NDPMine: Efficiently Mining Discriminative Numerical Features for Pattern-Based Classification

Hyungsul Kim, Sangkyum Kim, Tim Weninger, Jiawei Han, and Tarek Abdelzaher

Department of Computer Science,
University of Illinois at Urbana-Champaign,
Urbana IL 61801, USA
{hkim21,kim71,weninge1,hanj,zaher}@illinois.edu

Abstract. Pattern-based classification has demonstrated its power in recent studies, but because the cost of mining discriminative patterns as features in classification is very expensive, several efficient algorithms have been proposed to rectify this problem, such as *DDPMine* [3], *corrmine* [14], and *gPLS* [17]. *DDPMine* obtains its efficiency by iteratively pruning the input dataset thereby potentially losing important information. Other methods, such as *gBoost* [18], are either memory efficient or computationally efficient, but not both. These algorithms also assume that feature values of the mined patterns are binary, i.e., the pattern either exists or not. In some problems, such as software behavior analysis, the number of times a pattern appears is more informative than whether the pattern appears or not. To resolve these deficiencies, we propose a mathematical programming method that directly mines discriminative patterns as numerical features for classification. We also propose a novel search space shrinking technique which addresses the inefficiencies in iterative pattern mining algorithms. Finally, we show that our method is an order of magnitude faster, significantly more memory efficient and more accurate than current approaches.

Keywords: Pattern-Based Classification, Discriminative Pattern Mining

1 Introduction

Pattern-based classification is a process of learning a classification model where patterns are used as features. Recent studies show that classification models which make use of pattern-features can be more accurate and more understandable than the original feature set [2, 3]. Pattern-based classification has been adapted to work on data with complex structures such as sequences [12, 9, 15, 6, 20], and graphs [17, 18, 16], where discriminative frequent patterns are taken as features to build high quality classifiers.

These approaches can be grouped into two settings: binary or numerical. Binary pattern-based classification is the well-known problem setting in which the feature space is $\{0, 1\}^d$, where d is the number of features. This means that a classification model only uses information about whether an interesting pattern exists or not. On the other hand, a numerical pattern-based classification model's feature space is $\mathbb{N}^d$, which means that the classification model uses information about how many times an interesting pattern appears. For instance, in the analysis of software traces loops and other

repetitive behaviors may be responsible for failures. Therefore, it is necessary to determine the number of times a pattern occurs in a sequence.

Pattern-based classification techniques are prone to a major efficiency problem to the exponential number of possible patterns. Several studies have identified this issue and offered solutions [3, 6]. However, to our knowledge there has not been any work addressing this issue in the case of numerical features. Furthermore, these solutions achieve their efficiency by removing patterns that could have been features.

Recently a boosting approach was proposed by Saigo *et al.* called *gBoost* [18]. Their algorithm employs a linear programming approach to boosting as a base algorithm combined with a pattern mining algorithm. The linear programming approach to boosting algorithm (LPBoost) [5] is shown to converge faster than ADABoost [7] and is proven to converge to a global solution. *gBoost* works by iteratively growing and pruning a search space of patterns via branch and bound search. In work prior to *gBoost* [16] by the same authors, the search space is erased and rebuilt during each iteration. However, in their most recent work, the constructed search space is reused in each iteration to minimize computation time; the authors admit that this approach would not scale but were able to complete their case study with 8GB main memory.

The high cost of the finding numerical features along with the accuracy issues of binary-only features motivates us to investigate an alternative approach. What we wish to develop is a method which is both efficient and able to mine numerical features for classification. This leads to our proposal of a *numerical direct pattern mining* approach, **NDPMine**. Our approach employs a mathematical programming method that directly mines discriminative patterns as numerical features. We also address the fundamental problem of iterative pattern mining algorithms, and propose a novel search space shrinking technique to prune memory space without removing potential features. We show that our method is an order of magnitude faster, significantly more memory efficient and more accurate than current approaches.

The structure of this paper is as follows. In Section 2 we provide a brief background survey and discuss in further detail the problems that *NDPMine* claims to remedy. In Section 3 we introduce the problem setting. Section 4 describes our discriminative pattern mining approach, pattern search strategy and search space shrinking technique. The experiments in Section 5 compare our algorithm with current methods in terms of efficiency and accuracy. Finally, Section 6 contains our conclusions.

2 Background and Related Work

The first pattern mining algorithms originated from the domain of association rule mining in which CBA [11] and CMAR [10] used the **two-step** pattern mining process to generate a feature set for classification. Cheng *et al.* [2] showed that, within a large set of frequent patterns, those patterns which have higher *discriminative* power, *i.e.* had higher information gain and/or Fisher score, are useful in classification. With this intuition, their algorithm (*MMRFS*) selects patterns for inclusion in the feature set based on the information gain or Fisher score of each pattern. The following year, Cheng *et al.* [3] showed that they could be more efficient if they performed pattern-based classification by a **direct process** which directly mines discrim-inative patterns (*DDPMine*). A

separate algorithm by Fan *et al.* (called M^bT) [6], developed at the same time as *DDPMine*, uses a decision tree-like approach, which recursively splits the training instances by picking the most discriminative patterns.

As alluded to earlier, an important problem with the many approaches is that the feature set used to build the classification model is entirely binary. This is a significant drawback because many datasets rely on the number of occurrences of a pattern in order to train an effective classifier. One such dataset comes from the realm of software behavior analysis in which patterns of events in software traces are available for analysis. Loops and other repetitive behaviors observed in program traces may be responsible for failures. Therefore, it is necessary to mine not only the execution patterns, but also the number of occurrences of the patterns. Lo *et al.* [12] proposed a solution to this problem (hereafter called SoftMine) which mines *closed unique iterative patterns* from normal and failing program traces in order to identify software anomalies. Unfortunately, this approach employs the less efficient two-step process which exhaustively enumerates a huge number of frequent patterns before finding the most discriminative patterns.

