An Exhaustive Covering Approach to Parameter-free Mining of Non-redundant Discriminative Itemsets

Yoshitaka Kameya

Dept. of Information Engineering, Meijo University
1-501 Shiogama-guchi, Tenpaku-ku, Nagoya 468-8502, Japan
ykameya@meijo-u.ac.jp

Abstract. Discriminative pattern mining is a promising extension of frequent pattern mining. This paper proposes an algorithm called ExCover, a shorthand for exhaustive covering, for finding non-redundant discriminative itemsets. ExCover outputs non-redundant patterns where each pattern covers best at least one positive transaction. With no control parameters limiting the search space, ExCover efficiently performs an exhaustive search for best-covering patterns using branch-and-bound pruning. During the search, candidate best-covering patterns are concurrently collected for each positive transaction. Formal discussions and experimental results exhibit that ExCover efficiently finds a more compact set of patterns in comparison with previous methods.

Keywords: discriminative patterns, sequential covering, branch-and-bound search

1 Introduction

Discriminative pattern mining is a promising extension of frequent pattern mining, which has been studied under several different names [6, 15]. In a typical setting of discriminative pattern mining, each transaction belongs to one of two or more pre-defined classes, and we are interested in a particular class c. For example, we may be studying mushrooms which are edible. Then, we attempt to find discriminative patterns that frequently occur in the transactions of class c (called *positive* transactions) and do not frequently occur in the transactions not of class c (called *negative* transactions). The patterns found are expected to characterize well c, the class of interest.

A main difficulty of discriminative pattern mining is that the quality score for a pattern is *not* anti-monotonic w.r.t. set-inclusion, and thus we cannot enjoy powerful pruning like the one with the minimum support threshold. In the literature, several authors have conducted branch-and-bound search (e.g. [19]). Another difficulty is redundancy among patterns. For example, if a pattern $\{\mathsf{A}\}$ has significantly high quality in characterizing a class c of interest, patterns including A, such as $\{\mathsf{A}, \mathsf{B}\}$, $\{\mathsf{A}, \mathsf{C}\}$ and $\{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, also tend to have high quality.

One remedy against redundancy is to use set-inclusion-based constraints among patterns. For example, closed pattern mining techniques [20, 24] have also been exploited in discriminative pattern mining [8, 22]. In the studies on subgroup discovery, the notion of irrelevant patterns has been introduced [8]. It is known that the productivity constraint [2] is efficiently tested in the search over suffix enumeration trees [1, 14, 16].

In this paper, we propose an algorithm called ExCover, which is a shorthand for *exhaustive covering*. ExCover outputs non-redundant patterns where each pattern $\boldsymbol{x}$ covers best at least one positive transaction t, namely, $\boldsymbol{x}$ is of the highest quality among the patterns that cover t. Hereafter we call this constraint the *best-covering* constraint. ExCover efficiently performs an exhaustive, depth-first search for best-covering patterns using branch-and-bound pruning techniques developed so far. During the search, candidate best-covering patterns are concurrently collected for each positive transaction.

The merit of ExCover is three-fold. First, ExCover outputs patterns that are non-redundant from the viewpoint of coverage over positive transactions. In previous work, HARMONY [25] employs the same strategy, and surely covering all positive transactions is counted as an advantage in its original paper. In fact, however, the best-covering constraint also works for removing redundancy — a formal result presented in this paper is that the best-covering constraint is tighter than productivity, the aforementioned constraint ensuring non-redundancy. The second merit of ExCover is that it finds patterns in an exhaustive manner. In practice, we occasionally use rule learners for knowledge discovery, rather than for black-box classification. Unfortunately, however, most of traditional learners rely on greedy search and some heuristics [7, 26], and consequently, for some generated rules, the reason why such rules have been generated may not be clear to non-experts. We believe that the exhaustiveness of ExCover would enhance the explainability of the obtained results. Lastly, ExCover has no control parameters required to be tuned for limiting the search space. In frequent pattern mining, it is often said to be tedious or infeasible to tune the minimum support threshold for having a manageable amount of useful patterns. To alleviate this inconvenience, Han et al. [13] proposed a top-k mining method without tuning minimum support. Here k, the number of patterns to be output, is more user-centric than minimum support. ExCover follows this line of research and does not even require k since it automatically stops the search when every positive transaction has been covered best by some pattern already examined.

To illustrate, let us consider a dataset shown in Table 1 (1). There are five positive transactions (belonging to class +) and five negative transactions (belonging to class −). Each transaction is associated with an identifier called TID. From this dataset, we attempt to find top-k discriminative patterns w.r.t. class $c = +$ with $k = 20$ and obtain the patterns in Table 1 (2a). The quality of each pattern in the first field is measured by the F-score recorded in the second field. The third field contains the TIDs of positive transactions covered by the pattern in the first field. Table 1 (2a) includes more than 20 patterns due to the tie score at the bottom. For example, for class $c = +$ and the top-ranked pattern

Table 1. Example transactions (1), and discriminative patterns (2a)∼(2d) and (3).

(1)

TID	Class	Transaction
1	+	{A, B, D, E}
2	+	{A, B, C, D, E}
3	+	{A, C, D, E}
4	+	{A, B, C}
5	+	{B}
6	−	{A, B, D, E}
7	−	{B, C, D, E}
8	−	{C, D, E}
9	−	{A, D, E}
10	−	{A, D}

(2a) Top-k

Pattern	F-score	TIDs
{A, C}	0.750	2,3,4
{B}	0.727	1,2,4,5
{A}	0.667	1,2,3,4
{A, B}	0.667	1,2,4
{A, D, E}	0.600	1,2,3
{A, E}	0.600	1,2,3
{C}	0.600	2,3,4
{A, B, C}	0.571	2,4
{A, C, D}	0.571	2,3
{A, C, D, E}	0.571	2,3
{A, C, E}	0.571	2,3
{A, D}	0.545	1,2,3
{A, B, D}	0.500	1,2
{A, B, D, E}	0.500	1,2
{A, B, E}	0.500	1,2
{B, C}	0.500	2,4
{B, D}	0.444	1,2
{B, D, E}	0.444	1,2
{B, E}	0.444	1,2
{C, D}	0.444	2,3
{C, D, E}	0.444	2,3
{C, E}	0.444	2,3

(2b) Productive

Pattern	F-score	TIDs
{A, C}	0.750	2,3,4
{B}	0.727	1,2,4,5
{A}	0.667	1,2,3,4
{C}	0.600	2,3,4

(2c) Closed

Pattern	F-score	TIDs
{A, C}	0.750	2,3,4
{B}	0.727	1,2,4,5
{A}	0.667	1,2,3,4
{A, B}	0.667	1,2,4
{A, D, E}	0.600	1,2,3
{A, B, C}	0.571	2,4
{A, C, D, E}	0.571	2,3
{A, B, D, E}	0.500	1,2
{A, B, C, D, E}	0.333	2

(2d) Productive & Closed

Pattern	F-score	TIDs
{A, C}	0.750	2,3,4
{B}	0.727	1,2,4,5
{A}	0.667	1,2,3,4

(3) Best-covering (ExCover)

Pattern	F-score	TIDs
{A, C}	0.750	2,3,4
{B}	0.727	1,2,4,5

$\boldsymbol{x} = \{\mathsf{A}, \mathsf{C}\}$, we have its (positive) support $p(\boldsymbol{x} \mid c) = 0.6$ since three out of five positive transactions are covered by {A, C}. Similarly, we have $\boldsymbol{x}$'s confidence $p(c \mid \boldsymbol{x}) = 1$ since all transactions covered by {A, C} belong to class +. The F-score of {A, C} is then obtained as the harmonic mean of support $p(\boldsymbol{x} \mid c)$ and confidence $p(c \mid \boldsymbol{x})$ which amounts to $2 \times 0.6 \times 1/(0.6 + 1) = 0.75$. Remark here that the patterns in Table 1 (2a) have been obtained under no constraint except the top-k constraint, and more patterns would be output with a larger k.

