Direct Discriminative Pattern Mining for Effective Classification

Hong Cheng [#1], Xifeng Yan [*2], Jiawei Han [#3], Philip S. Yu [+4]

[#] *University of Illinois at Urbana-Champaign*
Urbana, IL USA
[1] hcheng3@uiuc.edu
[3] hanj@cs.uiuc.edu

[*] *IBM T. J. Watson Research Center*
Hawthorne, NY USA
[2] xifengyan@us.ibm.com

[+] *University of Illinois at Chicago*
Chicago, IL USA
[4] psyu@cs.uic.edu

***Abstract*— The application of frequent patterns in classification has demonstrated its power in recent studies. It often adopts a two-step approach: frequent pattern (or classification rule) mining followed by feature selection (or rule ranking). However, this two-step process could be computationally expensive, especially when the problem scale is large or the minimum support is low. It was observed that frequent pattern mining usually produces a huge number of "patterns" that could not only slow down the mining process but also make feature selection hard to complete.**

In this paper, we propose a direct discriminative pattern mining approach, DDPMine, to tackle the efficiency issue arising from the two-step approach. DDPMine performs a branch-and-bound search for directly mining discriminative patterns without generating the complete pattern set. Instead of selecting best patterns in a batch, we introduce a "feature-centered" mining approach that generates discriminative patterns sequentially on a progressively shrinking FP-tree by incrementally eliminating training instances. The instance elimination effectively reduces the problem size iteratively and expedites the mining process. Empirical results show that DDPMine achieves orders of magnitude speedup without any downgrade of classification accuracy. It outperforms the state-of-the-art associative classification methods in terms of both accuracy and efficiency.

I. Introduction

Frequent pattern-based classification has been explored in recent years and its power was demonstrated by multiple studies in several domains, including (1) associative classification [1], [2], [3], [4], [5], [6] on categorical data, where a classifier is built based on high-support, high-confidence association rules; and (2) frequent pattern-based classification [7], [8], [9], [10], [11] on text or data with complex structures such as sequences and graphs, where discriminative frequent patterns are taken as features to build high quality classifiers. A frequent itemset (pattern) is a set of items that occur in a dataset no less than a user-specified minimum support (*min_sup*). Frequent patterns have been explored widely in classification tasks.

These studies achieve promising classification accuracy and demonstrate the success of frequent patterns (or association rules) in classification. For example, in [1], [2], [3], [5], [6], associative classification was found to be competitive with traditional classification methods, such as C4.5 and SVM, sometimes even better on categorical datasets [4]. In addition, frequent patterns are also promising for classifying complex structures such as strings and graphs [8], [9], [10], [11] with high accuracy.

Most of these studies [1], [2], [4], [11], [10] take a *two-step* process: First mine all frequent patterns or association rules which satisfy *min_sup* and then perform a feature selection or rule ranking procedure. Figure 1 (A) shows the flow of the two-step framework, where a dark circle represents one discriminative pattern. Although the approach is straightforward and achieves high classification accuracy, it could incur high computational cost. The efficiency issues exist in the following two aspects.

First, frequent pattern mining could take a long time to complete due to the exponential combinations among items, which is common for dense datasets or high-dimensional microarray data. When the problem scale is large or *min_sup* is low, it could take forever to complete the mining. It often turns out that the mining results, even those for closed frequent itemsets, are explosive in size.

More importantly, the classification tasks attach great importance to the frequent itemsets that are highly discriminative w.r.t. the class labels. Since frequent itemsets are generated solely based on support information, not based on discriminative power, a large number of indiscriminative itemsets can be generated during the mining step. When the complete mining results are prohibitively large, yet only the highly discriminative ones are of real interest, it is inefficient to wait forever for the mining algorithm to finish and then apply feature selection to post-process the huge-sized mining results. Even for a feature selection algorithm with linear complexity, it could be very expensive to process an explosive number, such as millions, of features which is a common scale in

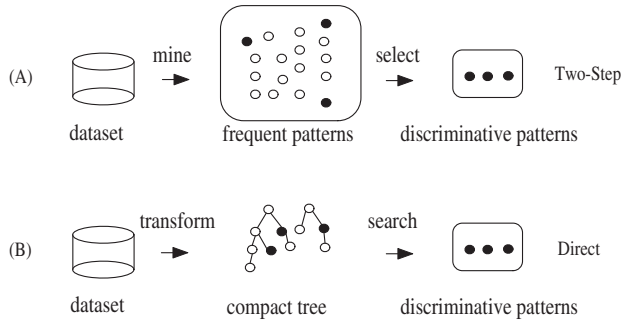


Fig. 1. Two-Step Approach vs. DDPMine

frequent patterns.

The computational cost raised by the two-step framework motivates us to investigate an alternative approach: *Instead of generating the complete set of frequent patterns, directly mine highly discriminative patterns for classification.* This leads to our proposal of a *direct discriminative pattern mining* approach, DDPMine. It integrates the feature selection mechanism into the mining framework by directly mining the most discriminative patterns, and then incrementally eliminating the training instances which are covered by those patterns. Figure 1 (B) illustrates the DDPMine mining methodology which first transforms data into a compact FP-tree ([12]) and then searches discriminative patterns directly. DDPMine is shown to outperform the two-step method with significant speedup.

Our contributions are summarized as follows.

- We examine the efficiency issue that arises from the two-step mining framework and propose a direct mining solution.
- A direct discriminative pattern mining algorithm DDPMine is proposed. DDPMine not only avoids generating a large number of indiscriminative patterns, but also incrementally reduces the problem size by eliminating training instances and progressively shrinking the FP-tree, which further speeds up the mining process.
- Instead of mining a set of discriminative patterns in a batch, our "feature-centered" mining approach exploited by DDPMine could single out patterns sequentially in the progressively shrinking FP-tree which is shown to be very efficient.
- Time complexity analysis is provided for the DDPMine algorithm. An upper bound is derived on the number of iterations it has to go through, showing the computational cost analytically.
- Our empirical results show that DDPMine achieves orders of magnitude speedup without any downgrade of classification accuracy. Moreover, it outperforms the state-of-the-art associative classification methods in terms of both accuracy and efficiency.

