A Generative Pattern Model for Mining Binary Datasets

Claudio Lucchese* Salvatore Orlando† Raffaele Perego‡

October 26, 2009

Abstract

In many application fields, huge binary datasets modeling real life-phenomena are daily produced. The dataset records are usually associated with observations of some events, and people are often interested in mining these datasets in order to recognize recurrent patterns. However, the discovery of the most important patterns is very challenging. For example, these patterns may overlap, or be related only to a particular subset of the observations. Finally, the mining can be hindered by the presence of noise.

In this paper, we introduce a generative pattern model, and an associated cost model for evaluating the goodness of the set of patterns extracted from a binary dataset. We propose an efficient algorithm, named GPM, for the discovery of the patterns being most important according to the model. We show that the proposed model generalizes other approaches and supports the discovery of higher quality patterns.

1 Introduction

Huge binary datasets are generated daily by many popular applications for example in the electronic commerce, or mobile communications domains. These datasets are composed of transactions (or observations), each made up of a possibly large set of items (or 0-1 attributes). The attributes present in these transactions are in many cases not independent as they model real-life phenomena and events. *Patterns*, i.e., collections of items whose occurrences are somehow related to each other, can thus be mined from data in order to understand common behavior and derive interesting knowledge. Finding such patterns can be very challenging, in particular it is particularly complex to detect the most "important" ones (often known as *Top-k patterns*). In fact, two or more patterns can overlap, or a particular pattern could be related only to a particular subset of the observations, or it may be hidden by presence of noise.

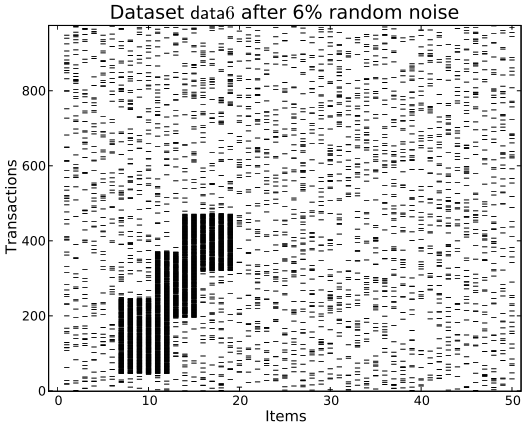

Figure 1: A synthetic binary dataset: row correspond to transactions, columns correspond to items.

In Figure 1 we show a simple synthetic dataset made of 50 items (columns) and 1000 transactions (rows). It is apparent that three overlapping patterns (i.e., three overlapping black rectangles) were embedded, consisting of three sets of items occurring together in three sets of transactions. Moreover, a number of bits where randomly flipped across the whole dataset. This simple intuition is not easy to formalize or to translate into an actual algorithm. Indeed, while binary matrices are omnipresent, no one of the algorithms proposed in the literature is able to discover that the dataset in Figure 1 simply contains three patterns plus noise.

Frequent itemset mining algorithms are not able to find those patterns, since due to the presence of noise, not all the items of a pattern may always occurs simultaneously in the set of supporting transac-

*ISTI "A.Faedo" - CNR, Pisa, Italy.

†Dept. of Computer Science, Univ. "Ca Foscari", Venice, Italy.

‡ISTI "A.Faedo" - CNR, Pisa, Italy.

tions/observations. A way to solve this issue is to relax the definition of support, as for Error Tolerant Itemsets (ETIs). But also ETIs have a number of drawbacks: they are too many, and they contain a lot of spurious itemsets containing only noisy occurrences.

Other approaches have been developed in the Information Retrieval field. Indeed, a corpus of document can be seen as a dataset whose transactions are documents, and items are words. This is different from binary datasets since a word may have multiple occurrences in a given document. Collection of documents are analyzed via a generative *topic* model, where a topic is a set of words co-occurring in a set of documents, i.e. a pattern. Matrix decomposition methods can be applied to find those hidden topics. The goal of generative topic models is to best approximate the input data. Therefore, also the noisy occurrences need to be explained by the model in terms of topics.