Other approaches have been developed to address specific datasets. For time series classification, Ye and Keogh [20] used patterns called shapelets to classify time-series data. Other algorithms include DPrefixSpan [15] which classifies action sequences, XRules [21] which classifies trees, and gPLS [17] which classifies graph structures.

Table 1. Comparison of related work

	Binary	**Numerical**
Two-step	MMRFS	SoftMine, Shapelet
Direct	DDPMine, M^bT, gPLS DPrefixSpan, gBoost	**NDPMine**

Table 1 compares the aforementioned algorithms in terms of the pattern's feature value (binary or numerical) and feature selection process (two-step or direct). To the best of our knowledge there do not exist any algorithms which mine patterns as numerical features in a direct manner.

3 Problem Formulation

Our framework is a general framework for numerical pattern-based classification. We, however, confine our algorithm for structural data classification such as sequences, trees, and/or graphs in order to present our framework more clearly. There are several pattern definitions for each structural data. For example, for sequence datasets, there are sequential patterns, episode patterns, iterative patterns, and unique iterative patterns [12]. The pattern definition which is better for classification depends on each dataset, and thus, we assume that the definition of a pattern is given as an input. Let $D = \{x_i, y_i\}_{i=1}^n$ be a dataset, containing structural data, where x_i is an object and y_i is its label. Let P be the set of all possible patterns in the dataset.

We will introduce several definitions, many of which are frequently used in pattern mining papers.

A pattern p in P is a sub-pattern of q if q contains p. If p is a sub-pattern of q, we say q is a super-pattern of p. For example, in a sequence, a sequential pattern $\langle A, B \rangle$

is a *sub-pattern* of a sequential pattern $\langle A, B, C\rangle$ because we can find $\langle A, B\rangle$ within $\langle A, B, C\rangle$.

The number of occurrences of a given pattern p in a data x is denoted by $occ(p, x)$. For example, if we count the number of non-overlapped occurrences of a pattern, the number of occurrences of a pattern $p = \langle A, B\rangle$ in a data $x = \langle A, B, C, D, A, B\rangle$ is 2, and $occ(p, x) = 2$. Since the number of occurrences of a pattern in a data depends on a user's definition, we assume that the function occ is given as an input.

The *support* of a pattern p in D is denoted by $sup(p, D)$, where $sup(p, D) = \sum_{x_i \in D} occ(p, x_i)$. A pattern p is *frequent* if $sup(p, D) \geq \theta$, where θ is a minimum support threshold.

A function f on P is said to posses the *apriori property* if $f(p) \leq f(q)$ for any pattern p and all its sub-patterns q.

With these definitions, the problem we present in this paper is as follows: Given a dataset $D = \{x_i, y_i\}_{i=1}^n$, and an occurrence function occ with the *apriori property*, we want to find a good feature set of a small number of discriminative patterns $F = \{p_1, p_2, \ldots, p_m\} \subseteq P$ so that we map D into $\mathbb{N}^m$ space to build a classification model. The training dataset in $\mathbb{N}^m$ space for building a classification model is denoted by $D' = \{\boldsymbol{x}'_i, y_i\}_{i=1}^n$, where $x'_{ij} = occ(p_j, x_i)$.

4 NDPMine

From the discussion in Section 1, we see the need for a method which efficiently mines discriminative numerical features for pattern-based classification. This section describes such a method called NDPMine (**N**umerical **D**iscriminative **P**attern Mining).

4.1 Discriminative Pattern Mining with LP

For direct mining of discriminative patterns two properties are required: (1) a measure for discriminative power of patterns (2) a theoretical bound of the measure for pruning search space. Using information gain and Fisher score, *DDPMine* successfully showed the theoretical bound when feature values of patterns are binary. However, there are no theoretical bounds for information gain and Fisher score when feature values of patterns are numerical. Since standard statistical measures for discriminative power are not suitable in our problem, we take different approach: model-based feature set mining. Model-based feature set mining find a set of patterns as a feature set while building a classifier. In this section, we will show that *NDPMine* has the two properties required for direct mining of discriminative patterns by formulating and solving an optimization problem of building a classifier.

To do that, we first convert a given dataset into a high-dimensional dataset, and learn a hyperplane as a classification boundary.

Definition 1. *A pattern and class label pair (p, c) is called class-dependent pattern, where $p \in P$ and $c \in C = \{-1, 1\}$. Then, the value of a class-dependent pattern (p, c) for data x is denoted by $s_c(p, x)$, where $s_c(p, x) = c \cdot occ(p, x)$.*

Since there are $2|P|$ class-dependent patterns, we have $2|P|$ values for an object x in D. Therefore, by using all class-dependent patterns, we can map x_i in D into $\boldsymbol{x}'_i$ in $\mathbb{N}^{2|P|}$ space, where $x'_{ij} = s_{c_j}(p_j, x'_i)$. One way to train a classifier in high dimensional space is to learn a classification hyperplane (*i.e.,* a bound with maximum margin) by formulating and solving an optimization problem. Given the training data $D' = \{x'_i, y_i\}_{i=1}^n$, the optimization problem is formulated as follows:

$$\begin{aligned} &\max_{\boldsymbol{\alpha},\rho} \rho \\ &\text{s.t.} \sum_{(p,c)\in P\times C} y_i \alpha_{p,c} s_c(p, x'_i) \geq \rho, \quad \forall i \\ &\sum_{(p,c)\in P\times C} \alpha_{p,c} = 1, \quad \alpha_{p,c} \geq 0, \end{aligned} \tag{1}$$

where $\boldsymbol{\alpha}$ represents the classification boundary, and ρ is the margin between two classes and the boundary.