Together with our previous work [11], we demonstrated that frequent pattern-based classification is not only **accurate** but also **scalable**.

II. Notations and Definitions

Let D be a training database, $I = \{i_1, i_2, \ldots, i_m\}$ be the set of distinct items, and $C = \{c_1, c_2, \ldots, c_k\}$ be the set of class labels. Assume D contains a set of n training instances $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$, where $\boldsymbol{x}_i \subseteq I$ is a set of items and $y_i \in C$ is a class label.

An itemset $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_l\}$ is a subset of I. Given a dataset $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$, the set of data that contains α is denoted as $D_\alpha = \{(\boldsymbol{x}_i, y_i) | \alpha \subseteq \boldsymbol{x}_i\}$. α is *frequent* if $\theta = \frac{|D_\alpha|}{|D|} \geq \theta_0$, where $\theta = \frac{|D_\alpha|}{|D|}$ is the relative support of α, and θ_0 is the *min_sup* threshold, $0 \leq \theta_0 \leq 1$. The set of frequent itemsets is denoted as F.

III. Frequent Pattern-based Classification

Frequent Pattern-Based Classification is learning a classification model in the feature space of single features as well as frequent patterns. The framework of frequent pattern-based classification basically includes three steps: (1) frequent itemset mining, (2) feature selection, and (3) model learning. The first step generates frequent patterns (or itemsets), on which a feature selection algorithm is applied to single out a compact set of discriminative patterns. Then the training data is represented in the feature space of those discriminative patterns. Finally, a classification model is constructed. Any learning algorithm could be used as the classification model. Both frequent itemset mining and feature selection steps in the classification framework could potentially be a computational bottleneck, due to the exponential combination among items.

A. Computational Bottleneck

The frequent pattern mining step could take a very long time to generate the complete set of frequent patterns. When the complete mining results are prohibitively large on dense datasets, of which only a small number are truly discriminative for classification, it is inefficient to wait forever for the mining algorithm to finish and then apply feature selection to post-process the huge-sized mining results.

In addition to pattern mining, feature selection could be computational bottleneck too. The goal of feature selection in this framework is to select highly discriminative features. Due to an explosive number of frequent patterns, any feature selection algorithm with polynomial time complexity can hardly scale. Even if a linear algorithm is employed, it could still run slowly. According to our experiments in [11], a low *min_sup* generates over millions of frequent itemsets, on top of which feature selection algorithms fail to complete in days.

B. Feature Selection by Sequential Coverage

In this study, we explore a direct mining approach for a general feature selection algorithm that is based on the "sequential coverage paradigm" [13]. The sequential coverage algorithm takes as input a set of training instances D and a set of features F, and iteratively applies the feature selection step. At each step, the algorithm selects the feature with the highest discriminative measure. After selecting this feature, all the training instances containing this feature are eliminated

from D and the feature is marked as selected. In the next iteration, the same step is applied, but on a smaller set of training examples. This algorithm continues in an iterative fashion until either all the training examples are eliminated or all the features are selected.

Algorithm 1 Feature Selection

Input: Frequent patterns F, Training database D
Output: A selected pattern set F_s

1: $F_s := \varnothing$;
2: **while** (true)
3: Find the best pattern α in $F - F_s$;
4: If α can correctly cover at least one instance in D
5: $F_s := F_s \cup \{\alpha\}$;
6: $F := F - \{\alpha\}$;
7: $D := D - D_\alpha$;
7: If $D = \varnothing$ or $F = \varnothing$
8: break;
9: **return** F_s

Algorithm 1 sketches the feature selection process in a set of frequent patterns. The criterion of "best pattern" in Line 3 could be information gain [14], fisher score [15] or others. The time complexity of this sequential coverage algorithm is $O(|F| \cdot |D|)$, where $|F|$ is the size of feature set and $|D|$ is the size of training data.

IV. Direct Discriminative Pattern Mining

In the DDPMine approach, there are two objectives we want to achieve: (1) for efficiency concerns, we want to directly mine a set of highly discriminative patterns; and (2) for accuracy consideration, we impose a *feature coverage* constraint: every training instance has to be covered by one or multiple features.

DDPMine developed two modules to meet these two objectives: (1) a branch-and-bound search method to identify the most discriminative pattern in a data set; (2) an instance elimination process to remove the training instances that are covered by the patterns selected so far. The branch-and-bound search algorithm is based on the upper bound estimation of discriminative measures derived from our previous work [11], which is able to prune the search space effectively.

DDPMine progressively reduces the dataset size by iteratively eliminating training instances. This expedites the mining process since the mining complexity is closely related to the dataset size.

Both processes are actually implemented in a compact tree structure, FP-Tree, and are able to avoid the generation of the complete pattern set.

A. Branch-and-Bound Search

An upper bound of discriminative measures such as *information gain* [14] was derived by [11] which is a function of pattern frequency. The discriminative power of low-frequency patterns is upper bounded by a small value. Based on this conclusion, we design a branch-and-bound search for directly mining discriminative patterns and pruning the indiscriminative ones. We adopt FP-growth [12] as the basic mining methodology and show how to incorporate the theoretical upper bound to facilitate a branch-and-bound search. For details of FP-growth mining, please refer to [12].