An interesting algorithm tailored to the discovery of patterns in binary databases is HYPER+ [15], which is based on the notion of *hyper-rectangle*, i.e., the Cartesian product between a set of items I and a set of transactions T. A set of hyper-rectangles is a *cover* of the database $\mathcal{D}$ if any item i occurring in a transaction t forms a couple (i, t) included in the Cartesian product of one of hyper-rectangles. Finally, each hyper-rectangles is associated with a cost. The HYPER+ algorithm aims at discovering the cover of the database of size K that minimizes the sum of the costs of its hyper-rectangles. HYPER+ produces a cover of $\mathcal{D}$ starting from a collection of frequent itemsets in the database.

In this paper we extend the framework introduced in [15], and propose a novel algorithm, named GPM, for the discovery of patterns in a noisy binary dataset, like the one shown in Figure 1.

Our work contains several original contributions. First, we introduce a new cost model for evaluating the goodness of the set of patterns extracted. Second, we propose GPM, a new algorithm that, unlike HYPER+, does not need to extract all the frequent patterns from the database in advance. GPM can also better deal with noise in the data, since it can 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 GPM outperforms HYPER+ both in terms of running time and quality of the extracted patterns.

2 Problem Statement

Our input data is a transactional dataset, which is represented as an $N \times M$ *binary matrix* denoted as $\mathcal{D}$.

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 adopt a generative model to describe the observed dataset. In particular, we assume that each transaction of $\mathcal{D}$ records the occurrence of one or more *latent patterns*, with the addition of some noise.

Let Ω be the latent pattern set of unknown size. Every pattern $P \in \Omega$ corresponds to two sets: the former set specifies which items were generated by the pattern, and the latter which transactions were affected by the pattern. Each pattern $P \in \Omega$ is formally defined as $P = \langle P_I, P_T \rangle$, where $P_I \in \{0,1\}^M$ and $P_T \in \{0,1\}^N$. Therefore, $P_I(j) = 1$ *iff* item a_j belongs to the pattern, and $P_T(i) = 1$ *iff* the pattern P participates in the generation of t_i.

We can exemplify the generative process of a given transaction t_i as follows:

- $\forall\ P \in \Omega$, such that $P_T(i) = 1$, item a_j is generated in t_i iff $P_I(j) = 1$;
- eventually, some items in $\mathcal{I}$ are randomly added to (removed from) t_i due to noise.

The resulting observed t_i corresponds to the i^{th} row of matrix $\mathcal{D}$.

This model has many similarities with other methods such as probabilistic latent semantics analysis, factor analysis, and topic models, where the keyword pattern is replaced by *aspect*, *unobserved class variable*, *latent topic* or *factor*. But, differently from those models, we focus on noisy binary data.

The binary representation of datasets and patterns helps us in explaining algebraically this generative process of the observed dataset $\mathcal{D}$, which is indeed obtained as the result of the "OR sum" of the occurrences of the patterns Ω and some noise.

Definition 2 (Generative Pattern Model)
Let $\mathcal{D}$ be the binary representation of the observed input data. It is generated by a set of patterns Ω as follows:

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

where $\bigvee$ is the OR operation, $P_T \cdot P_I^{\mathsf{T}}$ is the outer product of the two binary vectors, and 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*, and acts by randomly flipping bits after the "OR sum" of the patterns' occurrences.*

We call *ground truth* how the dataset would look like if noise was not present, i.e. if $\mathcal{D}$ was generated by the pattern set only. The gound truth deriving from the pattern set Ω is:

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

Therefore noise can be computed as $\mathcal{N} = \mathcal{D} \oplus \overset{\triangle}{\Omega}$.