Let $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\rho}$ be the optimal solution for (1). Then, the prediction rule learned from (1) is $f(\boldsymbol{x}') = sign(\boldsymbol{x}' \cdot \tilde{\boldsymbol{\alpha}})$, where $sign(v) = 1$ if $v \geq 0$ and -1, otherwise. If $\exists (p, c) \in P \times C : \alpha_{p,c} = 0$, $f(\boldsymbol{x}')$ is not affected by the dimension of the class-dependent pattern (p, c). Let $F = \{p | \exists c \in C, \tilde{\alpha}_{p,c} > 0\}$. If we use F instead of P in (1), we will have the same prediction rule. In other words, only the small number of patterns in F, we can learn the same classification model as the one learned by P. With this observation, we want to mine such a pattern set (equivalently: a feature set) F to build a classification model.

In addition, we want F to be as small as possible. In order to obtain a relatively small feature set, we need to obtain a very sparse vector $\boldsymbol{\alpha}$, where only few dimensions are non-zero values. To obtain a sparse weight vector $\boldsymbol{\alpha}$, we adopt the formulation from LPBoost [5].

$$\begin{aligned} &\max_{\boldsymbol{\alpha},\boldsymbol{\xi},\rho} \rho - \omega \textstyle\sum_{i=1}^n \xi_i \\ &\text{s.t.} \sum_{(p,c)\in P\times C} y_i \alpha_{p,c} s(\boldsymbol{x}'_i; p, c) + \xi_i \geq \rho, \quad \forall i \\ &\sum_{(p,c)\in P\times C} \alpha_{p,c} = 1, \quad \alpha_{p,c} \geq 0 \\ &\xi_i \geq 0, \quad i = 1, \ldots, n \end{aligned} \tag{2}$$

,where ρ is a soft-margin, $\omega = \frac{1}{\nu \cdot n}$, and ν is a parameter for misclassification cost. The difference between here is that (2) allows mis-classifications of the training instances to cost ω, where (1) does not. To allow mis-classifications, (2) introduces slack variables $\boldsymbol{\xi}$, and makes $\boldsymbol{\alpha}$ sparse in its optimal solution [5].

Next, we do not know all patterns in P unless we mine all of them, and mining all patterns in P is intractable. Therefore, we cannot solve (2) directly. Fortunately, such a linear optimization problem can be solved by *column generation* – a classic optimization technique [13]. The column generation technique, also called the cutting-plane algorithm, starts with an empty set of constraints in the dual problem and iteratively adds the most violated constraints. When there are no more violated constraints, the optimal solution under the set of selected constraints is equal to the optimal solution under all

constraints. To use the column generation technique in our problem, we give the dual problem of (2) as shown in [5].

$$\begin{aligned} \min_{\boldsymbol{\mu},\gamma}\ & \gamma \\ \text{s.t.}\ & \sum_{i=1}^{n} \mu_i y_i s_c(p, \boldsymbol{x}'_i) \le \gamma, \quad \forall (p,c) \in P \times C \\ & \sum_{i=1}^{n} \mu_i = 1, \quad 0 \le \mu_i \le \omega, \quad i = 1, \ldots, n. \end{aligned} \tag{3}$$

,where $\boldsymbol{\mu}$ can be interpreted as a weight vector for the training instances.

Each constraint $\sum_{i=1}^{n} \mu_i y_i s_c(p, \boldsymbol{x}'_i) \le \gamma$ in the dual 3 corresponds to a class-dependent pattern (p, c). Thus, the column generation finds a class-dependent pattern at each iteration whose corresponding constraint is violated the most. Let $H^{(k)}$ be the set of class-dependent patterns found so far at the k^{th} iteration. Let $\boldsymbol{\mu}^{(k)}$ and $\gamma^{(k)}$ be the optimal solution for k^{th} restricted problem:

$$\begin{aligned} \min_{\boldsymbol{\mu^{(k)}},\gamma^{(k)}}\ & \gamma^{(k)} \\ \text{s.t.}\ & \sum_{i=1}^{n} \mu_i^{(k)} y_i s_c(p, x'_i) \le \gamma^{(k)}, \quad \forall (p,c) \in H^{(k)} \\ & \sum_{i=1}^{n} \mu_i^{(k)} = 1, \quad 0 \le \mu_i^{(k)} \le \omega, \quad i = 1, \ldots, n \end{aligned} \tag{4}$$

After solving the k^{th} restricted problem, we search a class-dependent pattern (p^*, c^*) whose corresponding constraint is violated the most by the optimal solution $\gamma^{(k)}$ and $\boldsymbol{\mu}^{(k)}$, and add (p^*, c^*) to $H^{(k)}$.

Definition 2. *For a given (p, c), let $v = \sum_{i=1}^{n} \mu_i^{(k)} y_i s_c(p, x'_i)$. If $v \le \gamma^{(k)}$, the corresponding constraint of (p, c) is not violated by $\gamma^{(k)}$ and $\boldsymbol{\mu}^k$ because $\sum_{i=1}^{n} \mu_i^{(k)} y_i s_c(p, x'_i) = v \le \gamma^{(k)}$. If $v > \gamma^{(k)}$, then we say the corresponding constraint of (p, c) is violated by $\gamma^{(k)}$ and $\boldsymbol{\mu}^{(k)}$, and the margin of the constraint is defined as $v - \gamma^{(k)}$.*

In this view, (p^*, c^*) is the class-dependent pattern with the maximum margin. Now, we define our measure for discriminative power of class-dependent patterns.