Algorithm 2 The Branch-and-bound Mining Algorithm

Input: An FP-tree P, *min_sup* s, a prefix α
Output: The most discriminative feature $bestPat$
Global variable: $maxIG := 0$, $bestPat := null$

Procedure $branch\_and\_bound(P, s, \alpha)$
1: **if** $P = \varnothing$
2: **return**;
3: **for** each item a_i in P **do**
4: generate pattern $\beta = a_i \cup \alpha$ with support=a_i.support;
5: compute information gain $IG(\beta)$;
6: **if** $IG(\beta) > maxIG$
7: $maxIG := IG(\beta)$;
8: $bestPat := \beta$;
9: construct pattern β's conditional database D_β;
10: $IG_{ub}(|D_\beta|) := upper\_bound(|D_\beta|)$;
11: **if** $maxIG \geq IG_{ub}(D_\beta)$
12: skip mining on D_β;
13: **else**
14: construct β's conditional FP-tree P_β;
15: $branch\_and\_bound(P_\beta, s, \beta)$;

The basic idea is, during the recursive FP-growth mining, we use a global variable to record the most discriminative itemset discovered so far and its information gain score. Before proceeding to construct a conditional FP-tree, we first estimate the upper bound of information gain, given the size of the conditional database. Since the support of any itemset from this conditional database cannot be greater than the conditional database size, the information gain of any itemset from this conditional database is bounded by the upper bound value. If the upper bound value is no greater than the current best value, we could safely skip this conditional FP-tree as well as any FP-tree recursively constructed from this one. Algorithm 2 shows the branch-and-bound mining algorithm. $IG(\beta)$ on line 6 is the information gain of frequent pattern β and $IG_{ub}(|D_\beta|)$ on line 10 is the information gain upper bound given the conditional database D_β. The upper bound formulae were derived in [11] and are provided in Appendix of this paper.

We will illustrate this method through the following example. Table I shows a training database which contains eight instances and two classes. Let $min\_sup = 2$. The global FP-tree is illustrated in Figure 2. The FP-tree is a compact prefix-tree structure. A node represents an item with the count and a path represents a transaction.

The first frequent itemset generated is d with an informa-

TID	Set of Items	Class Label
100	a, b, c	1
200	a, b, c, d	1
300	a, b, c	1
400	a, b, d	1
500	c, d	0
600	b, c	0
700	a, b, c	1
800	a, b, c	1

TABLE I
A SAMPLE TRAINING DATABASE D

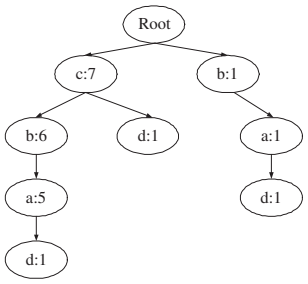


Fig. 2. The Global FP-tree

tion gain value $IG(d) = 0.016$. Then $maxIG$ is assigned 0.016. The conditional database and FP-tree on d is shown in Figure 3. Given the size of the conditional database is 3, the information gain upper bound is $IG_{ub}(3) = 0.467$. Since $IG_{ub}(3) > maxIG$, we cannot prune the conditional FP-tree on d. Therefore, we perform recursive mining on the conditional FP-tree and get ad, bd, cd and abd, with $IG(ad) = 0.123$, $IG(bd) = 0.123$, $IG(cd) = 0.074$ and $IG(abd) = 0.123$.

As the mining proceeds to the frequent itemset a, we can compute its information gain $IG(a) = 0.811$ which is assigned to $maxIG$ as well. The conditional database and FP-tree on a is shown in Figure 4. Given the size of the conditional database is 6, the information gain upper bound is $IG_{ub}(6) = 0.811$. Since $maxIG = IG_{ub}(6)$, any itemset generated from the conditional FP-tree will have an information gain no greater than $maxIG$. Therefore, the conditional FP-tree can be pruned without any mining. To confirm our analysis, we could double check the actual mining results from this conditional FP-tree: ab, ac, ad, abd and abc. A careful verification shows that the information gain of all these itemsets is no greater than $maxIG$, which is consistent with our pruning decision.

B. Training Instance Elimination

The branch-and-bound search directly mines the discriminative patterns and effectively prunes the search space. To achieve the feature coverage objective that ensures every training instance is covered by one or multiple features, there are two different approaches: a *transaction-centered* approach and a *feature-centered* approach. We first present the transaction-centered approach and analyze why it does not work efficiently. Then we propose the feature-centered approach which is a core part of DDPMine.

Items	Class Label
a, b, c	1
a, b	1
c	0

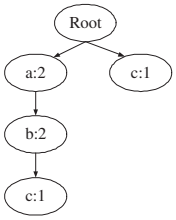


Fig. 3. Conditional DB and FP-tree on d

Items	Class Label
b, c	1
b, c, d	1
b, c	1
b, d	1
b, c	1
b, c	1

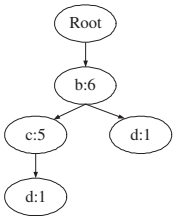


Fig. 4. Conditional DB and FP-tree on a

1) Transaction-Centered Approach: The transaction-centered approach is to mine a set of discriminative features to satisfy the feature coverage constraint for each training instance.

The transaction-centered approach could be built on FP-growth mining with some additional operations. It keeps the best feature generated so far for each transaction. When a frequent pattern β is generated, the transaction id list $T(\beta)$ is computed. We go through every instance $t \in T(\beta)$ and check whether β is the best feature for t. To get $T(\beta)$, we have to check the training database. Therefore, the cost associated with this approach is

$$Cost = T_{mining} + m \cdot T_{check\_db} \quad (1)$$

where T_{mining} is the time of FP-growth mining and T_{check_db} is the time to get $T(\beta)$ for a frequent itemset β. m is the number of frequent itemsets generated. Since the number of frequent itemsets is usually very huge, even explosive, m is a very large factor. In our experiments, we observe that m is usually between hundreds of thousands to several million. Table II shows the mining time, the total $check\_db$ time as well as the number of itemsets m on several large datasets. As shown in the table, m is usually very huge and the $check\_db$ time is at least two orders of magnitude larger than the mining time. Therefore, the $check\_db$ operation becomes the major computational bottleneck due to the large factor m, which makes the transaction-centered approach inefficient.