In contrast, the patterns in Table 1 (2b)∼(2d) are obtained under some additional, set-inclusion-based constraints. Specifically, Table 1 (2b) only contains productive patterns, i.e. the patterns having no improvement in quality from their sub-patterns are excluded. For example, the patterns in Table 1 (2a) containing item B together with some other items are all excluded in Table 1 (2b), since their F-scores are lower than the F-score of pattern {B}. On the other hand, Table 1 (2c) contains the patterns closed on the positive transactions. One may find that the patterns in Table 1 (2a) that cover the same positive transactions are replaced with the largest pattern listed in Table 1 (2c). For example, patterns $\{\mathsf{A},\mathsf{D}\}$, $\{\mathsf{A},\mathsf{E}\}$ and $\{\mathsf{A},\mathsf{D},\mathsf{E}\}$ in Table 1 (2a) cover the same positive transactions 1, 2 and 3, whereas Table 1 (2c) only contains the largest one $\{\mathsf{A},\mathsf{D},\mathsf{E}\}$. Choosing productive patterns from the closed patterns in Table 1 (2c) yields the patterns in Table 1 (2d). Note that, even with a larger k, no other patterns will be output, and therefore additional constraints surely work for reducing the number of patterns to be output.

Furthermore, we wish to have fewer patterns that are sufficient to characterize the class of interest. Indeed, in the current example, ExCover only outputs two patterns listed in Table 1 (3). Pattern $\{\mathsf{A}\}$ has been excluded here since each of transaction 1, 2, 3 and 4 covered by $\{\mathsf{A}\}$ is also covered by other patterns $\{\mathsf{A},\mathsf{C}\}$ and $\{\mathsf{B}\}$ of higher quality. We say that $\{\mathsf{A},\mathsf{C}\}$ covers best transactions 2, 3 and 4, and $\{\mathsf{B}\}$ covers best transactions 1 and 5. From the viewpoint of coverage over positive transactions, in the current example, performing top-k mining where $k = 1$ seems inappropriate, since we obviously lose the information from transactions 1 and 5. Having that said, however, we are not able to know it beforehand. So eliminating k in ExCover would reduce the user's effort.

The rest of the paper is outlined as follows. First, Section 2 gives several background notions and notations related to ExCover. We then describe the details of ExCover in Section 3. Section 4 presents some results of our experiments, and Section 5 discusses some related work. Finally Section 6 concludes the paper.

2 Background

2.1 Preliminaries

This paper shares several background notions and notations with [14]. We first consider a dataset $\mathcal{D} = \{t_1, t_2, \ldots, t_N\}$, a multiset of size N, where t_i $(1 \leq i \leq N)$ is a set of items called a transaction. Each transaction belongs to one of predefined classes $\mathcal{C}$, and let c_i be the class of transaction t_i. The set of all items appearing in $\mathcal{D}$ is denoted by $\mathcal{X}$. A pattern $\boldsymbol{x}$ is a subset of $\mathcal{X}$. and we say that $\boldsymbol{x}$ *covers* a transaction t_i when $\boldsymbol{x} \subseteq t_i$. For convenience, we interchangeably denote a pattern as a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, as a set $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$, or as a conjunction $\boldsymbol{x} = (x_1 \wedge x_2 \wedge \ldots \wedge x_n)$.

We then define some subsets of a dataset $\mathcal{D}$: $\mathcal{D}_c = \{t_i \mid c_i = c, 1 \leq i \leq N\}$, $\mathcal{D}(\boldsymbol{x}) = \{t_i \mid \boldsymbol{x} \subseteq t_i, 1 \leq i \leq N\}$ and $\mathcal{D}_c(\boldsymbol{x}) = \{t_i \mid c_i = c, \boldsymbol{x} \subseteq t_i, 1 \leq i \leq N\}$, where $c \in \mathcal{C}$ is the class of interest. We use a symbol $\neg$ for negation, e.g. $\mathcal{D}_{\neg c} = \mathcal{D} \setminus \mathcal{D}_c$, $\mathcal{D}_c(\neg\boldsymbol{x}) = \mathcal{D}_c \setminus \mathcal{D}_c(\boldsymbol{x})$ and $\mathcal{D}_{\neg c}(\boldsymbol{x}) = \mathcal{D}(\boldsymbol{x}) \setminus \mathcal{D}_c(\boldsymbol{x})$. The transactions in $\mathcal{D}_c$ (resp. $\mathcal{D}_{\neg c}$) are called positive (resp. negative) transactions.

The probabilities treated in this paper are all empirical, i.e. they are computed from the dataset $\mathcal{D}$. Specifically, a joint probability $p(c, \boldsymbol{x})$ is obtained as $|\mathcal{D}_c(\boldsymbol{x})|/N$. Similarly we have $p(c, \neg\boldsymbol{x}) = |\mathcal{D}_c(\neg\boldsymbol{x})|/N$, $p(\neg c, \boldsymbol{x}) = |\mathcal{D}_{\neg c}(\boldsymbol{x})|/N$, and so on. Using joint probabilities, marginal probabilities and conditional probabilities are computed in a standard way, e.g. $p(\boldsymbol{x}) = p(c, \boldsymbol{x}) + p(\neg c, \boldsymbol{x})$, $p(c) = p(c, \boldsymbol{x}) + p(c, \neg\boldsymbol{x})$ or $p(c \mid \boldsymbol{x}) = p(c, \boldsymbol{x})/p(\boldsymbol{x})$. In the traditional terminology on pattern mining, we often call conditional probabilities $p(\boldsymbol{x} \mid c)$, $p(\boldsymbol{x} \mid \neg c)$ and

$p(c \mid \boldsymbol{x})$ positive support, negative support and confidence, respectively. For brevity, 'support' means positive support unless explicitly noted.