Definition 3. *We define* gain function *for a given weight $\boldsymbol{\mu}$ as follows:*

$$gain(p, c; \boldsymbol{\mu}) = \sum_{i=1}^{n} \mu_i y_i s_c(p, x'_i).$$

For given $\gamma^{(k)}$ and $\boldsymbol{\mu}^{(k)}$, choosing the constraint with maximum margin is the same as choosing the constraint with maximum gain. Thus, we search for a class-dependent pattern with maximum gain in each iteration until there are no more violated constraints. Let k^* be the last iteration. Then, we can get the optimal solution $\tilde{\rho}$ and $\tilde{\boldsymbol{\alpha}}$ for (2) by

Algorithm 1 Discriminative Pattern Mining

$H^{(0)} \leftarrow \emptyset$
$\gamma^{(0)} \leftarrow 0$
$\mu_i^{(0)} = 1/n \quad \forall i = 1, \ldots, n$
for $k = 1, \ldots$ **do**
  $(p^*, c^*) = \mathrm{argmax}_{(p,c) \in P \times C}\, gain(p, c; \boldsymbol{\mu}^{(k-1)})$
  if $gain(p^*, c^*; \boldsymbol{\mu}^{(k-1)}) - \gamma^{(k-1)} < \epsilon$ **then**
    break
  end if
  $H^{(k)} \leftarrow H^{(k-1)} \cup \{(p^*, c^*)\}$
  Solve the k^{th} restricted problem (4) to get $\gamma^{(k)}$ and $\boldsymbol{\mu}^{(k)}$
end for
Solve (5) to get $\tilde{\boldsymbol{\alpha}}$
$F \leftarrow \{p | \exists c \in C, \tilde{\alpha}_{p,c} > 0\}$

solving the following optimization problem and setting $\tilde{\alpha}_{(p,c)} = 0$ for all $(p, c) \notin H^{(k^*)}$.

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\xi}, \rho} \; & -\rho + \omega \textstyle\sum_{i=1}^{n} \xi_i \\ \text{s.t.} \; & \sum_{(p,c) \in H^{(k^*)}} y_i \alpha_{p,c} s(\boldsymbol{x}_i'; p, c) + \xi_i \geq \rho, \quad \forall i \\ & \sum_{(p,c) \in H^{(k^*)}} \alpha_{p,c} = 1, \quad \alpha_{p,c} \geq 0 \\ & \xi_i \geq 0, \quad i = 1, \ldots, n \end{aligned} \tag{5}$$

The difference is that now we have the training instances in $|H^{(k^*)}|$ dimensions, not in $2|P|$ dimensions. Once we have $\tilde{\boldsymbol{\alpha}}$, as explained before, we can make a feature set F such that $F = \{p | \exists c \in C, \tilde{\alpha}_{p,c} > 0\}$. As a summary, the main algorithm of *NDPMine* is presented in Algorithm 1.

4.2 Optimal Pattern Search

As in *DDPMine* and other direct mining algorithms, our search strategy is a branch-and-bound approach. We assume that there is a canonical search order for P such that all patterns in P are enumerated without duplication. Many studies have been done for canonical search orders for most of structural data such as sequence, tree, and graph. Most of the pattern enumeration methods in these canonical search orders create the next pattern by extending the current pattern. Our aim is to find a pattern with maximum gain. Thus, for efficient search, it is important to prune the unnecessary or unpromising search space. Let p be the current pattern. Then, we compute the maximum gain bound for all super-patterns of p and decide whether we can prune the branch or not based on the following theorem.

Algorithm 2 Branch-and-bound Pattern Search

Global variables: $maxGain$, $maxPat$
procedure $search\_optimal\_pattern(\boldsymbol{\mu}, \theta, D)$
1: $maxGain \leftarrow 0$
2: $maxPat \leftarrow \emptyset$
3: $branch\_and\_bound(\emptyset, \boldsymbol{\mu}, \theta, D)$

function $branch\_and\_bound(p, \boldsymbol{\mu}, \theta, D)$
1: **for** $q \in \{$extended patterns of p in the canonical order$\}$ **do**
2: **if** $sup(q, D) \geq \theta$ **then**
3: **for** $c \in \{-1, +1\}$ **do**
4: **if** $gain(q, c; \boldsymbol{\mu}) > maxGain$ **then**
5: $maxGain \leftarrow gain(q, c; \boldsymbol{\mu})$
6: $maxPat \leftarrow (q, c)$
7: **end if**
8: **end for**
9: **if** $gainBound(p; \boldsymbol{\mu}) > maxGain$ **then**
10: $branch\_and\_bound(q, \boldsymbol{\mu}, \theta, D)$
11: **end if**
12: **end if**
13: **end for**

Theorem 1. *If $gainBound(p; \boldsymbol{\mu}) \leq g^*$ for some g^*, then $gain(q, c; \boldsymbol{\mu}) \leq g^*$ for all super-patterns q of p and all $c \in C$, where*

$$gainBound(p; \boldsymbol{\mu}) = \max\left(\sum_{\{i|y_i=+1\}} \mu_i \cdot occ(p, x'_i), \sum_{\{i|y_i=-1\}} \mu_i \cdot occ(p, x'_i)\right)$$

Proof. We will prove it by contradiction. Suppose that there is a super-pattern q of p such that $gain(q, c; \boldsymbol{\mu}) > gainBound(p; \boldsymbol{\mu})$. If $c = 1$,

$$\begin{aligned} gain(q, c; \boldsymbol{\mu}) &= \sum_{i=1}^{n} \mu_i y_i s_c(q, x'_i) = \sum_{i=1}^{n} \mu_i y_i occ(q, x'_i) \\ &= \sum_{\{i|y_i=1\}} \mu_i occ(q, x'_i) - \sum_{\{i|y_=-1\}} \mu_i occ(q, x'_i) \\ &\leq \sum_{\{i|y_i=1\}} \mu_i occ(q, x'_i) \leq \sum_{\{i|y_i=1\}} \mu_i occ(p, x'_i) \\ &\leq gainBound(p; \boldsymbol{\mu}) \end{aligned}$$

Therefore, it is a contradiction. Likewise, if $c = -1$, we can derive a similar contradiction. Note that $occ(q, x'_i) \leq occ(p, x'_i)$ because occ has *apriori property*.