TABLE II
RUNTIME OF TRANSACTION-CENTERED APPROACH

Dataset	*min_sup*	T_{mining}	$m \cdot T_{check\_db}$	m
adult	50	0.203	355.648	22228
chess	1000	9.219	883.971	844258
hypo	1000	1.766	158.995	153412
sick	1000	6.531	454.379	495380

Algorithm 3 The DDPMine Algorithm

Input: An FP-tree P, *min_sup* s
Output: A set of selected features F_s

Procedure DDPMine(P, s)
1: **if** $P = \varnothing$
2: **return**;
3: $\alpha := branch\_and\_bound(P, s, null)$;
4: **if** $\alpha = null$
5: **return**;
6: Compute the transaction id list $T(\alpha)$ containing α;
7: $P' := update\_tree(P, T(\alpha))$;
8: $F_s := \{\alpha\} \cup$ DDPMine(P', s);
9: **return** F_s;

2) Feature-Centered Approach – Our Choice: In light of the cost associated with the transaction-centered approach, we want to directly mine discriminative features while reducing the number of $check\_db$ operations. Therefore, we propose a *feature-centered* approach which, as suggested by the name, primarily focuses on the discriminative features.

The basic idea is, a branch-and-bound search produces the most discriminative itemset α but the mining process does not compute any transaction id at all. After mining, the transaction id list $T(\alpha)$ is computed. Then we eliminate the transactions in $T(\alpha)$ from the FP-tree and repeat the branch-and-bound search on the updated tree. This process iterates until all transactions are removed from the FP-tree.

This approach is feature-centered in the sense that the mining process only concerns mining the most discriminative pattern. The $chech\_db$ operation is not performed on any intermediate mining results, but only on the best feature produced by the mining process.

The DDPMine algorithm, which integrates the branch-and-bound search and the feature-centered approach, is presented in Algorithm 3. It takes two inputs: an FP-tree and *min_sup*. An initial FP-tree is constructed from the training database. $branch\_and\_bound$ searches the most discriminative feature α. Then the transaction set $T(\alpha)$ containing α is computed and removed from P. The resulting FP-tree is P'. Then DDPMine is recursively invoked on P' until the FP-tree becomes empty. If the $branch\_and\_bound$ search function fails to discover any feature w.r.t. *min_sup* in the current FP-tree, the whole procedure terminates. The final output is the set of frequent itemsets generated by the iterative mining process.

Correspondingly, the cost associated with DDPMine is

$$Cost = n \cdot (T_{mining} + T_{check\_db} + T_{update}) \qquad (2)$$

where n is the number of iterations which is usually very small. We will derive an upper bound of n in Section IV-C. T_{update} is the time to update the FP-tree. We design an efficient method for the update operation in Section IV-B.3.

3) Progressively Shrinking FP-Tree: One step in the DDPMine algorithm is *update_tree*, which removes the set of training instances $T(\alpha)$ containing the feature α from the FP-tree. We design an efficient method for update_tree operation with the corresponding data structure.

When we insert a training instance into an FP-tree, we register the transaction id of this instance at the node which corresponds to the very last item in the instance. Accordingly, the FP-tree carries training instance id lists. For efficiency concern, the id lists are only registered with the global FP-tree, but not propagated in the conditional FP-trees when performing the recursive FP-growth mining.

When a frequent itemset α is generated, the training instances $T(\alpha)$ have to be removed from the global FP-tree. Then we perform a traversal of the FP-tree and examine the id lists associated with the tree nodes. When an id in a node appears in $T(\alpha)$, this id is removed. Correspondingly, the count on this node is reduced by 1, as well as the count on all the ancestor nodes up to the root of the tree. When the count reaches 0 at any node, the node is removed from the FP-tree. Therefore, the $update$ operation basically is a traversal of the FP-tree, the complexity of this operation is

$$T_{update} = O(|V| + |D|) \qquad (3)$$

where $|V|$ is the number of nodes in the FP-tree and $|D|$ is the number of training instances in the database.

In our previous example, when we discover the itemset a with $T(a) = \{100, 200, 300, 400, 700, 800\}$, $T(a)$ are removed from the FP-tree. Figure 5 shows the updated tree where the gray nodes are the nodes with 0 count and will be removed from the updated tree. The rectangle boxes are the transaction id lists associated with the nodes. Since the global FP-tree is updated incrementally in each iteration, we call it *progressively shrinking FP-tree*.

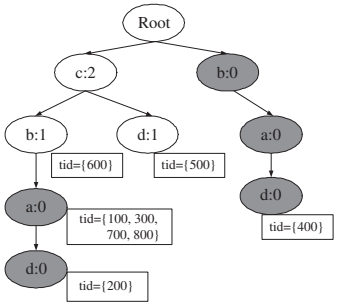


Fig. 5. The Updated FP-tree with TID

4) Feature Coverage: In the DDPMine algorithm, when a feature is generated, the transactions containing this feature are removed. In real classification tasks, we may want to generate multiple features to represent a transaction for accuracy consideration. To realize this purpose, we introduce a feature coverage parameter δ: A transaction is eliminated from further consideration when it is covered by at least δ features. This feature could be easily integrated into DDPMine with some minor changes in the data structure: We keep a counter for each transaction. Whenever a feature is generated, the counter

for each transaction containing this feature is incremented by one. When a counter reaches δ, the corresponding transaction is removed from the tree. The counters are stored in an array of integers, called *CTable*.

Besides the counter, we need to keep a global hash table, called *HTable*, to keep track of the features that are already discovered. When $\delta > 1$, a transaction will not be eliminated unless the counter reaches δ. As a result, the FP-tree may remain unchanged when no transactions are eliminated in one iteration. In such a case, we need to use a hash table to keep track of the features that are already discovered and thus avoid generating the same features multiple times. Let's follow the example in Table I and assume $\delta = 2$. In the first iteration, we generate the feature a. The CTable and HTable are shown in Figure 6. Since no counter reaches $\delta = 2$, no transaction is removed from the FP-tree. Thus, it remains unchanged.

TID	Count
100	1
200	1
300	1
400	1
500	0
600	0
700	1
800	1

FID	Items	Info Gain
1	a	0.811

Fig. 6. CTable and HTable at Iteration 1

TID	Count
100	2
200	2
300	2
400	2
500	0
600	0
700	2
800	2

FID	Items	Info Gain
1	a	0.811
2	ab	0.811

Fig. 7. CTable and HTable at Iteration 2

In the following iterations, the HTable will be checked for duplication whenever a new feature is discovered. If there is a "hit" in the HTable, the new feature will be ignored and the mining proceeds. In the second iteration of this example, the mining algorithm first discovers a which exists in HTable already. Then the algorithm ignores it and proceeds. Finally, it generates ab as the most discriminative feature. After ab is generated, the CTable and HTable are changed and shown in Figure 7. Accordingly, transactions $\{100, 200, 300, 400, 700, 800\}$ are removed from the FP-tree.