Since we can only observe $\mathcal{D}$, there may exist several pattern sets and noise matrices that can "explain" the input data, according to Definition 2. However, we argue that we can approximate Ω by choosing the "best" pattern set Π on the basis of the Minimum Description Length (MDL) principle [11]. When choosing among different candidate models, which are somehow encoded, the MDL principle suggests to adopt the model with the minimum code length. To this end, we define two encoding cost functions $\gamma_P : \{0,1\}^N \times \{0,1\}^M \to \mathbb{R}$ and $\gamma_{\mathcal{N}} : \{0,1\}^{N \times M} \to \mathbb{R}$, and require the selected pattern set Π and the resulting noise matrix to have the minimum costs. Finally, we can introduce our *generative pattern model discovery problem.*

Problem 1 (Generative Pattern Model Discovery Problem)
Given a binary dataset $\mathcal{D} \in \{0,1\}^{N \times M}$, find the pattern set Π that minimizes the following cost:

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

where $\mathcal{N} = \mathcal{D} \oplus \overset{\triangle}{\Pi}$, being $\overset{\triangle}{\Pi}$ the estimated ground truth*, $\gamma_P(X,Y) = \|X\|_1 + \|Y\|_1$ and $\gamma_{\mathcal{N}}(\mathcal{Z}) = \sum_{i,j} \mathcal{Z}(i,j)$.*
Π is a generative pattern model of the observed dataset $\mathcal{D}$.

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 1s is present in $\mathcal{D}$. The mining algorithm has to decide whether to consider this rectangle being noise or a 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 half-perimeter (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 supposingly noise.

Regarding noise, the model allows both *false positives* and *false negatives* 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:

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

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

Most approaches require every item occurring in $\mathcal{D}$ to be covered, and therefore the do not allow false negatives. This is crucial in presence of noise, since every noisy occurrence needs to be matched, clearly facing overfitting issues.

The Pattern Model Discovery Problem, extends the mining problem described by the authors of Hyper+ [15]. They propose a model similar to the one described in Definition 2, but they assume that no noise is present in the dataset. Therefore, while γ_P is the same, $\gamma_{\mathcal{N}}$ is always 0. Therefore, in their approach 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 $C_T(i) = 1 \Leftrightarrow F \subseteq t_i$.

Hyper+ is a greedy algorithm that first selects the 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.

Unlike our approach, Hyper+ only allows false positives to occur. 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.

Our model generalizes the framework introduced in [15], by borrowing from related approaches for matrix decomposition problems. These usually try to find a set of patterns that may approximate as well as possible the original data $\mathcal{D}$. They thus take into consideration the cost $\gamma_{\mathcal{N}}$, and discard γ_P. For example, consider [10] where the authors adopt our $\gamma_{\mathcal{N}}$ function, but use a cost function $\gamma_P(\cdot) = 0$ for the extracted patterns.

We show that, by taking into account both cost components, it is possible to dramatically improve the quality of the patterns found by HYPER+.

Section 4 proposes GPM, a new algorithm for the Pattern Model Discovery Problem that, unlike HYPER+, does not need to extract all the frequent patterns from the database in advance. Also, GPM can produce a better pattern set, since it is able to face noise in the data by dealing with both false positives and negatives.

3 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 Generative Pattern Model discovery problem. However, there 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) [7] 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 Def. 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) [1], Independent Component Analysis (ICA) [8], Non-negative Matrix Factorization, etc. However, evaluations studies suggest that these models cannot be trivially adapted to binary datasets [6].

An improvement over this models has been proposed in [12] and [10]. In [12] 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.

In [10], the authors formalize the Discrete Basis Problem, which aims at finding a binary valued matrix decomposition of a binary dataset. The formulation is equivalent to Def. 2. The authors proposed a greedy algorithm where candidate patterns are created in advance on the basis 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 [12] and [10] 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 [3] 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 [9] 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 [4], 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.

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 $\|P_T\|_1 \geq \overline{\sigma}$ and:

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

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) [16] are the first example of such a generalization. The pattern $P = \langle P_I, P_T \rangle$ is said to be weak ETI iff $\|P_T\|_1 > \overline{\sigma}$ and the implication in Eq. 1 is violated at most $\epsilon \cdot \|P_I\|_1 \cdot \|P_T\|_1$ 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 $\epsilon_r \cdot \|P_I\|_1$ 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 [13] 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) [14] 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 $\epsilon_c \cdot \|P_T\|_1$ 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 [2] 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 adopt the notion of core pattern in the formulation of the GPM algorithm in Sec. 4.

