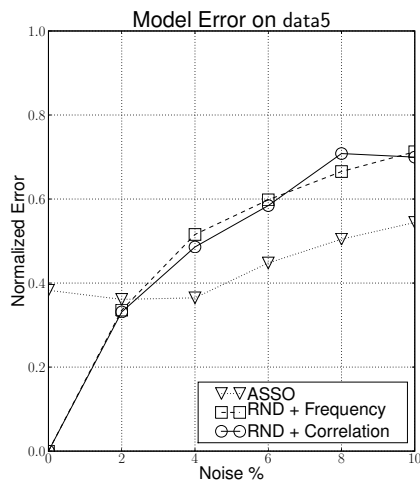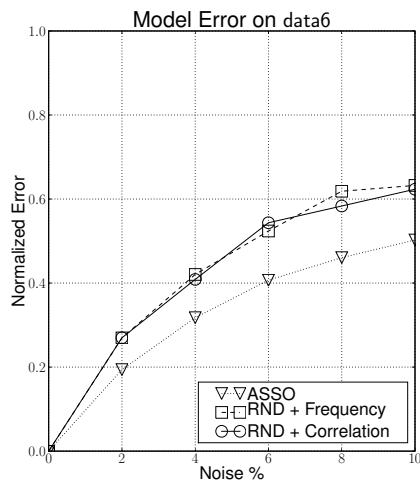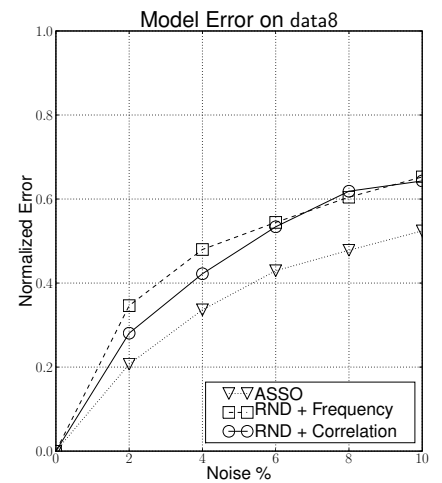
Figure 4: Comparison of PANDA and ASSO on different synthetic datasets, on varying the noise level.

Table 1: Evaluation of PANDA and ASSO on 22 real-world datasets.