C. Efficiency Analysis

DDPMine works in an iterative way and terminates when the training database becomes empty. We derive an upper bound of the number of iterations DDPMine has to go through.

Assume *min_sup* $= \theta_0$. DDPMine produces a frequent itemset α_i in the i-th iteration, $sup(\alpha_i) \geq \theta_0$. In the i-th iteration, we eliminate the training instances $T(\alpha_i)$ from the current set of training instances since they are covered by the feature α_i. Therefore, we have the following equation which specifies the reduction of the training instance database:

$$|D_i| = |D_{i-1}| - |T(\alpha_i)| \tag{4}$$

where D_i is the training instances remaining after the i-th iteration, $T(\alpha_i)$ is the id list of transactions which contain α_i, and D_0 is the complete set of training instances.

Since $sup(\alpha_i) \geq \theta_0$, we have $|T(\alpha_i)| \geq \theta_0 |D_{i-1}|$ in equivalence. Then we have

$$|D_i| = |D_{i-1}| - |T(\alpha_i)| \leq (1 - \theta_0)|D_{i-1}| \tag{5}$$

According to Eq. (5), we have

$$|D_i| \leq (1 - \theta_0)^i |D_0| \tag{6}$$

Assume after n iterations, the training database reduces to $|D_n| = 1$. Since $(1 - \theta_0)^n |D_0| \geq |D_n| = 1$, we can derive

$$n \leq \frac{log|D_0|}{log\frac{1}{1-\theta_0}} = log_{\frac{1}{1-\theta_0}}|D_0| \tag{7}$$

According to Eq. (7), if $\theta_0 = 0.5$, $n \leq log_2|D_0|$. If $\theta_0 = 0.2$, $n \leq log_{1.25}|D_0|$. If the training database has 1 million instances, then $n \leq 20$ if $\theta_0 = 0.5$; $n \leq 62$ if $\theta_0 = 0.2$.

The above bound analysis assumes the feature coverage parameter $\delta = 1$. If $\delta > 1$, the bound of n becomes

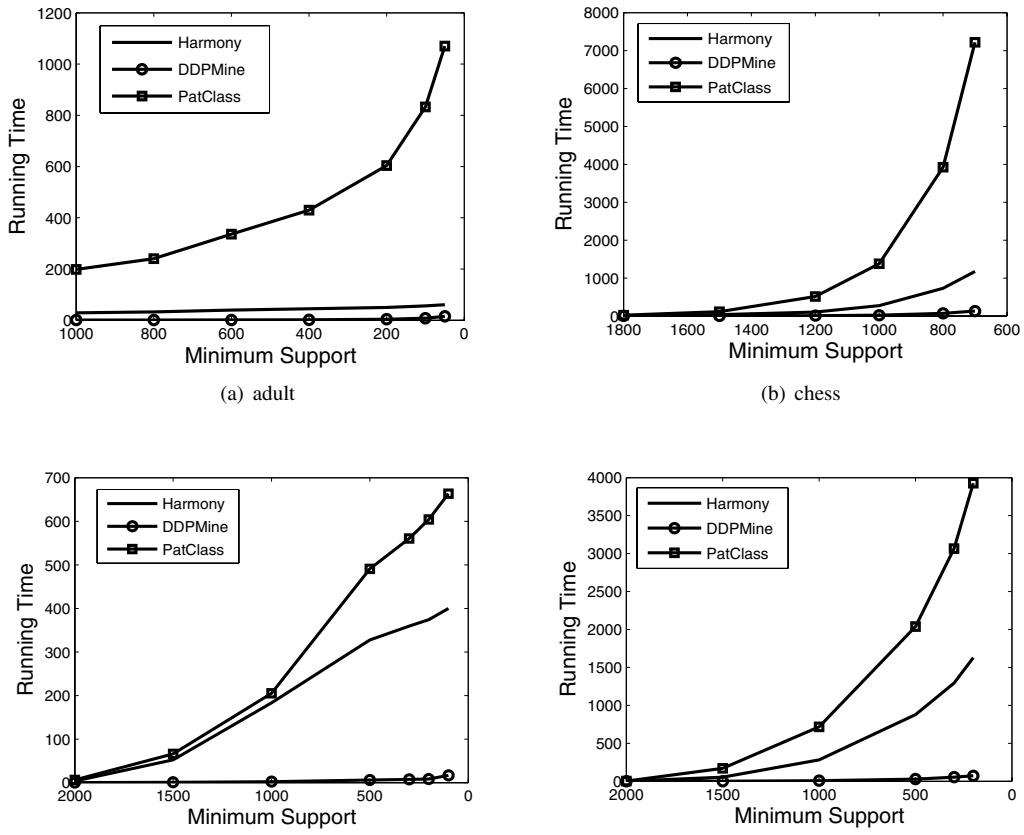
$$n \leq \delta \cdot \frac{log|D_0|}{log\frac{1}{1-\theta_0}} = \delta \cdot log_{\frac{1}{1-\theta_0}}|D_0| \tag{8}$$

Eqs. (7) and (8) provide an upper bound of the number of iterations. In each iteration, the major computational cost is the frequent itemset mining. But it is more efficient than the original FP-growth mining, since it has the branch-and-bound pruning mechanism. In addition, the mining becomes more and more efficient as the training database shrinks. Combining Eqs. (2) and (8), the cost of DDPMine is

$$Cost \leq \delta \cdot log_{\frac{1}{1-\theta_0}}|D_0| \cdot (T_{mining} + T_{check\_db} + T_{update}) \tag{9}$$

V. Experimental Results

We conduct a systematic experimental study to evaluate DDPMine on both efficiency and accuracy. A series of datasets from UCI Machine Learning Repository are tested. The implementation of the branch-and-bound mining is done based on FPClose [16]. LIBSVM [17] is used as the classification model. 5-fold cross validation is used for evaluation: Each dataset is partitioned into five parts evenly. Each time, one part is used for test and the other four are used in DDPMine for mining and training. The classification accuracies on the five test sets are averaged. The algorithm is implemented in Microsoft Visual C++ and experiments were run on a 3GHz PC with 1GB memory.