All the above 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 [5] all of the above frequent itemsets based algorithms, plus some of their extensions are evaluated over a collection of synthetic datasets. This constitutes the first benchmark for approximate frequent itemsets mining algorithm. In Section 5 we illustrate these datasets and the evaluation measure adopted. We used the same datasets and the same measures to test the goodness of the patterns extracted by our algorithm.

4 Algorithm GPM

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.

Algorithm 1 GPM 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($\mathcal{D}_R$, Π, $\mathcal{D}$)
5: $C^+ \leftarrow$ EXTEND-CORE(C, E, Π, $\mathcal{D}$)
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**

To tackle such a large search space, we propose a greedy algorithm named GPM. It adopts two heuristics. First the problem of discovering a pattern is decomposed into two simpler problems: discovering a *core pattern* and extending it to form a good approximate pattern. Seconds, 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 [2], 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, GPM adds items and transactions to C, until the largest extension of C that minimizes the overall cost γ is found.

While HYPER+ needs the collection of frequent itemsets in the dataset as a starting point for the discovery of Ω, GPM is able to avoid this expensive step, by directly mining the input data.

Algorithm 1 gives an overview of the GPM 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 discovered (line 4) and then it is extended to form a new pattern C^+ (line 5) that is added to the current pattern set Π. These two steps are described in detail in the following sections.

A user-provided parameter K has been introduced to make GPM comparable to HYPER+. However, we have also included in GPM a check that stops the generation of 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 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 $X_T(i) = 1$ and $X_I(j) = 1$ for some previously discovered pattern X. 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.

4.1 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.

We restrict the search space, which has size 2^M, by sorting the items in $\mathcal{D}$ and considering them one by one in the descending frequency order.

The procedure is described in Algorithm 2. The extension list E is initialized to the empty set, while core pattern C is initialized with the most frequent item j_1 and its supporting transactions in $\mathcal{D}_R$. Then items in $\mathcal{D}_R$ are processed one by one in decreasing support order (line 3). The current item j_h is used to create a new candidate pattern $C*$ (lines 7-9). Note that when we add an item to a pattern (line 8), as a consequence we can have a reduction in the number of supporting transactions (line 9). If C^* reduces the cost of the pattern set with respect to C, then C^* is promoted to be the new candidate (line 11) and it is used in the subsequent iteration. Otherwise, i_h is appended to the extension list E (line 13), 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. 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. 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 $\mathcal{D}_R$. A prefix-tree allow 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.

Algorithm 2 Core Pattern discovery.

1: **function** FIND-CORE($\mathcal{D}_R$, Π, $\mathcal{D}$)
2: $E \leftarrow \emptyset$ ▷ Extension list
3: $\{j_1, \ldots, j_M\} \leftarrow$ SORT-ITEMS-BY-SUPP($\mathcal{D}_R$)
4: $C \leftarrow \langle C_T, C_I \rangle$, where C_I and C_T are zero-vectors.
5: $C_I(j_1) \leftarrow 1$, and $C_T(i) \leftarrow 1 \; \forall i$ s.t. $\mathcal{D}_R(i, j_1) = 1$
6: **for all** $h \leftarrow 2, \ldots, M$ **do**
7: $C^* \leftarrow C$ ▷ create a new candidate
8: $C^*_I(j_h) \leftarrow 1$
9: $C^*_T(i) \leftarrow 0 \; \forall i$ s.t. $\mathcal{D}_R(i, j_h) = 0$
10: **if** $\gamma(\Pi \cup C^*, \mathcal{D}) \leq \gamma(\Pi \cup C, \mathcal{D})$ **then**
11: $C \leftarrow C^*$
12: **else**
13: E.append(j_h)
14: **end if**
15: **end for**
16: **return** C, E
17: **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: $C^*_T(i) = 1$
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**

4.2 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 correctly a number of items that were previously considered as noise, thus reducing the amount of noise. 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 instance, suppose that C^* does not introduce any noise, and that all the item 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 to C^*. This data structure is updated after a new pattern is discovered, in order to distinguish items that are already covered by the current pattern set Π.