If the maximum gain among the ones observed so far is greater than $gainBound(p; \boldsymbol{\mu})$, we can prune the branch of a pattern p. The optimal pattern search algorithm is presented in Algorithm 2.

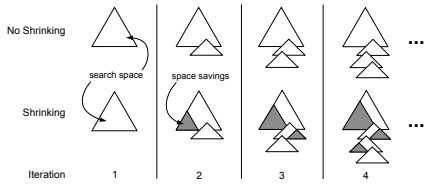


Fig. 1. Search Space Growth with and without Shrinking Technique. Dark regions represent shrinked search space (memory savings).

4.3 Search Space Shrinking Technique

In this section, we explain our novel search space shrinking technique. Mining discriminative patterns instead of frequent patterns can prune more search space by using a bound function $gainBound$. However, this requires an iterative procedure like in *DDPMine*, which builds a search space tree again and again. To avoid the repetitive searching, *gBoost* [18] stores the search space tree of previous iteration in main memory. The search space tree keeps expanding as iteration goes because it needs to mine different discriminative patterns. This may work for small datasets on a machine with enough main memory, but is not scalable.

In this paper, we also store the search space of previous iteration, but introduce search space shrinking technique to resolve the scalability issue.

In each iteration k of the column generation, we look for a pattern whose gain is greater than $\gamma^{(k-1)}$, otherwise the termination condition will hold. Thus, if a pattern p cannot have greater gain than $\gamma^{(k-1)}$, we do not need to consider p in the k^{th} iteration and afterwards because $\gamma^{(k)}$ is non-decreasing by the following theorem.

Theorem 2. *$\gamma^{(k)}$ is non-decreasing as k increases.*

Proof. In each iteration, we add a constraint that is violated by the previous optimal solution. Adding more constraints does not decrease the value of objective function in a minimization problem. Thus, $\gamma^{(k)}$ is not decreasing.

Definition 4. $maxGain(p) = \max_{\boldsymbol{\mu},c} gain(p, c; \boldsymbol{\mu})$, *where* $c \in C$, *and* $\forall i\ 0 \leq \mu_i \leq \omega$.

If there is a pattern p such that $maxGain(p) \leq \gamma^{(k)}$, we can safely remove the pattern from main memory after the k^{th} iteration without affecting the final result of *NDPMine*. By removing those patterns, we shrink the search space in main memory after each iteration. Also, since $\gamma^{(k)}$ increases during each iteration, we remove more patterns as k increases. This memory shrinking technique is illustrated in Figure 1.

In order to compute $maxGain(p)$, we could consider all the possible values of $\boldsymbol{\mu}$ by using linear programming. However, we can compute $maxGain(p)$ efficiently by using the greedy algorithm *greedy_maxGain* presented in Algorithm 3.

Theorem 3. *The greedy algorithm $greedy\_maxGain(p)$ gives the optimal solution, which is equal to $maxGain(p)$*

Algorithm 3 Greedy Algorithm for $maxGain$

Global Parameter: ω

function $greedy\_maxGain(p)$

1: $maxGain^+ \leftarrow greedy\_maxGainSub(p, +1)$
2: $maxGain^- \leftarrow greedy\_maxGainSub(p, -1)$
3: **if** $maxGain^+ > maxGain^-$ **then**
4: **return** $maxGain^+$
5: **else**
6: **return** $maxGain^-$
7: **end if**

function $greedy\_maxGainSub(p, c)$

1: $maxGain \leftarrow 0$
2: $weight \leftarrow 1$
3: $X \leftarrow \{x'_1, x'_2, \ldots, x'_n\}$
4: **while** $weight > 0$ **do**
5: $x'_{best} = \text{argmax}_{x'_i \in X}\, y_i \cdot s_c(p, x'_i)$
6: **if** $weight \geq \omega$ **then**
7: $maxGain \leftarrow maxGain + \omega \cdot y_{best} \cdot s_c(p, x'_{best})$
8: $weight \leftarrow weight - \omega$
9: **else**
10: $maxGain \leftarrow maxGain + weight \cdot y_{best} \cdot s_c(p, x'_{best})$
11: $weight \leftarrow 0$
12: **end if**
13: $X \leftarrow X - \{x'_{best}\}$
14: **end while**
15: **return** $maxGain$

Proof. Computing $maxGain(p)$ is very similar to continuous knapsack problem (or fractional knapsack problem) – one of the classic greedy problems. We can think our problem as follows: Suppose that we have n items, each with weight of 1 pound and a value. Also, we have a knapsack with capacity of 1 pound. We can have fractions of items as we want, but not more than ω. The only difference from continuous knapsack problem is that we need to have the knapsack full, and the values of items can be negative. Therefore, the proof of the optimality of $greedy\_maxGain$ can be easily derived from the proof of the optimality of the greedy algorithm for continuous knapsack problem.