(a) adult

(b) chess

(c) hypo

(d) sick

Fig. 8. Efficiency Tests

A. Efficiency Evaluation

1) Efficiency Comparison: We test the efficiency of DDPMine as *min_sup* varies. For comparison, we tested two other methods.

- HARMONY [5] which is an instance-centric rule-based classifier. It directly mines the final set of classification rules. By introducing several novel search strategies and pruning methods into the rule discovery process, HARMONY has shown high efficiency and better classification accuracy than CPAR [3] and SVM.
- PatClass [11] which is a frequent pattern-based classifier. It takes the two-step procedure by first mining a set of frequent itemsets followed by a feature selection step. This method has shown to achieve high classification accuracy. LIBSVM is used as the classification model in PatClass as well.

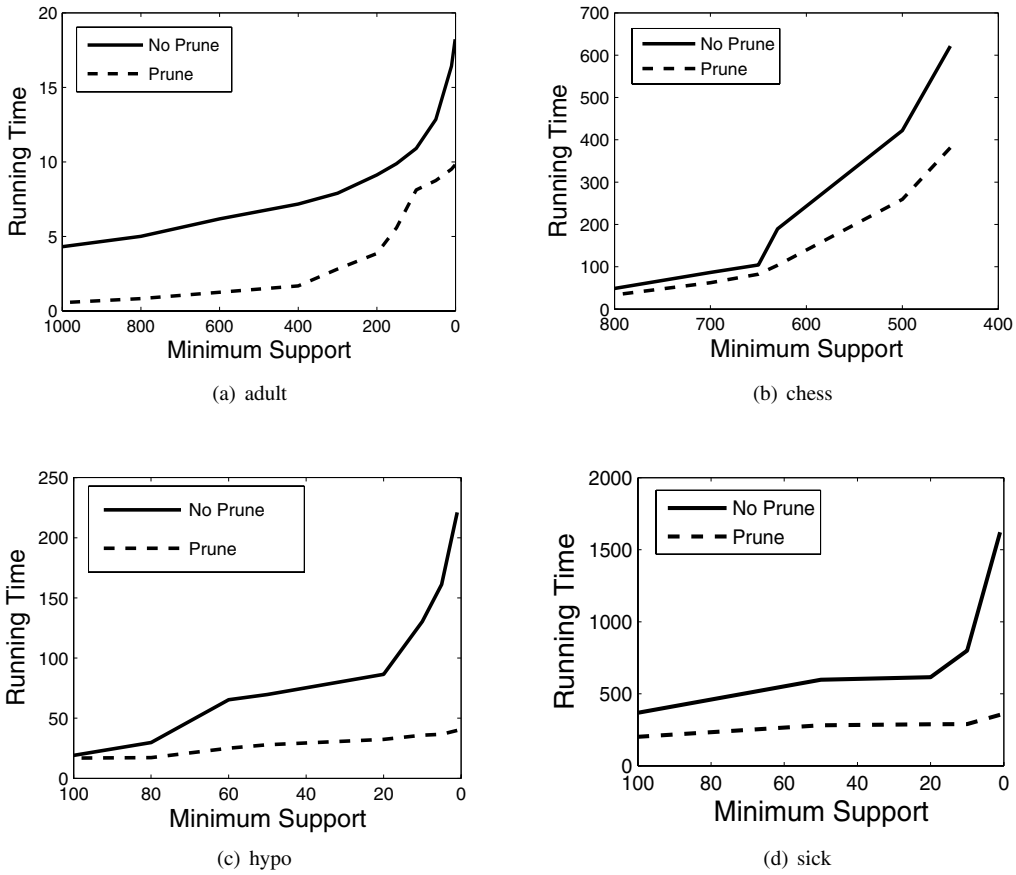
In this experiment, we choose four large datasets: *adult*, *chess*, *hypo* and *sick*. We tested the running time of these three methods as *min_sup* varies. Figure 8 shows the results on four datasets. On all four datasets in Figure 8, it is clear that DDPMine is the most efficient method. It outperforms both HARMONY and PatClass by an order of magnitude or even more. In addition, the running time of DDPMine is not affected much by *min_sup*. A reasonable explanation for this property is the effect of the training instance elimination: The problem size progressively shrinks which expedites the mining process.

On the other hand, PatClass is the least efficient method. The performance of PatClass is very sensitive to *min_sup*: as *min_sup* lowers down, the running time increases dramatically, due to an explosive set of frequent itemsets produced. Besides the mining process, feature selection also slows down since the set of frequent itemsets as input is bulky.

HARMONY stands in the middle: it is quite efficient when *min_sup* is high but slows down as *min_sup* decreases. Compared with PatClass, it is still much more efficient, due to the pruning strategies it exploits.

2) Branch and Bound Search: We also evaluate the effectiveness of the pruning by the branch-and-bound search. Figure 9 shows the running time of DDPMine with and without the branch-and-bound pruning respectively, as *min_sup* varies. As we can see from Figure 9, when the pruning strategy is used in the mining process, the efficiency gain is at least $1/3$, usually between $1/2$–$2/3$, especially when *min_sup* is low.

3) Problem Size Reduction: In this experiment, we test how fast the problem size reduces in an iterative fashion in DDPMine. We tested on the four large datasets again and set *min_sup* to be 20%. We run the DDPMine algorithm and record the database size (in terms of the number of training



(a) adult

(b) chess

(c) hypo

(d) sick

Fig. 9. Branch and Bound Search

instances) remaining in each iteration. Figure 10 shows the problem size reduction curve. As we can see clearly from Figure 10, the problem size shrinks very quickly. Usually by 10 or fewer iterations, the database reduces to empty and the program terminates. In addition, we observed that, as the problem size reduces incrementally, the branch-and-bound FP-growth mining becomes more and more efficient even though the relative *min_sup* is the same. This is because mining efficiency is also closely related to the data set size.