5 Experiments

5.1 Evaluation

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

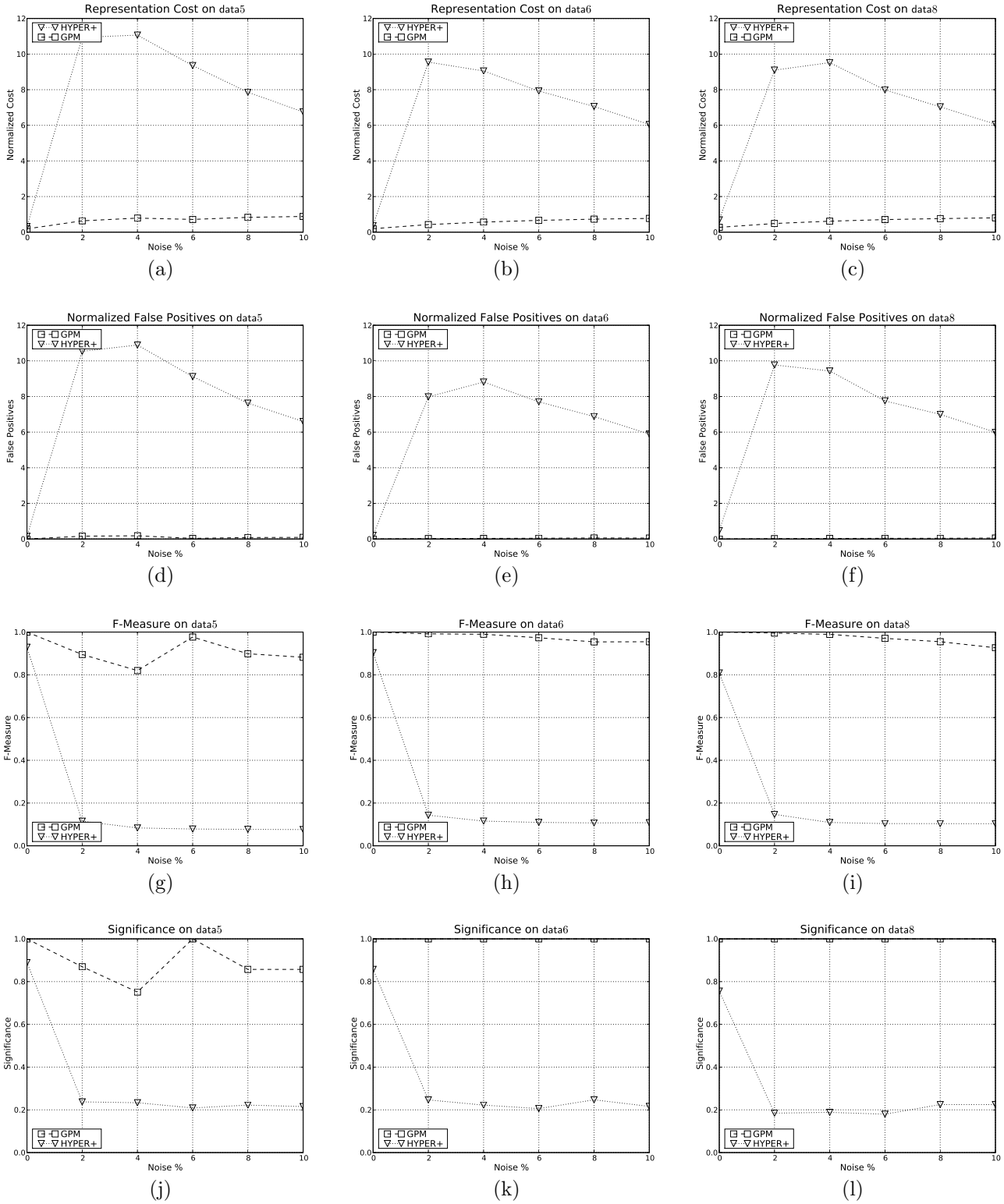

Figure 2: Comparison between GPM and HYPER+.