5 Experiments

The major advantages of our method is that it is accurate, efficient in both time and space, produces small number of expressive features, and operates on different data types. In this section, we evaluate these claims by testing the accuracy, efficiency and expressiveness on two different data types: sequences and trees. For comparison-sake we re-implemented the two baseline approaches described in Section 5.1. All experiments are done on a 3.0GHz Pentium Core 2 Duo computer with 8GB main memory.

5.1 Comparison Baselines

As described in previous sections, *NDPMine* is the only algorithm that uses the direct approach to mine numerical features, therefore we compare *NDPMine* to the two-step process of mining numerical features in computation time and memory usage. Since we have two different types of datasets, sequences and trees, we re-implemented the two-step *SoftMine* algorithm by Lo *et al.* [12] which is only available for sequences. By showing the running time of *NDPMine* and *SoftMine*, we can appropriately compare the computational efficiency of direct and two-step approaches.

In order to show the effectiveness of the numerical feature values used by *NDPMine* over the effectiveness of binary feature values, we re-implemented the binary *DDPMine* algorithm by Cheng *et al.* [3] for sequences and trees. *DDPMine* uses the sequential covering method to avoid forming redundant patterns in a feature set. In the original *DDPMine* algorithm [3], both the Fisher score and information gain were introduced as the measure for discriminative power of patterns; however, for fair comparison of the effectiveness with *SoftMine*, we only use the Fisher score in *DDPMine*.

By comparing the accuracy of both methods, we can appropriately compare the numerical features mined by *NDPMine* with the binary features mined by *DDPMine*.

In order to show the effectiveness of the memory shrinking technique, we implemented our framework in two different versions, one with memory shrinking technique and another one without it.

5.2 Experiments on Sequence Datasets

Sequence data is a ubiquitous data structure. Examples of sequence data include text, DNA sequences, protein sequences, web usage data, and software execution traces. Among several publicly available sequence classification datasets, we chose to use software execution traces from [12]. These software trace datasets contained sequences of nine different software traces. More detail description of the software execution trace datasets is available in [12].

The goal of this classification task was to determine whether a program's execution trace (represented as an instance in the dataset) contains a failure or not. For this task, we needed to define what constitutes a pattern in a sequence and how to count the number of occurrences of a pattern in a sequence. We defined a pattern and the occurrences of a pattern the same as in [12].

5.3 Experiments on Tree Datasets

Datasets in tree structure are also widely available. Web documents in XML are good examples of tree datasets. XML datasets from [21] are one of the commonly used datasets in tree classification studies. However, we collected a very interesting tree dataset for authorship classification. In information retrieval and computational linguistics, authorship classification is one of the classic problems. Authorship classification aims to classify the author of a document. In order to attempt this difficult problem with our *NDPMine* algorithm, we randomly chose 4 authors – Jack Healy, Eric Dash, Denise Grady, and Gina Kolata – and collected 100 documents for each author from

NYTimes.com. Then, using the Stanford parser [19], we parsed each sentence into a tree of *POS*(Part of Speech) tags. We assumed that these trees reflected the author's writing style and thus could be used in authorship classification. Since a document consisted of several sentences, each document was parsed into a set of labeled trees.

From this we had a forest, or a set of trees, as a representative for a document where its author's name is the label. We used induced subtree patterns as features in classification. The formal definition of induced subtree patterns can be found in [4]. We defined the number of occurrences of a pattern in a document is the number of sentences in the document that contained the pattern. We employed *CMTreeMiner* [4], the-state-of-art tree mining algorithm, to mine frequent induced subtree patterns.

Since the goal of this classification task was to determine the author of each document, all pairs of authors and their documents were combined to make two-class classification dataset.

5.4 Parameter selection

Besides the definition of a pattern and the occurrence counting function for a given dataset, *NDPMine* algorithm needs two parameters as input: (1) the minimum support threshold θ and (2) the misclassification cost parameter ν. The θ parameter was given as input. The ν parameter was tuned in the same way SVM tunes its parameters: using cross-validation on the training dataset.

DDPMine and *SoftMine* are dependent on two parameters: (1) the minimum support threshold θ, and (2) the sequential coverage threshold δ. Because we were comparing these algorithms to *NDPMine* in accuracy and efficiency, for sequence and tree datasets, we selected parameters which were best suited to each task.

First, we fixed $\delta = 10$ for the sequence datasets as suggested in [12], and $\delta = 20$ for the tree datasets. Then, we found the appropriate minimum support θ in which *DDPMine* and *SoftMine* performed their best. Thus, we set $\theta = 0.05$ for the sequence datasets and $\theta = 0.01$ for the tree datasets.

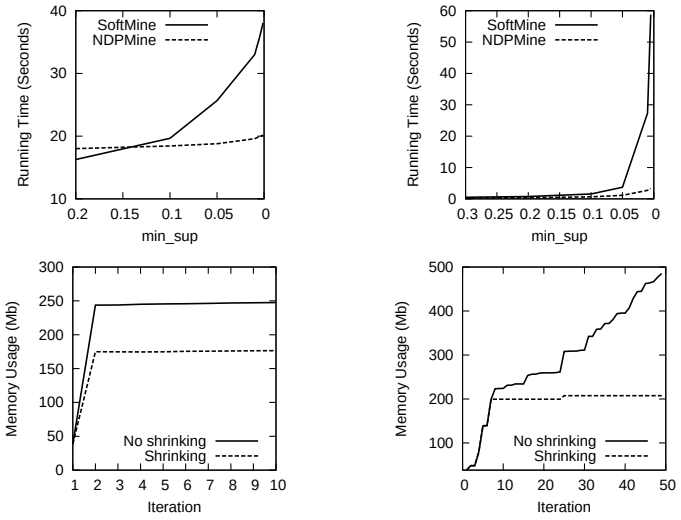
5.5 Computation Efficiency Evaluation

We discussed in Section 1 that some pattern-based classification models can be inefficient because they use the two-step mining process. We compared the computation efficiency of the two-step mining algorithm *SoftMine* with *NDPMine* as θ varies. The sequential coverage threshold is fixed to the value from Section 5.4. Due to the limited space, we only show the running time for each algorithm on the *schedule* dataset and the $\langle$D. Grady, G. Kolata$\rangle$ dataset in Figure 2. Other datasets showed similar results.