B. Efficiency and Accuracy: UCI datasets

We evaluate DDPMine on a series of UCI datasets in terms of both efficiency and accuracy, by comparing with HARMONY and PatClass, which are the state-of-the-art associative classification methods.

Table III shows the running time (in seconds) of the three methods. For both DDPMine and PatClass we compute the running time for both frequent itemset mining and feature selection; whereas for HARMONY, we compute the running time for association rule mining. The running time was averaged over 5-fold cross validation.

From Table III, we can see that, DDPMine is the most efficient algorithm, followed by HARMONY while PatClass is the least efficient one. On average, DDPMine outperforms HARMONY by an order of magnitude and outperforms PatClass by two orders of magnitude. This result is consistent with those in Figure 8.

Table IV shows the accuracy comparison between these three methods. On average, DDPMine has comparable accuracy with PatClass, and both outperform HARMONY by 9.8%. One would notice that the accuracy by PatClass and DDPMine is not identical although DDPMine simulates the mechanism of PatClass. The reason is that the set of discriminative patterns produced by these two methods are different. In PatClass, discriminative patterns are generated based on the complete set of training instances; whereas in DDPMine, discriminative patterns are generated based on the remaining training instances in each iteration.

VI. Related Work

Our study is related to associative classification [1], [2], [3], [4], [5] in which, a classifier is built based on high-confidence, high-support association rules. The association between frequent patterns and class labels is used for prediction.

Earlier studies on associative classification [1], [2], [3] mainly focus on mining high-support, high-confidence rules and build a rule-based classifier. Prediction is made based on the top-ranked rule or multiple rules.

A recent work on top-k rule mining [4] discovers top-k covering rule groups for high-dimensional gene expression

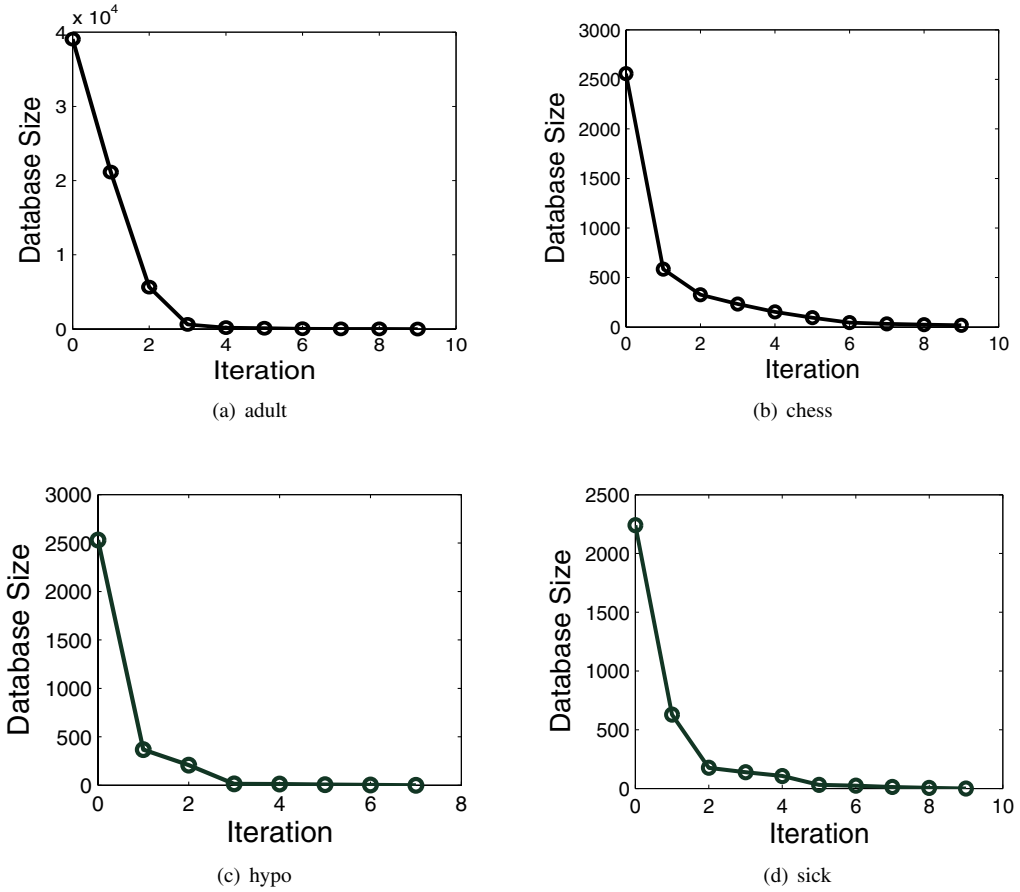


Fig. 10. Problem Size Reduction

TABLE III
RUNTIME COMPARISON

Dataset	HARMONY	PatClass	DDPMine
adult	60.78	1070.39	**8.75**
chess	37.09	113.98	**1.20**
crx	0.71	7.56	**0.57**
hypo	52.19	66.09	**0.66**
mushroom	**0.63**	34.42	0.83
sick	53.45	170.94	**1.70**
sonar	5.53	15.83	**0.83**
waveform	8.06	85.23	**4.34**
total	218.44	1564.44	**18.88**

TABLE IV
ACCURACY COMPARISON

Dataset	HARMONY	PatClass	DDPMine
adult	81.90	84.24	**84.82**
chess	43.00	91.68	**91.85**
crx	82.46	**85.06**	84.93
hypo	95.24	**99.24**	**99.24**
mushroom	99.94	99.97	**100.00**
sick	93.88	97.49	**98.36**
sonar	77.44	**90.86**	88.74
waveform	87.28	91.22	**91.83**
average	82.643	92.470	**92.471**

profiles. A classifier RCBT is constructed from the top-k covering rule groups and achieves very high accuracy.