therefore there is no *ground truth* that we can use as a benchmark. However, in [5], a collection of synthetic datasets, and a set of evaluation measures is proposed to evaluate the goodness of the itemsets extracted in presence of noise. The idea is to synthesize a dataset by embedding a given set of patterns – i.e., our *ground truth* Ω – and then introduce noise by flipping some bits in the corresponding binary representation. Since the patterns occurring in the dataset are known, a number of evaluation measures can be used to compare the patterns extracted with the embedded true patterns. Due to space constraints, we use only 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 some overlaps between the embedded patterns. All the datasets have 50 items and 1000 transactions.

These synthetic datasets are pretty simple, and we cannot use them to state the high-quality of an algorithm. Nevertheless they carry important features. First, they allow for repeatability of experiments. Second, they can be used to detect and understand the weaknesses of an algorithm.

The authors of [5] propose four evaluation measures: *Recoverability*, *Spuriousness*, *Significance* and *Redundancy*. Recoverability and Spuriousness are similar to the measures Recall and Precision used in Information Retrieval. Significance is the harmonic mean of the first two, analogously to F-Measure. Redundancy measures the overlaps between patterns. Redundancy is not very useful in our case, since patterns have overlaps. In our experiments, we will consider only Significance, since it is a summary of the first two. For a detailed description of the datasets and the evaluation measures, an interested reader can refer to [5].

Note that all of the above evaluation measures only take into account the items of a discovered pattern, i.e. only P_I for each pattern P. For the sake of completeness, we are interested in evaluating an additional quality measure, which takes into account both the sets P_I and P_T. Let Ω be the set of true patterns embedded in the dataset, and let Π the set of patterns extracted by the mining algorithm, 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 the above does not imply $\mathcal{D}(i,j) = 1$.

We adapt the usual precision and recall to this setting, and compute the F-Measure F as follows:

$$Prec(\Pi) = \frac{\sum_{i,j} \overset{\triangle}{\Pi}(i,j) \cdot \overset{\triangle}{\Omega}(i,j)}{\sum_{i,j} \overset{\triangle}{\Pi}(i,j)}$$

$$Rec(\Pi) = \frac{\sum_{i,j} \overset{\triangle}{\Pi}(i,j) \cdot \overset{\triangle}{\Omega}(i,j)}{\sum_{i,j} \overset{\triangle}{\Omega}(i,j)}$$

$$F(\Pi) = \frac{2 \cdot Prec(\Pi) \cdot Rec(\Pi)}{Prec(\Pi) + Rec(\Pi)}$$

5.2 Results

In Figures 2(a,b,c) we report the representation costs obtained by varying the noise amount. The cost is normalized by the amount of 1 bits in the binary dataset, i.e. $\sum_{i,j} \mathcal{D}(i,j)$. Both the two algorithms HYPER+ and GPM were asked to extract K patterns, where K was the actual number of patterns embedded in the dataset. Regarding HYPER+, we set the minimum support threshold for the frequent itemsets extraction to 5%.

It is apparent that the representation produced by HYPER+ is definitely larger, and it is even larger than $\sum_{i,j} \mathcal{D}(i,j)$ whenever some noise in added to the dataset. HYPER+ only minimizes γ_P, without considering $\gamma_{\mathcal{N}}$. In presence of noise, the discovered set of patterns produces a tremendous amount of false positives, whose effect is the increase in the total cost. In Figures 2(d,e,f) we report the number of false positives produced by the two algorithms, again normalized by $\sum_{i,j} \mathcal{D}(i,j)$. Such weakness of the HYPER+ also comes from the fact that it requires to cover every single bit in $\mathcal{D}$, i.e. it does not allow false negatives.

Since the representation cost may not be a sound indicator of the goodness of a pattern mining algorithm, we also report their Significance and F-Measure. Recall that significance only takes into account the items (P_I) of each pattern, while the F-Measure considers the supporting transactions (P_T) as well. In Figures 2(g-l) we show that GPM largely outperforms GPM both in terms of significance and F-Measure.