2.2 Dual-monotonicity

The quality of a pattern $\boldsymbol{x}$ for class c is written as $R_c(\boldsymbol{x})$, and most of popular quality functions are functions of positive support $p(\boldsymbol{x} \mid c)$ and negative support $p(\boldsymbol{x} \mid \neg c)$ [14]. As an instance of R_c, throughout the paper, we use *F-score* $\mathrm{F}_c(\boldsymbol{x}) = 2p(c \mid \boldsymbol{x})p(\boldsymbol{x} \mid c)/(p(c \mid \boldsymbol{x}) + p(\boldsymbol{x} \mid c))$, which can be simplified as Dice Index $2p(\boldsymbol{x}, c)/(p(\boldsymbol{x}) + p(c))$. Also F-score gives the same ranking over patterns as the one by Jaccard Index $p(\boldsymbol{x}, c)/(p(\boldsymbol{x}) + p(c) - p(\boldsymbol{x}, c))$ for a given class c of interest. Since we seek for the patterns characterizing a particular class c, we focus on the patterns $\boldsymbol{x}$ such that $p(\boldsymbol{x} \mid c) \geq p(\boldsymbol{x} \mid \neg c)$ or equivalently $p(c \mid \boldsymbol{x}) \geq p(c)$.[1] In the previous work, the *convexity* of quality scores has been exploited in branch-and-bound pruning [19], and recently, a relaxed condition called *dual-monotonicity* was introduced in [14]:

Definition 1. *Let R_c be a quality score for a class c. Then, R_c is dual-monotonic iff, for any pattern $\boldsymbol{x}$, $R_c(\boldsymbol{x})$ is monotonically increasing w.r.t. $p(\boldsymbol{x} \mid c)$ and monotonically decreasing w.r.t. $p(\boldsymbol{x} \mid \neg c)$ wherever $p(\boldsymbol{x} \mid c) \geq p(\boldsymbol{x} \mid \neg c)$.* □

ExCover works with any dual-monotonic quality score. Indeed, like the algorithm proposed in [14], dual-monotonicity plays a crucial role in various aspects of ExCover. Several well-known quality scores such as F-score, the Fisher score, information gain, Gini index, χ^2 and support difference are all dual-monotonic.

2.3 Branch-and-bound Pruning in Top-k Mining

Suppose that we are performing a branch-and-bound search for top-k patterns under a dual-monotonic quality score R_c. Also consider an anti-monotonic upper bound $\overline{R}_c(\boldsymbol{x})$ of $R_c(\boldsymbol{x})$ of a pattern $\boldsymbol{x}$. Then, if it is found that $\overline{R}_c(\boldsymbol{x}) < R_c(\boldsymbol{z})$, where $\boldsymbol{z}$ is the pattern with the k-th greatest score at the moment, we can safely prune the subtree rooted by $\boldsymbol{x}$ in the enumeration tree. This pruning exploits the anti-monotonicity of $\overline{R}_c$ w.r.t. pattern-inclusion, which guarantees $R_c(\boldsymbol{x}') \leq \overline{R}_c(\boldsymbol{x}') \leq \overline{R}_c(\boldsymbol{x}) < R_c(\boldsymbol{z})$ for any super-pattern $\boldsymbol{x}'$ of $\boldsymbol{x}$. $R_c(\boldsymbol{x})$ and $\overline{R}_c(\boldsymbol{x})$ are computed from $p(\boldsymbol{x} \mid c)$ and $p(\boldsymbol{x} \mid \neg c)$, which in turn are computed from the statistics stored in the (compressed) databases such as FP-trees.

The next question is how to obtain such an anti-monotonic upper bound. Since $R_c(\boldsymbol{x})$ is dual-monotonic, by definition $R_c(\boldsymbol{x})$ is monotonically increasing (resp. decreasing) w.r.t. $p(\boldsymbol{x} \mid c)$ (resp. $p(\boldsymbol{x} \mid \neg c)$), and both $p(\boldsymbol{x} \mid c)$ and $p(\boldsymbol{x} \mid \neg c)$ are anti-monotonic w.r.t. pattern-inclusion. Thus, the most optimistic scenario when extending $\boldsymbol{x}$ into $\boldsymbol{x}'$ is that $p(\boldsymbol{x}' \mid c)$ remains $p(\boldsymbol{x} \mid c)$ and $p(\boldsymbol{x}' \mid \neg c)$ turns to be zero. One general way for obtaining an upper bound $\overline{R}_c(\boldsymbol{x})$ is then to

[1] In the previous example with the target class $c = +$, a pattern $\boldsymbol{x} = \{\mathsf{D}\}$ is excluded, since $p(\boldsymbol{x} \mid c) = 3/5 = 0.6$ and $p(\boldsymbol{x} \mid \neg c) = 5/5 = 1$.

Item x	$p(x \mid +)$	$p(x \mid -)$	$\mathrm{F}_+(x)$
A	0.8	0.6	0.667
B	0.8	0.4	0.727
C	0.6	0.4	0.600
D	0.6	1.0	0.462
E	0.6	0.8	0.500

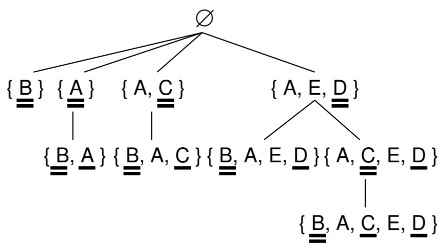

Fig. 1. F-scores of items (left) and the enumeration tree with SPC extension (right).

substitute $p(\boldsymbol{x} \mid \neg c) := 0$ into the definition of $R_c(\boldsymbol{x})$.[2] The upper bound $\overline{R}_c(\boldsymbol{x})$ where $p(\boldsymbol{x} \mid \neg c)$ is constant at zero is always anti-monotonic w.r.t. pattern-inclusion thanks to the dual-monotonicity of R_c. For example, the upper bound of F-score is obtained as $\overline{\mathrm{F}}_c(\boldsymbol{x}) = 2p(\boldsymbol{x} \mid c)/(1+p(\boldsymbol{x} \mid c))$. The operations described here are applicable to any dual-monotonic quality score, but we should remark that some quality score such as confidence $p(c \mid \boldsymbol{x})$, its upper bound obtained by the way above goes into infinity and pruning does not work in a practical sense.

2.4 The Closedness Constraint

As stated in the introduction, one popular technique for redundancy elimination is to use the closedness constraint. For that, we first introduce a closure operator Γ such that $\Gamma(\boldsymbol{x}, \mathcal{D}) = \bigcap_{t \in \mathcal{D}(\boldsymbol{x})} t$, where $\mathcal{D}$ is the transactions and $\boldsymbol{x}$ is some pattern. Here $\Gamma(\boldsymbol{x}, \mathcal{D})$ is called a *closure* of $\boldsymbol{x}$ w.r.t. $\mathcal{D}$. A closed pattern is then a pattern $\boldsymbol{x}$ such that $\boldsymbol{x} = \Gamma(\boldsymbol{x}, \mathcal{D})$. Each closed pattern $\boldsymbol{x}$ is the largest pattern in an equivalence class $[\boldsymbol{x}] = \{\boldsymbol{x}' \mid \mathcal{D}(\boldsymbol{x}) = \mathcal{D}(\boldsymbol{x}')\} = \{\boldsymbol{x}' \mid \boldsymbol{x} = \Gamma(\boldsymbol{x}', \mathcal{D})\}$ and seen as a representative of $[\boldsymbol{x}]$. Since the size of $[\boldsymbol{x}]$ can be exponential, focusing only on closed patterns often leads to a significant reduction of the search space.