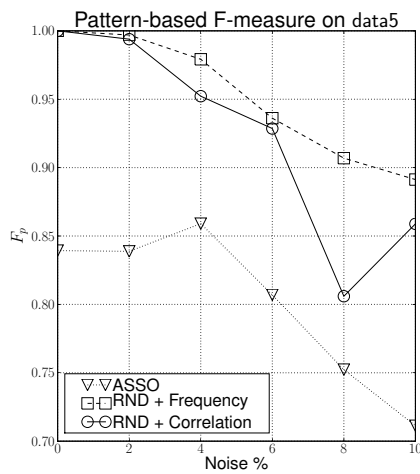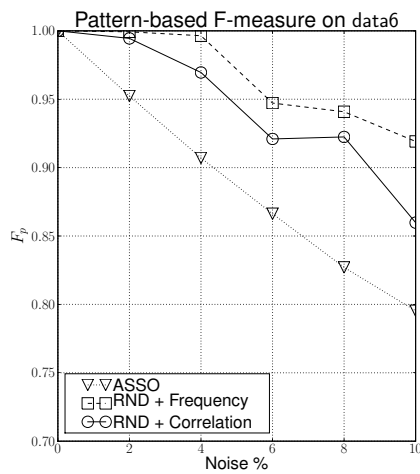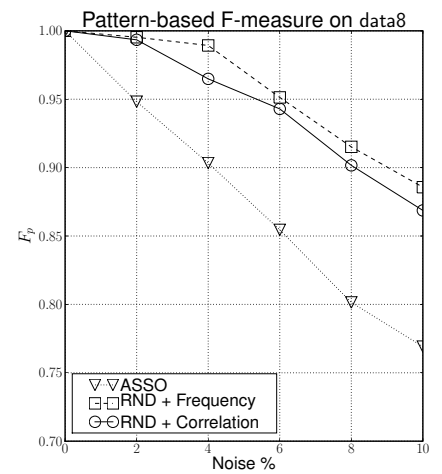| dataset	PANDA RND + Frequency				PANDA RND + Correlation				ASSO			
	len	supp	F_c	J	len	supp	F_c	J	len	supp	F_c	J
adult	3.73	6992.47	**0.75**	**0.64**	3.93	7291.53	0.71	0.54	11.67	10950.60	0.50	0.19
anneal	4.93	133.13	**0.69**	0.55	3.27	136.13	**0.69**	**0.60**	11.93	175.47	0.43	0.14
breast	2.89	136.00	**0.70**	**0.55**	2.89	136.00	**0.70**	**0.55**	9.00	350.25	0.65	0.38
connect4	4.93	11442.40	**0.64**	**0.50**	4.67	12144.40	**0.64**	**0.50**	37.40	20210.00	0.34	0.13
cylBands	7.93	111.33	0.66	**0.50**	6.67	107.40	**0.67**	**0.50**	28.80	135.60	0.41	0.17
dematology	4.00	52.67	0.34	0.19	3.60	55.87	0.34	0.20	10.13	79.60	**0.51**	**0.22**
ecoli	3.38	62.50	0.46	0.30	2.62	42.92	0.45	0.29	5.87	41.73	**0.69**	**0.52**
flare	4.47	221.67	0.68	0.45	3.40	244.73	**0.76**	**0.64**	8.07	273.00	0.46	0.18
glass	3.93	31.07	**0.57**	**0.34**	3.73	29.40	0.54	0.32	6.73	30.60	0.50	0.19
heart	3.87	50.53	0.49	0.35	3.07	51.20	**0.53**	**0.40**	11.47	61.07	0.38	0.15
hepatitis	6.75	38.38	**0.76**	0.64	4.87	25.07	**0.76**	**0.65**	17.07	27.33	0.53	0.23
horseColic	5.20	69.13	0.59	0.38	3.73	66.80	**0.63**	**0.43**	9.80	81.00	0.46	0.16
ionosphere	13.93	58.93	0.72	0.58	10.27	65.07	**0.73**	**0.60**	24.33	53.00	0.58	0.25
iris	3.30	17.30	0.83	0.61	2.29	16.79	**0.94**	**0.79**	2.90	24.00	0.70	0.52
mushroom	6.20	1347.73	**0.65**	0.45	5.80	1443.93	0.63	**0.46**	16.53	1245.47	0.43	0.18
pageBlocks	4.57	790.00	0.86	0.81	4.22	616.00	0.86	0.81	9.67	398.93	**0.89**	**0.84**
penDigits	6.87	1660.00	**0.48**	**0.23**	5.53	1718.93	0.46	0.22	5.67	1858.40	0.46	0.21
pima	3.58	72.58	**0.69**	0.54	2.80	59.07	**0.69**	**0.55**	7.87	83.40	0.66	0.36
ticTacToe	2.87	159.5	**0.42**	**0.15**	2.00	153.27	0.30	0.12	2.07	246.33	0.36	0.13
waveform	5.00	894.13	**0.63**	**0.42**	4.53	945.40	**0.63**	**0.42**	16.20	999.80	0.36	0.15
wine	5.07	24.50	0.87	0.60	4.13	26.80	**0.89**	**0.65**	7.07	28.47	0.67	0.33
zoo	7.17	35.33	0.41	0.25	5.44	25.33	0.43	0.25	14.93	20.00	**0.85**	**0.73**
arit. mean	5.21	1109.15	0.63	0.46	4.25	1154.6	**0.64**	**0.48**	12.51	1698.82	0.54	0.29
			+16%	+58%			+18%	+65%				
geom. mean	4.83	166.98	**0.61**	0.42	3.96	158.53	**0.61**	**0.43**	10.18	186.95	0.52	0.24
			+17%	+75%			+17%	+83%				

are four features that are considered altogether only by our framework: (a) our model is specifically tailored for binary datasets, (b) our model allows for overlapping patterns, (c) our model minimizes the error with respect to the original dataset _and_ the encoding cost of the pattern set discovered, (d) frequent itemsets need not to be extracted from the original dataset. In the following, we illustrate in more detail the algorithms falling in the aforementioned categories.