Finally, in Figure 5.2 we show the actual patterns extracted by GPM from data6 after 6% random flips. The dataset is the same as the one shown in Figure 1. The GPM algorithm was able to discover the three

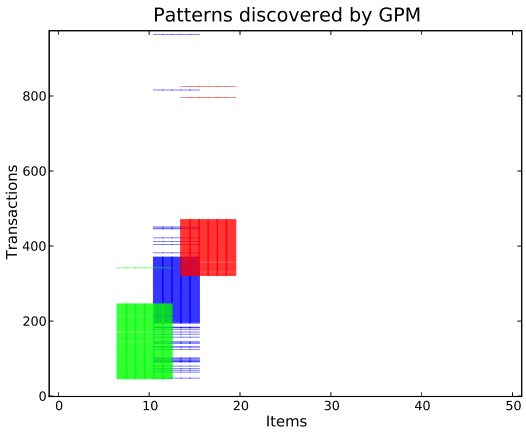

Figure 3: Patterns extracted by GPM from data6

embedded patterns and their supporting transaction. Only a few spurious transactions were added, because they contain large portion of patterns' items due to noise.

Due to space constraints, we shortly report that the runing time of GPM is within the tenths of a second on every dataset, while HYPER+ may require tens of second because of the frequent itemsets extraction and their processing.

6 Conclusions

In this paper we showed that a binary dataset can be described with a *generative pattern model*. Since several pattern sets may support a generative pattern model for the same dataset, we proposed to use the MDL principle to choose the best among them. We thus defined a novel cost model for evaluating the goodness of the set of patterns extracted, and we proposed a new algorithm, named GPM for the discovery of patterns in binary datasets minimizing the aforementioned cost model.

We compared GPM with HYPER+, an algorithm specifically tailored for binary datasets, and inspired to a similar framework. GPM, unlike HYPER+, does not need to extract all the frequent patterns from the database in advance. GPM can also better deal with noise in the data since it can deal with both *false positives* and *false negatives*. Finally, we show that GPM outperforms HYPER+ both in terms of running time and quality of the extracted patterns.

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 several synthetic datasets.

References

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

[2] Hong Cheng, Philip S. Yu, and Jiawei Han. Ac-close: Efficiently mining approximate closed itemsets by core pattern recovery. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 839–844. IEEE Computer Society, 2006.

[3] Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. *Discovery Science*, pages 278–289, 2004.

[4] Aristides Gionis, Heikki Mannila, and Jouni K. Seppänen. Geometric and combinatorial tiles in 0-1 data. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 173–184, 2004.

[5] Rohit Gupta, Gang Fang, Blayne Field, Michael Steinbach, and Vipin Kumar. Quantitative evaluation of approximate frequent pattern mining algorithms. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 301–309. ACM, 2008.

[6] Heli Hiisila and Ella Bingham. Dependencies between transcription factor binding sites: Comparison between ica, nmf, plsa and frequent sets. In *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 114–121, 2004.

[7] Thomas Hofmann. Probabilistic latent semantic indexing. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

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

[9] Tao Li. A general model for clustering binary data. In *KDD '05: Proceeding of the 11th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 188–197. ACM, 2005.

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

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

[12] Jouni K. Seppänen, Ella Bingham, and Heikki Mannila. A simple algorithm for topic identification in 0-1 data. In *European Conference on Principles and Practice of Knowledge Discovery in Databases: PKDD*, pages 423–434, September 2003.

[13] Jouni K. Seppänen and Heikki Mannila. Dense itemsets. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 683–688, 2004.

[14] Michael Steinbach, Pang-Ning Tan, and Vipin Kumar. Support envelopes: a technique for exploring the structure of association patterns. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 296–305, 2004.

[15] Yang Xiang, Ruoming Jin, David Fuhry, and Feodor F. Dragan. Succinct summarization of transactional databases: an overlapped hyperrectangle scheme. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 758–766. ACM, 2008.

[16] Cheng Yang, Usama Fayyad, and Paul S. Bradley. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 194–203, 2001.