Closed patterns are also beneficial from the viewpoint of quality, especially when the closure operator is applied only to the positive transactions $\mathcal{D}_c$. Such patterns are often said to be *closed on the positives*. Let c be a class of interest, $\mathcal{D}_c$ be positive transaction, and $\boldsymbol{x}$ be some pattern. Also let $\boldsymbol{x}^* = \Gamma_c(\boldsymbol{x})$, where $\Gamma_c(\boldsymbol{x})$ is an abbreviation of $\Gamma(\boldsymbol{x}, \mathcal{D}_c)$. We further note that $\mathcal{D}_c(\boldsymbol{x}^*) = \mathcal{D}_c(\boldsymbol{x})$ since $\boldsymbol{x}^*$ and $\boldsymbol{x}$ are in the same equivalence class $[\boldsymbol{x}]$, and $\mathcal{D}_{\neg c}(\boldsymbol{x}^*) \subseteq \mathcal{D}_{\neg c}(\boldsymbol{x})$ since $\boldsymbol{x}^*$ is the largest pattern in $[\boldsymbol{x}]$. Then, under a dual-monotonic quality score R_c, we have $R_c(\boldsymbol{x}^*) \geq R_c(\boldsymbol{x})$ since $p(\boldsymbol{x}^* \mid c) = p(\boldsymbol{x} \mid c)$ and $p(\boldsymbol{x}^* \mid \neg c) \leq p(\boldsymbol{x} \mid \neg c)$ [8, 14, 22]. This means that the quality of a patterns closed on the positives is no lower than the qualities of the patterns in the same equivalence class.

To enumerate patterns closed on the positives without duplicate visits to a pattern, we perform a variant of prefix-preserving closure (PPC) extension used in LCM [24], called *suffix-preserving closure* (SPC) *extension* [14]. Here we will explain SPC extension by illustration. First, consider again the dataset in Table 1 (1), where we have a total order $\mathsf{B} \prec \mathsf{A} \prec \mathsf{C} \prec \mathsf{E} \prec \mathsf{D}$ over items as the

[2] Equivalent substitutions are also possible: $p(c \mid \boldsymbol{x}) := 1$, $p(\neg \boldsymbol{x} \mid \neg c) := 1$, and so on.

descending order of F-score in Fig. 1 (left). Here the quality $R_c(x)$ of an item x is defined as $R_c(\{x\})$. Then, Fig. 1 (right) is the enumeration tree obtained by exhaustive applications of SPC extension. In enumeration, we introduce a new pattern by adding a *core item* to a pattern already visited. In Fig. 1 (right), the core items are underlined, and among them, the core items lastly added are doubly underlined. Other items, called *accompanying items* here, are those taken along into a pattern by the closure operation. In SPC extension, a pattern having *no* accompanying item which is a successor w.r.t. $\prec$ of the core item lastly added are considered to preserve the suffix of the original pattern. Then, a pattern *not* preserving the suffix is immediately pruned. At each branch in the enumeration tree, the core items to be added are chosen from unadded predecessors of the core item lastly added, in the ascending order w.r.t. $\prec$.

For example, given an empty pattern $\emptyset$, we apply an SPC extension by item D to the positive transactions in Table 1 (1) and obtain $\Gamma_c(\{\mathsf{D}\} \cup \emptyset) = \{\mathsf{A}, \mathsf{E}, \mathsf{D}\}$. In this case, D is the last core item in $\{\mathsf{A}, \mathsf{E}, \mathsf{D}\}$, while A and E are taken along into the pattern by the closure operation. For a new pattern $\{\mathsf{A}, \mathsf{E}, \mathsf{D}\}$, we further add B and C which have not been added yet, following the ascending order w.r.t. $\prec$. One may see that adding E into an empty pattern also yields the same pattern $\Gamma_c(\{\mathsf{E}\} \cup \emptyset) = \{\mathsf{A}, \mathsf{E}, \mathsf{D}\}$, where an accompanying item D is a successor of the core item E lastly added. The pattern $\{\mathsf{A}, \mathsf{E}, \mathsf{D}\}$ obtained in this way is not suffix-preserving and so is immediately pruned. We finally obtain the enumeration tree in Fig. 1 (right) which has nine non-root nodes which correspond to the patterns closed on the positives shown in Table 1 (2c).

In practice, the choice of the total order $\prec$ is important. Suppose we have $x \prec x'$ iff $R_c(x) \geq R_c(x')$ like the example above. Then, in a depth-first search, we visit patterns including high quality items earlier, and hence there would be more chances for pruning described in Sections 2.3 and 2.5.

2.5 The Productivity Constraint

We explain another constraint called productivity, whose original version is defined with confidence [2]. This constraint has also been used in associative classification [17] and explanatory analysis of Bayesian networks [28]. Productivity is defined as follows:

Definition 2. *Let c be a class of interest. Then, for a pair of patterns $\boldsymbol{x}$ and $\boldsymbol{x}'$, $\boldsymbol{x}$ is* weaker than *$\boldsymbol{x}'$ iff $\boldsymbol{x} \supset \boldsymbol{x}'$ and $R_c(\boldsymbol{x}) \leq R_c(\boldsymbol{x}')$. A pattern $\boldsymbol{x}$ is* productive *iff $\boldsymbol{x}$ is not weaker than any sub-pattern of $\boldsymbol{x}$.* □

It is mentioned in [14] that, under a dual-monotonic quality score, the productivity constraint above is tighter than the irrelevancy constraint [8, 10] among the patterns closed on the positives. Also, it is desirable to test easily the productivity constraint. In a depth-first search, the following property is useful [14]:

Proposition 1. *When a pattern $\boldsymbol{x}$ is visited in a depth-first search with SPC extension, all of $\boldsymbol{x}$'s sub-patterns have already been visited.* □

Algorithm 1 SeqCover

```
1: $L :=$ an empty set
2: while $\mathcal{D}_c \neq \emptyset$ do
3:     Induce the best rule $\boldsymbol{x} \Rightarrow c$ from $\mathcal{D}_c$ and $\mathcal{D}_{\neg c}$
4:     $L := L \cup \{\boldsymbol{x} \Rightarrow c\}$
5:     Remove all positive examples covered by $\boldsymbol{x}$ from $\mathcal{D}_c$
6: end while
7: Output the rules in $L$
```

In Fig. 1 (right), $\{\mathsf{B},\mathsf{A},\mathsf{E},\mathsf{D}\}$ is visited after all its sub-patterns $\{\mathsf{B}\}$, $\{\mathsf{A}\}$, $\{\mathsf{B},\mathsf{A}\}$ and $\{\mathsf{A},\mathsf{E},\mathsf{D}\}$ being visited in a depth-first search with SPC extension. So, when visiting $\{\mathsf{B},\mathsf{A},\mathsf{E},\mathsf{D}\}$, we can easily compare the quality of $\{\mathsf{B},\mathsf{A},\mathsf{E},\mathsf{D}\}$ with the qualities of $\{\mathsf{B}\}$, $\{\mathsf{A}\}$, $\{\mathsf{B},\mathsf{A}\}$ and $\{\mathsf{A},\mathsf{E},\mathsf{D}\}$ to test whether $\{\mathsf{B},\mathsf{A},\mathsf{E},\mathsf{D}\}$ is productive. This is not the case with PPC extension in the original LCM.