HARMONY [5] is another rule-based classifier which directly mines classification rules. It uses an instance-centric rule-generation approach and assures for each training instance, that one of the highest-confidence rules covering the instance is included in the rule set. HARMONY is shown to be more efficient and scalable than previous rule-based classifiers.

[6] is a recently proposed innovative association rule-based classification method. Different from all the above studies, it is based on a *lazy* (non-eager) classification philosophy, in which the computation is performed on a demand-driven basis. This lazy classification method effectively reduces the number of rules produced by focusing on the test instance only. Experimental studies show that the lazy approach outperforms both the eager associative classification approach and decision tree-based classifiers in terms of accuracy. Based on the experimental results on common datasets, [11] is shown to achieve high accuracy than [6].

[11] is a newly proposed frequent pattern-based classification method. Highly discriminative frequent itemsets are selected to represent the data in a feature space, based on

which any learning algorithm can be used for model learning. This method first mines a set of frequent itemsets, then performs feature selection on the mining results to single out a compact set of highly discriminative itemsets. This method is shown to achieve very high accuracy.

VII. Conclusions

Frequent pattern-based classification methods have shown to be very effective at classifying categorical or high dimensional sparse datasets. However, many existing methods which mine a set of frequent itemsets or association rules encounter non-trivial computational bottleneck in the mining step, due to the explosive combination between the items. In addition, the explosive number of features poses great computational challenges for feature selection.

In this study, we proposed a direct discriminative pattern mining approach DDPMine which directly mines the discriminative patterns and integrates feature selection into the mining framework. A branch-and-bound search is imposed on the FP-growth mining process, which prunes the search space significantly. DDPMine works in an iterative fashion and reduces the problem size incrementally by eliminating training instances which are covered by the selected features. Experimental results show that DDPMine achieves orders of magnitude speedup over the two-step method without any downgrade of classification accuracy. DDPMine also outperforms the state-of-the-art associative classification methods in terms of both accuracy and efficiency.

VIII. Appendix

An upper bound of some discriminative measures was derived in [11] as a function of the itemset frequency. We use the widely used discriminative measure *information gain* [14] to show the bound.

Information gain is defined as

$$IG(C|X) = H(C) - H(C|X) \tag{10}$$

where $H(C)$ is the *entropy* and $H(C|X)$ is the *conditional entropy*.

Assume we have an itemset α whose absence or presence is represented by a random variable X, $X \in \{0, 1\}$. Assume $C = \{0, 1\}$. Let $P(x = 1) = \theta$ (θ is the frequency of the itemset α), $P(c = 1) = p$ and $P(c = 1|x = 1) = q$. The upper bound function shown below assumes $\theta \leq p$ since $\theta \geq p$ is a symmetric case. Then when $q = 0$ or $q = 1$, $IG(C|X)$ reaches the upper bound. When $q = 1$, the upper bound is

$$\begin{aligned} IG_{ub}(C|X) = &-p \log p - (1-p)\log(1-p) \\ &+(p-\theta)\log\frac{p-\theta}{1-\theta} + (1-p)\log\frac{1-p}{1-\theta} \end{aligned}$$

When $q = 0$, the upper bound is

$$\begin{aligned} IG_{ub}(C|X) = &-p \log p - (1-p)\log(1-p) \\ &+p\log\frac{p}{1-\theta} + (1-p-\theta)\log\frac{1-p-\theta}{1-\theta} \end{aligned}$$

In addition, we have the following conclusion:

$$\frac{\partial IG_{ub}(C|X)}{\partial \theta} > 0, \text{ if } \theta \leq p \tag{11}$$

This result shows that the information gain upper bound is a function of support θ. $IG_{ub}(C|X)$ is monotonically increasing with θ. When θ is small, $IG_{ub}(C|X)$ is small. Therefore, the information gain of low-frequency patterns is bounded by a small value.

Acknowledgment

The work was supported in part by the U.S. National Science Foundation (NSF) IIS-05-13678/06-42771 and BDI-05-15813. We thank Prof. Jianyong Wang at Tsinghua Univ. for giving us the HARMONY software.

References

[1] B. Liu, W. Hsu, and Y. Ma, "Integrating classification and association rule mining," in *Proc. of KDD*, 1998, pp. 80–86.

[2] W. Li, J. Han, and J. Pei, "CMAR: Accurate and efficient classification based on multiple class-association rules," in *Proc. of ICDM*, 2001, pp. 369–376.

[3] X. Yin and J. Han, "CPAR: Classification based on predictive association rules," in *Proc. of SDM*, 2003, pp. 331–335.

[4] G. Cong, K. Tan, A. Tung, and X. Xu, "Mining top-k covering rule groups for gene expression data," in *Proc. of SIGMOD*, 2005, pp. 670–681.

[5] J. Wang and G. Karypis, "HARMONY: Efficiently mining the best rules for classification," in *Proc. of SDM*, 2005, pp. 205–216.

[6] A. Veloso, W. M. Jr., and M. Zaki, "Lazy associative classification," in *Proc. of ICDM*, 2006, pp. 645–654.

[7] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," *Journal of Machine Learning Research*, vol. 2, pp. 419–444, 2002.

[8] C. Leslie, E. Eskin, and W. S. Noble, "The spectrum kernel: A string kernel for svm protein classification," in *Proc. of PSB*, 2002, pp. 564–575.

[9] T. Kudo, E. Maeda, and Y. Matsumoto, "An application of boosting to graph classification," in *Proc. of NIPS*, 2004, pp. 729–736.

[10] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis, "Frequent substructure-based approaches for classifying chemical compounds," *IEEE Trans. Knowl. and Data Eng.*, vol. 17, no. 8, pp. 1036–1050, 2005.

[11] H. Cheng, X. Yan, J. Han, and C. Hsu, "Discriminative frequent pattern analysis for effective classification," in *Proc. of ICDE*, 2007, pp. 716–725.

[12] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. of SIGMOD*, 2000, pp. 1–12.

[13] T. M. Mitchell, *Machine Learning*. McGraw Hill, 1997.

[14] J. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[15] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, 2nd ed. Wiley Interscience, 2000.

[16] G. Grahne and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets," in *ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, 2003.

[17] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.