Matrix decomposition based. The methods in this class aim at finding a product of matrices that describes the input data with a smallest possible amount of error. Probabilistic latent semantic indexing (PLSI) [10] is a well known technique that solves the above decomposition problem. PLSI was initially devised to model co-occurrence of terms in a corpus of documents $\mathcal{D}$. The core of PLSI is a generative model called _aspect model_, according to which occurrence of a words can be associated with an unobserved class variable $Z \in \mathcal{Z}$ called _aspect_, _topic_, or _latent variable_. The model generates each transaction t_i as follows. First, a topic $Z \in \mathcal{Z}$ is picked with probability $Z_T(i)$, and then an item j is picked with probability $Z_I(j)$. The whole database results from the contribution of every class variable, and thus we can write:

$$\mathcal{D} = \sum_{Z \in \mathcal{Z}} Z_T \cdot Z_I^\mathsf{T} + \mathcal{N} = \mathcal{Z}_T \mathcal{Z}_I + \mathcal{N}$$

where the matrices $\mathcal{Z}_T$ and $\mathcal{Z}_I$ result from the juxtaposition of the vectors Z_T and Z_I, for every $Z \in \mathcal{Z}$. An Expectation-Maximization algorithm can be designed to find $\mathcal{Z}_T$ and $\mathcal{Z}_I$ such that $\mathcal{N}$ is minimized.

This formulation is very similar to Definition 2. However, PLSI was not formulated for binary inputs: $\mathcal{D}$, $\mathcal{Z}_T$, $\mathcal{Z}_I$ and $\mathcal{N}$ are assumed to be real valued. Recall that the model was first designed to model occurrence of terms in a document, and thus multiple occurrences of the same term are allowed. This is not true for a binary, dataset, where an item can either occur or not. In other words, in one model pattern occurrences are _sum_-med, in the other they are _or_-ed.

Other similar approaches for non binary inputs have been studied, such as Latent Dirichlet allocation (LDA) [2], Independent Component Analysis (ICA) [11], Non-negative Matrix Factorization, etc. However, evaluations studies suggest that these models cannot be trivially adapted to binary datasets [9].

An improvement over these models has been proposed in [15] and [13]. In [15] a different generative model is proposed: an item occurs if it is generated by at least one class variable. This makes the model close to an _or_-ing of class variables. The proposed algorithm, called LIFT, is able to discover patterns when dominant items are present, i.e. items that are generated with high probability by one class variable only.

We already discussed the Discrete Basis Problem [13], and the proposed greedy algoirthm ASSO, which makes use of items' correlation statistics. This happens to be a limitation in many cases, since _global_ statistics may be too general to catch local correlations.

Both [15] and [13] aim at minimizing the noise matrix $\mathcal{N}$ only.

Database tiling. Database tiling algorithm are tailored for the discovery of large patterns in *binary* datasets. They differ on the notion of pattern adopted.

The maximum K-tiling problem introduced in [6] requires to find the set of K tiles, possibly overlapping, having the largest coverage of the given database $\mathcal{D}$. In this work, a tile is meant to be an pattern $\langle P_I, P_T \rangle$ such that if $P_T(i) = 1 \wedge P_I(j) = 1$ then $\mathcal{D}(i,j) = 1$. Therefore, this approach is not able to handle the noise present in the database.

Co-clustering [12] is a borderline approach between tiling and matrix decomposition. It is formulated as a matrix decomposition problem, and therefore its objective is to approximate the input data by minimizing $\mathcal{N}$. However, it does not allow for overlapping patterns. In this regards, its output is similar to a matrix block diagonalization.

According to [7], tiles can be hierarchical. A basic tile is indeed a hyper-rectangle with density p. A tile might contain several non overlapping sub-tiles, i.e., exceptional regions with larger or smaller density. Differently from our approach, low-density regions are considered as important as high density ones, and inclusion of tiles is preferred instead of overlapping.

Finally, the authors of [19] propose a model similar to the one described in Definition 2, but they assume that no noise is present in the dataset. Therefore, $\gamma_{\mathcal{N}}$ is always 0. The problem of finding Π is mapped to the problem of finding an optimal set covering of the input dataset. First frequent itemsets F_α are extracted from $\mathcal{D}$, with some minimum support threshold α. Each frequent itemset $F \in F_\alpha$ identifies a candidate pattern $C = \langle C_T, C_I \rangle$ such that $C_I(j) = 1 \Leftrightarrow a_j \in F$ and $C_T(i) = 1 \Leftrightarrow t_i \supseteq F$. The algorithm HYPER+ is greedy algorithm that first selects a number of candidates that cover the whole $\mathcal{D}$ with minimum cost, and then recursively merges the two candidates that introduce the smallest error, until only K patterns are left.