Moreover, we are able to conduct an aggressive pruning based on an extended notion of weakness which is defined as follows:

Definition 3. *Let c be a class of interest and $(\boldsymbol{x}, \boldsymbol{x}')$ be a pair of patterns. Then, $\boldsymbol{x}$ is prunably weaker than $\boldsymbol{x}'$ iff $\boldsymbol{x} \supset \boldsymbol{x}'$ and $\overline{R}_c(\boldsymbol{x}) \leq R_c(\boldsymbol{x}')$.* □

If a pattern $\boldsymbol{x}$ is prunably weaker than a pattern $\boldsymbol{x}'$ in the current top-k candidates, any super-pattern of $\boldsymbol{x}$ is also weaker than $\boldsymbol{x}'$, and thus we can safely prune the subtree rooted by $\boldsymbol{x}$. Prop. 1 is also useful for this pruning.

2.6 Sequential Covering

Sequential covering, also known as separate-and-conquer, is a traditional search strategy in rule learning [7, 26]. Also in the literature of discriminative pattern mining, several methods such as DDPMine [4] take this strategy. Algorithm 1 shows a typical workflow of sequential covering. Sequential covering takes as input the class c of interest and the dataset $\mathcal{D}_c \cup \mathcal{D}_{\neg c}$ and outputs a set of rules for class c. In sequential covering, we iteratively build a new rule (Line 3) and remove all positive examples covered by the new rule (Line 5). The iteration continues until there remain no positive examples to be covered (Line 2). DDPMine performs branch-and-bound search in building new rules.

The removal of positive examples surely reduces the overlap of coverage among generated rules, but there seem to be two problems. First, such a removal prevents us from *declarative* understanding of the generated rules. In other words, the meanings of the generated rules reflect a *procedural* behavior of Algorithm 1, which may not be clear to non-experts. For example, the statistics used in exploring the first new rule are different from those used in exploring the second new rule. Second, as Domingos [5] pointed out, available positive examples dwindle as the iteration continues, and therefore the rules generated at later iterations can be less accurate in a statistical sense.

Domingos proposed a greedy algorithm in which each rule is learned from the entire dataset, and referred to his own strategy by *conquering-without-separating* [5]. Later, Rijnbeek et al. [21] proposed an algorithm that directly finds a rule condition in disjunctive normal form (DNF), instead of finding conjunctive rule conditions one by one. A main drawback of this method is its computational cost, and therefore some extra control parameters limiting the size of the DNF condition are often required. ExCover, which will be explained next, also takes the conquering-without-separating approach in an exhaustive but light-weight manner without control parameters limiting the search space.

3 The Proposed Method

3.1 The Best-Covering Constraint

From now on, we explain the details of ExCover. As illustrated in Section 1, ExCover seeks for best-covering patterns that are closed on the positives. First, let us formally define the patterns output by ExCover:

Definition 4. *Let c be a class of interest and $t \in \mathcal{D}_c$ be a positive transaction. then, a pattern $\boldsymbol{x}$ is said to* cover t best *when $\boldsymbol{x}$ covers t (i.e. $\boldsymbol{x} \subseteq t$) and satisfies the following two conditions for any other pattern $\boldsymbol{x}'$ that also covers t:*

1. *$R_c(\boldsymbol{x}) \geq R_c(\boldsymbol{x}')$ if $\boldsymbol{x}'$ is not a subset of $\boldsymbol{x}$*
2. *$R_c(\boldsymbol{x}) > R_c(\boldsymbol{x}')$ if $\boldsymbol{x}'$ is a subset of $\boldsymbol{x}$*

A pattern $\boldsymbol{x}$ is said to be best-covering *if there is at least one positive transaction in $\mathcal{D}_c$ which is covered best by $\boldsymbol{x}$.* □

The definition of the "covers-best" relation above says that a pattern $\boldsymbol{x}$ is said to cover best a positive transaction t when $\boldsymbol{x}$ is of the highest quality among the patterns covering t. One technical point here is that the requirements on the quality score of $\boldsymbol{x}$ slightly differ depending on whether the pattern $\boldsymbol{x}'$ in comparison is a sub-pattern of $\boldsymbol{x}$ or not. Similarly to the "weaker-than" relation in Def. 2, no pattern covering t best is allowed to have the same quality as those of its sub-patterns. These slightly different requirements make it easy to prove a key property that the best-covering constraint is tighter than the productivity constraint (Def. 2):

Proposition 2. *A pattern $\boldsymbol{x}$ is productive if $\boldsymbol{x}$ is best-covering.* □

Proof. We prove this by contraposition. Suppose that $\boldsymbol{x}$ is not productive. That is, there exists a pattern $\boldsymbol{x}'$ such that $\boldsymbol{x}$ is weaker than $\boldsymbol{x}'$, i.e. $\boldsymbol{x}' \subset \boldsymbol{x}$ and $R_c(\boldsymbol{x}') \geq R_c(\boldsymbol{x})$. For this $\boldsymbol{x}'$, any transaction t covered by $\boldsymbol{x}$ is also covered by $\boldsymbol{x}'$ since $\boldsymbol{x}' \subset \boldsymbol{x} \subseteq t$. Then, for any positive transaction t covered by $\boldsymbol{x}$, the second condition of the "covers-best" relation is violated, and therefore $\boldsymbol{x}$ is not best-covering. □

Based on the definition of the best-covering constraint, we can immediately introduce the following pruning condition:

Proposition 3. *Let c be a class of interest and $\boldsymbol{x}$ be a pattern. Suppose that, for every positive transaction t covered by $\boldsymbol{x}$, there exists some other pattern $\boldsymbol{x}'$ covering t such that $\overline{R}_c(\boldsymbol{x}) < R_c(\boldsymbol{x}')$. Then, $\boldsymbol{x}$ and its super-patterns are not best-covering.* □

Algorithm 2 ExCover

```
1: $L :=$ an array indexed by $t \in \mathcal{D}_c$
2: Initialize $L[t]$ as an empty set for each $t \in \mathcal{D}_c$
3: $\boldsymbol{x} :=$ an empty pattern
4: $T :=$ an initial database constructed from $\mathcal{D}$
5: Call Grow$(\boldsymbol{x}, T)$
6: Output the patterns $\bigcup_{t \in \mathcal{D}_c} L[t]$
```

Proof. Let $\boldsymbol{u}$ be $\boldsymbol{x}$ or its super-pattern. Note that $R_c(\boldsymbol{u}) \leq \overline{R}_c(\boldsymbol{x})$ always holds and any transaction covered by $\boldsymbol{u}$ is also covered by $\boldsymbol{x}$. Then, for any positive transaction t covered by $\boldsymbol{u}$, $R_c(\boldsymbol{u}) \leq \overline{R}_c(\boldsymbol{x}) < R_c(\boldsymbol{x}')$ holds for $\boldsymbol{x}'$ in the proposition. From this fact, $\boldsymbol{u}$ cannot cover t best since either of the two conditions in Def. 4 is unsatisfied. □

In addition, the "covers-best" relation in Def. 4 is restated in a different form which reduces the computational cost required for set-inclusion check:

Proposition 4. *Let c be a class of interest and $t \in \mathcal{D}_c$ be a positive transaction. Then, a pattern $\boldsymbol{x}$ covers t best iff $\boldsymbol{x}$ covers t and satisfies one of the following conditions for any other pattern $\boldsymbol{x}'$ covering t:*

1. *$R_c(\boldsymbol{x}) > R_c(\boldsymbol{x}')$, and*
2. *$R_c(\boldsymbol{x}) = R_c(\boldsymbol{x}')$ and $\boldsymbol{x}$ is not a superset of $\boldsymbol{x}'$.* □

Proof. Let us introduce three Boolean variables A, B and C which respectively indicate "$\boldsymbol{x}'$ is a subset of $\boldsymbol{x}$," "$R_c(\boldsymbol{x}) > R_c(\boldsymbol{x}')$" and "$R_c(\boldsymbol{x}) = R_c(\boldsymbol{x}')$." The "covers-best" relation in Def. 4 is then written as $(\neg A \Rightarrow B \vee C) \wedge (A \Rightarrow B)$. This condition is simplified as $B \vee (C \wedge \neg A)$, which coincides with the proposition. □

3.2 Algorithm Description

Based on the notions and properties explained so far, ExCover efficiently performs a branch-and-bound search for best-covering patterns closed on the positives. An underlying strategy of ExCover is to conduct a top-1 mining (top-k mining where $k = 1$) concurrently for each positive transaction t. To be specific, we show the main routine of ExCover in Algorithm 2. Like sequential covering, ExCover takes as input the class c of interest and the dataset $\mathcal{D}_c \cup \mathcal{D}_{\neg c}$. We first introduce a global array variable L referring to the *candidate table* which stores candidate patterns (Lines 1–2). Formally, for each positive transaction t, $L[t]$ is a candidate set of t's best-covering patterns, i.e. those covering t and having the same highest quality score. Then, we call the Grow procedure with an empty pattern and an initial FP-tree-like database for finding best-covering patterns closed on the positives in a depth-first manner (Lines 3–5). After the call, ExCover outputs all patterns stored in L, removing duplicates (Line 6).[3]

[3] In other words, ExCover outputs all patterns having the same best score. We only exclude apparently redundant patterns to avoid the loss of crucial information.

Algorithm 3 GROW($\boldsymbol{x}, T$)

```
Require: $\boldsymbol{x}$: the current pattern, $T$: conditional database corresponding to $\boldsymbol{x}$
 1: $B := \{x \in \mathcal{X} \mid x \notin \boldsymbol{x}$ and $x$ is a predecessor of the core item lastly added into $\boldsymbol{x}\}$
 2: for each $x \in B$ enumerated in the ascending order w.r.t. $\prec$ do
 3:     $\boldsymbol{x}' := \{x\} \cup \boldsymbol{x}$ and compute $\overline{R}_c(\boldsymbol{x}')$ from $T$
 4:     if $\bigcup_{t \in \mathcal{D}_c(\boldsymbol{x}')} L[t] \neq \emptyset$ and $\overline{R}_c(\boldsymbol{x}') < \min_{t \in \mathcal{D}_c(\boldsymbol{x}'), \boldsymbol{z} \in L[t]} R_c(\boldsymbol{z})$ then continue
 5:     $\boldsymbol{x}^* := \Gamma_c(\boldsymbol{x}')$
 6:     if $\boldsymbol{x}^*$ does not preserve $\boldsymbol{x}$'s suffix then continue
 7:     Construct $T^*$ corresponding to $\boldsymbol{x}^*$ from $T$
 8:     Compute $R_c(\boldsymbol{x}^*)$ from $T^*$ together with $p(\boldsymbol{x}^* \mid c)$ and $p(\boldsymbol{x}^* \mid \neg c)$
 9:     Call ADD($\boldsymbol{x}^*$) if $p(\boldsymbol{x}^* \mid c) \geq p(\boldsymbol{x}^* \mid \neg c)$
10:     Call GROW($\boldsymbol{x}^*, T^*$)
11: end for
```

The GROW procedure, presented in Algorithm 3, recursively visits patterns in a depth-first order over the enumeration tree like the one illustrated in Fig. 1 (right). When visiting a pattern $\boldsymbol{x}$, as a part of SPC extension, we collect the core items to be added from unadded predecessors of the core item lastly added (Line 1) and pick them up one by one in the ascending order w.r.t. $\prec$ (Line 2). Then, for each collected item x, we first obtain a temporary pattern $\boldsymbol{x}'$ by adding x into the current pattern $\boldsymbol{x}$ and then compute its upper bound of quality from the conditional database corresponding to $\boldsymbol{x}$ (Line 3). In the pruning condition in Line 4, the former part $\bigcup_{t \in \mathcal{D}_c(\boldsymbol{x}')} L[t] \neq \emptyset$ checks whether there exists a candidate pattern stored in L covering a positive transaction covered by $\boldsymbol{x}'$.[4] If there does not exist, $\boldsymbol{x}'$ is free from being pruned regardless of its quality. The latter part, derived from Prop. 3, then checks the quality of the temporary pattern $\boldsymbol{x}'$. If the pruning condition is not satisfied, we generate a new pattern $\boldsymbol{x}^*$ closed on the positives from the temporary pattern (Line 5). We then we check the validity of $\boldsymbol{x}^*$ w.r.t. SPC extension (Line 6), and invalid ones are pruned immediately. For a valid pattern $\boldsymbol{x}^*$, we construct a new conditional database T^* (Line 7), evaluate the quality of $\boldsymbol{x}^*$ (Line 8), add $\boldsymbol{x}^*$ to the candidate table L (Line 9), and further visit $\boldsymbol{x}^*$ (Line 10). Since we only need the patterns characterizing class c, we exclude $\boldsymbol{x}^*$ such that $p(\boldsymbol{x}^* \mid c)$ is lower than $p(\boldsymbol{x}^* \mid \neg c)$, but in any case we continue to visit the super-patterns of $\boldsymbol{x}^*$.

The ADD procedure, presented in Algorithm 4, adds a pattern satisfying the best-covering constraint into the candidate table L. The procedure is derived directly from Prop. 4. Since $L[t]$ contains candidate patterns with the same highest quality score, to obtain the highest score for a positive transaction t, in Line 2, it is sufficient to pick up $\boldsymbol{z}$ arbitrarily from $L[t]$ as a representative.

Lastly, we add three remarks. First, the property of SPC extension on the visiting order in Prop. 1 effectively works in Line 4 of the GROW procedure. That is, since it is guaranteed that all sub-patterns of $\boldsymbol{x}'$ have already been visited, each $L[t]$ is not empty for $t \in \mathcal{D}_c(\boldsymbol{x}')$ at least when $\boldsymbol{x}'$ has two or more core items,

[4] Remind that $\mathcal{D}_c(\boldsymbol{x})$ denotes the set of positive transactions covered by a pattern $\boldsymbol{x}$.