We see from the graphs in Figure 2 that *NDPMine* outperforms *SoftMine* by an order of magnitude. Although the running times are similar for larger values of θ, the results show that the direct mining approach used in *NDPMine* is computationally more efficient than the two-step mining approach used in *SoftMine*.

5.6 Memory Usage Evaluation

As discussed in Section 3, *NDPMine* uses memory shrinking technique which prunes the search space in main memory during each iteration. We evaluated the effective-



(a) Sequence (b) Tree

Fig. 2. Running Time and Memory Usage

ness of this technique by comparing the memory usage of *NDPMine* with the memory shrinking technique to *NDPMine* without the memory shrinking technique. Memory usage is evaluated in terms of the number of the size (in megabytes) of the memory heap. Figure 2 shows the memory usage time for each algorithm on *schedule* dataset and ⟨D. Grady, G. Kolata⟩ dataset. We set $\theta = 0$ in order to use as much memory as possible. We see from the graphs in Figure 2 that *NDPMine* with memory shrinking technique is more memory efficient than *NDPMine* without memory shrinking technique. Although the memory space expands roughly at the same rate initially, the search space shrinking technique begins to save space as soon as $\gamma^{(k)}$ increases. The difference between the sequence dataset and the tree dataset in Figure 2 is because the search spaces of the tree datasets are much larger than the search spaces of the sequence datasets.

5.7 Accuracy Evaluation

We discussed in Section 1 that some pattern-based classification algorithms can only mine binary feature values, and therefore may not be able to learn an accurate classification model. For evaluation purposes, we compared the accuracy of the classification model learned with features from *NDPMine* to the classification model learned with features from *DDPMine* and *SoftMine* for the sequence and tree datasets. After the feature set was formed, a SVM (from the LIBSVM [1] package) was used to learn a classification model. The accuracy of each model was also measured by 5-fold cross validation. Table 2 shows the results for each algorithm in the sequence datasets. Similarly, Table 3 shows the results in the tree datasets. The accuracy is defined as the number of true positives and true negatives over the total number of examples, and determined by 5-fold cross validation.

Table 2. The summary of results on software behavior classification

	Accuracy			Running Time		Number of Patterns	
Software	*DDPMine*	*SoftMine*	*NDPMine*	*SoftMine*	*NDPMine*	*SoftMine*	*NDPMine*
x11	93.2	**100**	**100**	0.002	0.008	17.0	6.6
cvs_omission	100	**100**	**100**	0.008	0.014	88.8	3.0
cvs_ordering	96.4	**96.7**	96.1	0.025	0.090	103.2	24.2
cvs_mix	96.4	94.2	**97.5**	0.020	0.061	34.6	10.6
tot_info	92.8	91.2	**92.7**	0.631	0.780	136.4	25.6
schedule	92.2	**92.5**	90.4	25.010	24.950	113.8	16.2
print_tokens	96.6	**100**	99.6	11.480	24.623	76.4	27.4
replace	85.3	**90.8**	90.0	0.325	1.829	51.6	15.4
mysql	100	95.0	**100**	0.024	0.026	11.8	2.0
Average	94.8	95.6	**96.2**	4.170	5.820	70.4	14.5

Table 3. The summary of results on authorship classification

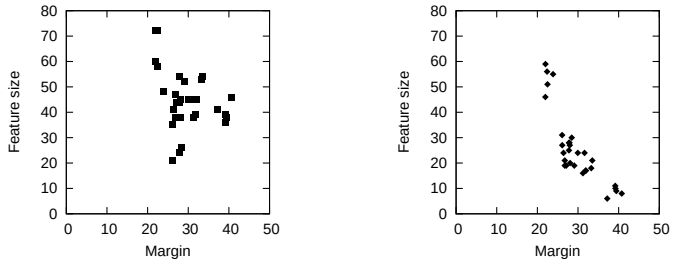
	Accuracy			Running Time		Number of Patterns	
Author Pair	*DDPMine*	*SoftMine*	*NDPMine*	*SoftMine*	*NDPMine*	*SoftMine*	*NDPMine*
⟨J. Healy, E. Dash⟩	89.5	91.5	**93.5**	43.83	1.45	42.6	24.6
⟨J. Healy, D. Grady⟩	94.0	94.0	**96.5**	52.84	1.26	47.2	19.4
⟨J. Healy, G. Kolata⟩	93.0	95.0	**96.5**	46.48	0.86	40.0	8.8
⟨E. Dash, D. Grady⟩	91.0	89.5	**95.0**	35.43	1.77	32.0	28.2
⟨E. Dash, G. Kolata⟩	92.0	90.5	**98.0**	45.94	1.39	43.8	18.8
⟨D. Grady, G. Kolata⟩	78.0	84.0	**86.0**	71.01	6.89	62.0	53.4
Average	89.58	90.75	**94.25**	49.25	2.27	44.6	25.53

These results confirm our hypothesis that numerical features, like those mined by *NDPMine* and *SoftMine*, may be used to learn more accurate models than binary features like those mined by *DDPMine*. We also confirm that feature selection by LP results in a better feature set than feature selection by sequential coverage.