Differently from our approach, HYPER+ allows only the presence of false positives. False negatives are not allowed, since the input data must be entirely covered. Also, since only the extracted patterns contribute to the cost function, this is not affected by the number of false positives. For this reason HYPER+ tends to create a large number of false positives, and therefore patterns that poorly describe the input data. Consider the case $K = 1$, rather than finding the best pattern, HYPER+ will extract a single pattern that covers whole dataset, i.e. $P = \langle 1^M, 1^N \rangle$.

Frequent itemsets based. The frequent itemset mining problem requires to discover those itemsets supported by at least $\overline{\sigma}$ times in the database $\mathcal{D}$. Formally a pattern $P = \{P_I, P_T\}$ is frequent if $\|P_T\|_F \geq \overline{\sigma}$ and:

$$(5.1) \qquad P_T(i) = 1 \wedge P_I(j) = 1 \Rightarrow \mathcal{D}(i,j) = 1.$$

This notion of support does not allow missing items. Generalization of frequent itemsets for dealing with noisy databases, is typically achieved by relaxing the notion of support.

Weak error tolerant itemset (ETI) [3] are the first example of such a generalization. The pattern $P = \langle P_I, P_T \rangle$ is said to be weak ETI iff $\|P_T\|_F > \overline{\sigma}$ and the implication in Eq. 5.1 is violated at most $\epsilon \cdot \|P_I\|_F \cdot \|P_T\|_F$ times, where ϵ is an error tolerance threshold.

In extreme cases, a transaction may not contain any item of the pattern, and still be included in the supporting set. To avoid the possibility of spurious transactions, *strong* ETI are introduced such that and error tolerance threshold ϵ_r is enforced on any transactions/row of the database: t_i may support P if it contains at least $\epsilon_r \cdot \|P_I\|_F$ items of P.

Unfortunately, the enumeration of all weak/strong ETI requires to explore the full itemsets space without any pruning. To overcome this limitation of ETI mining algorithms, in [16] the notion of dense itemset is introduced. An itemset is said *dense* if it is a weak ETI and all of its non empty subsets are weak ETI. This sort of downward closure allows for an Apriori-like algorithm which can find all the dense itemsets in a database.

Approximate Frequent Itemsets (AFI) [17] are an extension of strong ETI, where a row-wise and a column-wise tolerance thresholds are enforced. The pattern $P = \langle P_I, P_T \rangle$ is valid if it is a strong ETI and every item i of P is supported by at least $\epsilon_c \cdot \|P_T\|_F$ transactions. In this case, a relaxed anti-monotone property can be exploited, and thus the solution set can be extracted without exploring the full itemset space.

In [4] the notion of closed approximate frequent itemsets (AC-AFI) is introduced. A pattern is said to be AC-AFI if (a) it is an AFI, (b) there is not superset supported by the same transactions, and (c) it encloses a *core pattern*. Given a pattern $P = \langle P_I, P_T \rangle$, a core pattern $C = \langle C_I, C_T \rangle$ is such that $C_T(i) = 1 \Rightarrow P_T(i) = 1$, $C_I(j) = 1 \Rightarrow P_I(j) = 1$ and $C_T(i) = 1 \wedge C_I(j) = 1 \Rightarrow \mathcal{D}(i,j) = 1$. In fact core patterns are also frequent pattern, they can be quickly extracted and extended by adding items and transactions as long as the resulting pattern is still an AC-AFI. We adopted the notion of core pattern in the formulation of the PANDA algorithm in Sec. 3.

Frequent itemets based algorithms require a demanding data processing, and, more importantly, they tend to generate a large and very redundant collection

of patterns. The cost model embedded in our framework implicitly limits the pattern explosion, since a large number of patterns does not minimizes the representation cost. The cost model may also allow to rank patterns according to the contributed cost reduction.