Algorithm 4 Add($\boldsymbol{x}$)

```
Require: $\boldsymbol{x}$: a pattern to be added
1: for each $t \in \mathcal{D}_c(\boldsymbol{x})$ do
2:     Let $r$ be the quality $R_c(\boldsymbol{z})$ of an arbitrary pattern $\boldsymbol{z}$ in $L[t]$
3:     if $R_c(\boldsymbol{x}) > r$ then
4:         $L[t] := \{\boldsymbol{x}\}$
5:     else if $R_c(\boldsymbol{x}) = r$ and $\boldsymbol{x}$ is not a superset of any pattern $\boldsymbol{z}$ in $L[t]$ then
6:         $L[t] := L[t] \cup \{\boldsymbol{x}\}$
7:     end if
8: end for
```

and hence the latter part of the pruning condition is not skipped. Second, for reducing the computational effort, we delay the closure operation Γ_c, which is costly in general, until Line 5. Instead, we refer to $\overline{R}_c(\boldsymbol{x}')$ rather than $\overline{R}_c(\boldsymbol{x}^*)$ in the pruning condition in Line 4. This operation is justified by the nature of the patterns closed on the positives and the way of obtaining the upper bound $\overline{R}_c$, i.e. we have $\overline{R}_c(\Gamma_c(\boldsymbol{x})) = \overline{R}_c(\boldsymbol{x})$ for any pattern $\boldsymbol{x}$. The third remark is that, we frequently refer to $\mathcal{D}_c(\boldsymbol{x})$ for a given pattern $\boldsymbol{x}$. So for quick reference to $\mathcal{D}_c(\boldsymbol{x})$, a data structure for conditional databases in a vertical layout [29] is crucial from a practical point of view. In the implementation used in our experiments, we introduced a simple extension of FP-trees [12] in which each node contains a list of TIDs for each class in addition to the counts.

4 Experimental Results

We conducted two experiments. The first experiment aims at confirming whether we can obtain a more compact set of discriminative itemsets for the mushroom dataset, which is available from the UCI Machine Learning Repository (`http://archive.ics.uci.edu/ml/datasets/Mushroom`). The second one compares the search space using the datasets available from `http://dtai.cs.kuleuven.be/CP4IM/datasets/`.

In the first experiment, we transform the original dataset into a transaction dataset, treating each attribute-value pair as an item. We then obtained the patterns shown in Table 2 (above) by a previous method [14] for mining top-k productive patterns closed on the positives and the patterns shown in Table 2 (below) by ExCover. $k = 30$ was specified in the previous method but only eight patterns were generated. Among the patterns obtained by the previous method, the top-2 patterns cover 4,112 out of 4,208 positive transactions. Although the third-ranked pattern has high quality but the positive transactions covered by it are also covered by the top-2 patterns. The remaining 96 positive transactions, on the other hand, are covered the fifth-ranked pattern. From this result, we can say that the patterns like the third-ranked one are redundant and specifying $k < 5$ implies losing information from 96 positive transactions. In contrast, thanks to the best-covering constraint, redundant patterns including the third-ranked one in Table 2 (above) are automatically excluded from the output of ExCover.

Table 2. Discriminative patterns for the *edible* class in the mushroom dataset, obtained by a previous method [14] (above) and obtained by ExCover (below).

Rank	Pattern	F-score
1	{odor=n, veil-type=p}	0.881
2	{gill-size=b, stalk-surface-above-ring=s, veil-type=p}	0.866
3	{gill-size=b, stalk-surface-below-ring=s, veil-type=p}	0.837
4	{gill-size=b, veil-type=p}	0.798
5	{stalk-surface-above-ring=s, veil-type=p}	0.776
6	{ring-type=p, veil-type=p}	0.771
7	{stalk-surface-below-ring=s, veil-type=p}	0.744
8	{veil-type=p}	0.682

Rank	Pattern	F-score
1	{odor=n, veil-type=p}	0.881
2	{gill-size=b, stalk-surface-above-ring=s, veil-type=p}	0.866
3	{stalk-surface-above-ring=s, veil-type=p}	0.776

Table 3. Statistics on the datasets used in the second experiment.

Dataset	#Transactions	#Items	Dataset	#Transactions	#Items
anneal	812	93	lymph	148	68
audiology	216	148	mushroom	8,124	119
australian-credit	653	125	primary-tumor	336	31
german-credit	1,000	112	soybean	630	50
heart-cleveland	296	95	splice-1	3,190	287
hepatitis	137	68	tic-tac-toe	958	27
hypothyroid	3,247	88	vote	435	48
kr-vs-kp	3,196	73	zoo-1	101	36

The statistics on the datasets used in the second experiment is summarized in Table 3. With these datasets, we compare the search space, i.e. the number of visited patterns in the enumeration tree, between the previous method above and ExCover.[5] For the previous method, we made three runs varying $k = 10, 100$ and $1,000$. First, we show the number of output patterns in Fig. 2 (top). In the graphs, P+C(k=10) corresponds to the case with the previous method under $k = 10$ (P+C stands for productivity and closedness), and so on. The results show that ExCover outputs a more compact set of patterns, as Prop. 2 formally suggests. For most of the datasets, there are more than ten productive patterns, and there are fewer (sometimes by an order of magnitude) best-covering patterns. A more drastic difference was observed in the number of visited patterns shown in Fig. 2 (middle). The search space of the previous method varies with k and hence a suitable k is sometimes unclear in advance, whereas ExCover totally explores the search space of moderate size. Finally, Fig. 2 (bottom) shows the run time averaged over 30 runs. Our implementation is written in Java and we used Intel Core i7 3.6GHz. The result clearly shows that ExCover finishes in practical time, i.e. within one second in most datasets.

[5] More formally, a visited pattern is a pattern closed on the positives produced at Line 5 in the GROW procedure.

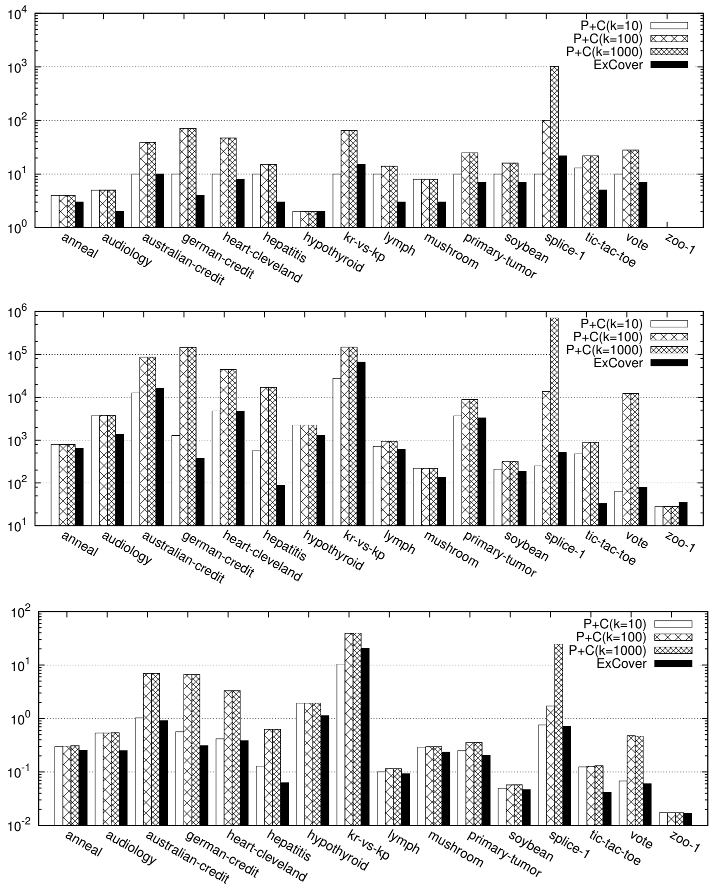

Fig. 2. The number of output patterns (top) and visited patterns (middle), and the average run time (bottom, in seconds). The y-axis is logarithmic and only one pattern were found in zoo-1. The error bars for average run time are narrow and omitted.