5.8 Expressiveness Evaluation

We also see from the results in Tables 2 and 3 that the numbers of patterns mined by *NDPMine* are typically smaller than those of *SoftMine*, yet the accuracy is similar or better. Because *NDPMine* and *SoftMine* both use SVM and mine numerical features in common, we can conclude that the feature set mined by *NDPMine* must be more expressive than the features mined by *SoftMine*.

Also, we observed that *NDPMine* mines more discriminative patterns for harder classification datasets and fewer for easier datasets under the same parameters θ, ν. We measured this by the correlation between the hardness of the classification task and the size of feature set mined by *NDPMine*. Among several hardness measures [8] we determine the separability of two classes in a given dataset as follows: (1) mine all frequent patterns, (2) build a SVM-classifier with linear kernel, and (3) measure the margin of the classifier. Note that SVM builds a classifier by searching the classification boundary with maximum margin. The margin can be interpreted as the separability of two classes. If the margin is large, it implies that the classification task is easy. Next, we



(a) *SoftMine* (b) *NDPMine*

Fig. 3. The correlation between the hardness of Classification tasks and feature sizes

computed the correlation between the hardness of a classification task and the feature set size of *NDPMine* by using Pearson product-moment correlation coefficient (PMCC) – a widely used correlation measures in statistics. A larger PMCC implies stronger correlation; conversely, a PMCC of 0 implies that there is no correlation between two variables. We investigated on the tree dataset, and drew the 30 points in Figure 3 (there are six pairs of authors and each pair has 5 testdata). The result in Figure 3 shows a correlation of -0.831 for *NDPMine* and -0.337 for *SoftMine*. Thus, we confirmed that *NDPMine* mines more patterns if the given classification task is more difficult. This is a very desired property for discriminative pattern mining algorithms in pattern-based classification.

6 Conclusions

Frequent pattern-based classification methods have shown to be very effective at classifying large and complex datasets. Until recently, existing methods which mine a set of frequent patterns either use the two-step mining process which is computationally inefficient or can only operate on binary features. Due to the explosive number of potential features, the two-step process poses great computational challenges for feature mining. Conversely, those algorithms which use a direct pattern mining approach are not capable of mining numerical features. We showed that the number of occurrences of a pattern in an instance is more important than whether a pattern exists or not by extensive experiments on the software behavior classification and authorship classification datasets.

To our knowledge, there does not exist a discriminative pattern mining algorithm which can *both* directly mine discriminative patterns as numerical features. In this study, we proposed an pattern-based classification approach which efficiently mines discriminative patterns as numerical features for classification *NDPMine*. A linear programming method is integrated into the pattern mining process, and a branch and bound search is employed to navigate the search space. A shrinking technique is applied to the search space storage procedure which reduces the search space significantly. Although *NDPMine* is a model-based algorithm, the final output from the algorithm is a set of features that can be used independently for other classification models.

Experimental results show that *NDPMine* achieves: (1) orders of magnitude speedup over two-step methods without degrading classification accuracy, (2) significantly higher

accuracy than binary feature methods, and (3) better efficiency in space by using memory shrinking technique. In addition, we argue that the features mined by *NDPMine* can be more understandable than current techniques.

References

1. C.-C. Chang and C.-J. Lin. *LIBSVM: a Library for Support Vector Machines*, 2001. Software is available for download at `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.
2. H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *ICDE*, 2007.
3. H. Cheng, X. Yan, J. Han, and P. S. Yu. Direct discriminative pattern mining for effective classification. In *ICDE*, 2008.
4. Y. Chi, Y. Xia, Y. Yang, and R. R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(2):190–202, 2005.
5. A. Demiriz, K. P. Bennett, and J. Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254, 2002.
6. W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. S. Yu, and O. Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *KDD*, 2008.
7. Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
8. T. K. Ho and M. Basu. Complexity measures of supervised classification problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(3):289–300, 2002.
9. S. Levy and G. D. Stormo. Dna sequence classification using dawgs. In *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht*, pages 339–352, London, UK, 1997. Springer-Verlag.
10. W. Li, J. Han, and J. Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In *ICDM*, pages 369–376, 2001.
11. B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD*, pages 80–86, 1998.
12. D. Lo, H. Cheng, J. Han, S.-C. Khoo, and C. Sun. Classification of software behaviors for failure detection: A discriminative pattern mining approach. In *KDD*, 2009.
13. S. G. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, New York, NY, 1996.
14. S. Nijssen, T. Guns, and L. D. Raedt. Correlated itemset mining in roc space: a constraint programming approach. In *KDD*, 2009.
15. S. Nowozin and K. T. Gökhan Bakõr. Discriminative subsequence mining for action classification. In *ICCV*, 2007.
16. H. Saigo, T. Kadowaki, T. Kudo, and K. Tsuda. A linear programming approach for molecular qsar analysis. In *MLG*, pages 85–96, 2006.
17. H. Saigo, N. Krämer, and K. Tsuda. Partial least squares regression for graph mining. In *KDD*, 2008.
18. H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda. gboost: a mathematical programming approach to graph classification and regression. *Mach. Learn.*, 75(1):69–89, 2009.
19. The Stanford Natural Language Processing Group. *The Stanford Parser: A statistical parser*. `http://www-nlp.stanford.edu/software/lex-parser.shtml`.
20. L. Ye and E. Keogh. Time series shapelets: a new primitive for data mining. In *KDD*, 2009.
21. M. J. Zaki and C. C. Aggarwal. Xrules: an effective structural classifier for xml data. In *KDD*, 2003.