In [8] all of the above frequent itemsets based algorithms, plus some of their extensions are evaluated over a collection of synthetic datasets. This collection, which we also used to evaluate our algorithm, constitutes the first benchmark for approximate frequent itemsets mining algorithms.

6   Conclusion

In this paper we have discussed a new problem formulation for discovering the Top-K patterns from binary datasets. We have shown that this problem aims to decompose the matrix in term of patterns plus noise. The noise models not only false negative (1 bits that are not covered from any pattern) but also false positives (0 bits covered by a pattern).

Since a noisy input matrix can be decomposed in terms of several pattern sets and noise, we propose an encoding function that allows a cost to be associated with the patterns and the noise detected. This cost is used for evaluating the goodness of the set of patterns extracted, or, in an equivalent way, of the matrix decomposition. Finally, according to the MDL principle, we argue that the best pattern model is the one that minimize the overall cost function charactering the matrix decomposition. Solving our problem is complex, so we have proposed a new greedy algorithm, called PaNDa, which extracts the pattern model from a binary datasets, by minimizing the aforementioned cost model. We compared PaNDa with Asso, a greedy algorithm specifically tailored for binary datasets, inspired to a similar framework. Asso aims at finding a Boolean decomposition of a binary matrix, by minimizing a cost function that, unlike PaNDa, only accounts for noise. We argue that this approach may generate over-fitted models. In fact, minimizing only the noise cost can lead to patterns that try to cover the noisy 1 bits present in the data.

The experiments conducted on both synthetic and real-world datasets have showed that PaNDa outperforms Asso. PaNDa can better deal with noise in the data, due to its cost function that also accounts for the complexity of the patterns extracted.

In conclusion, our results show that the proposed cost model, which balances the amount of noise estimated in a dataset with the length of the description of the discovered patterns, is able to get close to the set of true patterns actually embedded in both synthetic and real-world datasets.

References

[1] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[3] C. Cheng, U. Fayyad, and P.S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *Proc. of KDD*, pages 194–203, 2001.

[4] H. Cheng, P. S. Yu, and J. Han. Ac-close: Efficiently mining approximate closed itemsets by core pattern recovery. In *Proc. of ICDM*, pages 839–844. IEEE Computer Society, 2006.

[5] F. Coenen. The lucs-kdd discretised/normalised arm and carm data library, 2003.

[6] F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. *Discovery Science*, pages 278–289, 2004.

[7] A. Gionis, H. Mannila, and J.K. Seppänen. Geometric and combinatorial tiles in 0-1 data. In *Proc. of PKDD*, pages 173–184, 2004.

[8] R. Gupta, G. Fang, B. Field, M. Steinbach, and V. Kumar. Quantitative evaluation of approximate frequent pattern mining algorithms. In *Proc. of KDD*, pages 301–309. ACM, 2008.

[9] Heli Hiisila and Ella Bingham. Dependencies between transcription factor binding sites: Comparison between ica, nmf, plsa and frequent sets. In *Proc. of ICDM*, pages 114–121, 2004.

[10] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proc. of SIGIR*, pages 50–57. ACM, 1999.

[11] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent component analysis*. John and Wiley, 2001.

[12] Tao Li. A general model for clustering binary data. In *Proc. of KDD*, pages 188–197. ACM, 2005.

[13] P. Miettinen, T. Mielikainen, A. Gionis, G. Das, and H. Mannila. The discrete basis problem. *IEEE TKDE*, 20(10):1348–1362, Oct. 2008.

[14] Jorma Rissanen. *Stochastic Complexity in Statistical Inquiry Theory*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1989.

[15] J.K. Seppänen, E. Bingham, and H. Mannila. A simple algorithm for topic identification in 0-1 data. In *Proc. of PKDD*, pages 423–434, September 2003.

[16] Jouni K. Seppänen and Heikki Mannila. Dense itemsets. In *Proc. of KDD*, pages 683–688, 2004.

[17] M. Steinbach, P.N. Tan, and V. Kumar. Support envelopes: a technique for exploring the structure of association patterns. In *Proc. of KDD*, 2004.

[18] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.

[19] Y. Xiang, R. Jin, D. Fuhry, and F. F. Dragan. Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In *Proc. of KDD*, pages 758–766. ACM, 2008.

[20] Mohammed J. Zaki and Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE TKDE*, 17(4):462–478, April 2005.