5 Related Work

Associative classification [23] is a task for building classifiers from class association rules (CARs) and is closely related to discriminative pattern mining. CBA [18] is the first associative classifier, in which we first extract CARs using two user-specified control parameters: minimum support and minimum confidence. To filter out redundant CARs, CMAR [17] further introduces the productivity constraint (based on confidence) and χ^2 testing. The closest method to ExCover should be HARMONY [25] since it seeks for CARs which have the highest confidence for at least one positive transaction. However, as said earlier, its original paper does not mention redundancy among CARs. In addition, the pruning mechanism in HARMONY is fairly complicated and heavily relies on the minimum support threshold given by the user. One possible reason for this is that a finite upper bound of confidence is not easy to obtain. On the other hand, using a dual-monotonic quality score whose upper bound is finite, ExCover performs simple branch-and-bound pruning with no user-specified control parameters. To achieve parameter-freeness, as done by the fitcare algorithm [3], automatic tuning of control parameters looks promising, but it may weaken the explainability unless the objective function for tuning is clear to the user.

In addition, a generic problem-solving framework, which includes pattern set mining, based on constraint programming was recently developed [11]. We are rather focusing on exploring a desirable combination of comprehensible constraints, and developing a specific algorithm that works with less resources. For example, ExCover chooses a depth-first search over a suffix enumeration tree that brings less memory consumption and more chances for pruning. Lastly, Xin et al. proposed a method for mining non-redundant frequent patterns by minimizing the overlaps among the patterns' coverage over transactions [27]. The best-covering constraint balances well the relevance to the class of interest with the degrees of overlaps.

6 Concluding Remarks

This paper proposed an algorithm called ExCover that finds best-covering patterns closed on the positives in an exhaustive manner. Formal discussions and experimental results exhibit that ExCover efficiently finds a more compact set of patterns in comparison with previous methods. Although ExCover was motivated by knowledge discovery situations, it is interesting to build a practical classifier using best-covering patterns. For that, an extension for handling continuous attributes seems indispensable. In addition, the idea behind ExCover seems not limited to closed itemsets. For example, it would be useful to extend ExCover for discriminative sequential pattern mining [9].

References

1. Aggarwal, C.C.: Data Mining: The Textbook. Springer (2015)
2. Bayardo, R., Agrawal, R., Gunopulos, D.: Constraint-based rule mining in large, dense databases. Data Mining and Knowledge Discovery 4, 217–240 (2000)
3. Cerf, L., Gay, D., Selmaoui, N., Boulicaut, J.F.: A parameter-free associative classification method. In: Proc. of DaWaK-08. pp. 293–304 (2008)
4. Cheng, H., Yan, X., Han, J., Yu, P.S.: Direct discriminative pattern mining for effective classification. In: Proc. of ICDE-08. pp. 169–178 (2008)
5. Domingos, P.: The RISE system: conquering without separating. In: Proc. of ICTAI-94. pp. 704–707 (1994)
6. Dong, G., Bailey, J. (eds.): Contrast Data Mining: Concepts, Algorithms, and Applications. CRC Press (2012)
7. Fürnkranz, J., Gamberger, D., Lavrač, N.: Foundations of Rule Learning. Springer (2012)

8. Garriga, G.C., Kralj, P., Lavrač, N.: Closed sets for labeled data. J. of Machine Learning Research 9, 559–580 (2008)
9. Grosskreutz, H., Lang, B., Trabold, D.: A relevance criterion for sequential patterns. In: Proc. of ECML/PKDD-13. pp. 369–384 (2013)
10. Grosskreutz, H., Paurat, D.: Fast and memory-efficient discovery of the top-k relevant subgroups in a reduced candidate space. In: Proc. of ECML/PKDD-11. pp. 533–548 (2011)
11. Guns, T., Nijssen, S., De Raedt, L.: k-Pattern set mining under constraints. IEEE Trans. on Knowledge and Data Engineering 25(2), 402–418 (2013)
12. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proc. of SIGMOD-00. pp. 1–12 (2000)
13. Han, J., Wang, J., Lu, Y., Tzvetkov, P.: Mining top-k frequent closed patterns without minimum support. In: Proc. of ICDM-02. pp. 211–218 (2002)
14. Kameya, Y., Asaoka, H.: Depth-first traversal over a mirrored space for non-redundant discriminative itemsets. In: Proc. of DaWaK-13. pp. 196–208 (2013)
15. Kralj Novak, P., Lavrač, N., Webb, G.I.: Supervised descriptive rule discovery: a unifying survey of contrast set, emerging pattern and subgroup mining. J. of Machine Learning Research 10, 377–403 (2009)
16. Li, J., Li, H., Wong, L., Pei, J., Dong, G.: Minimum description length principle: generators are preferable to closed patterns. In: Proc. of AAAI-06. pp. 409–414 (2006)
17. Li, W., Han, J., Pei, J.: CMAR: accurate and efficient classification based on multiple class-association rules. In: Proc. of ICDM-01. pp. 369–376 (2001)
18. Liu, B., Hsu, W., Ma, Y.: Integrating classification and association rule mining. In: Proc. of KDD-98. pp. 80–86 (1998)
19. Morishita, S., Sese, J.: Traversing itemset lattices with statistical metric pruning. In: Proc. of PODS-00. pp. 226–236 (2000)
20. Pasquier, N., Bastide, Y., Taouli, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Proc. of ICDT-99. pp. 398–416 (1999)
21. Rijnbeek, P.R., Kors, J.A.: Finding a short and accurate decision rule in disjunctive normal form by exaustive search. Machine Learning 80, 33–62 (2010)
22. Soulet, A., Crémilleux, B., Rioult, F.: Condensed representation of emerging patterns. In: Proc. of PAKDD-04. pp. 127–132 (2004)
23. Thabtah, F.: A review of associative classification mining. Knowledge Engineering Review 22(1), 37–65 (2007)
24. Uno, T., Asai, T., Uchida, Y., Arimura, H.: An efficient algorithm for enumerating closed patterns in transaction databases. In: Proc. of DS-04. pp. 16–31 (2004)
25. Wang, J., Karypis, G.: HARMONY: efficiently mining the best rules for classification. In: Proc. of SDM-05. pp. 205–216 (2005)
26. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, 2nd edn. (2005)
27. Xin, D., Han, J., Yan, X., Cheng, H.: On compressing frequent patterns. Data & Knowledge Engineering 60(1), 5–29 (2007)
28. Yuan, C., Lim, H., Lu, T.C.: Most relevant explanation in Bayesian networks. J. of Artificial Intelligence Research 42, 309–352 (2011)
29. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. on Knowledge and Data Engineering 12(3), 372–390 (2000)