On Mining Instance-Centric Classification Rules

Jianyong Wang and George Karypis

***Abstract*— Many studies have shown that rule-based classifiers perform well in classifying categorical and sparse high-dimensional databases. However, a fundamental limitation with many rule-based classifiers is that they find the rules by employing various heuristic methods to prune the search space, and select the rules based on the sequential database covering paradigm. As a result, the final set of rules that they use may not be the globally best rules for some instances in the training database. To make matters worse, these algorithms fail to fully exploit some more effective search space pruning methods in order to scale to large databases.**

In this paper we present a new classifier, HARMONY, which directly mines the final set of classification rules. HARMONY uses an instance-centric rule-generation approach and it can assure for each training instance, one of the highest-confidence rules covering this instance is included in the final rule set, which helps in improving the overall accuracy of the classifier. By introducing several novel search strategies and pruning methods into the rule discovery process, HARMONY also has high efficiency and good scalability. Our thorough performance study with some large text and categorical databases has shown that HARMONY outperforms many well-known classifiers in terms of both accuracy and computational efficiency, and scales well w.r.t. the database size.

***Index Terms*— Data mining, classification rule, instance-centric, classifier.**

I. Introduction

As one of the most fundamental data mining tasks, classification has been extensively studied and various types of classification algorithms have been proposed. Among which, one category is the rule-based classifiers [29], [30], [13], [33]. They build a model from the training database as a set of high-quality rules, which can be used to predict the class labels of unlabeled instances. Many studies have shown that rule-based classification algorithms perform very well in classifying both categorical databases [30], [28], [27], [33] and sparse high-dimensional databases such as those arising in the context of document classification [6], [5].

Some traditional rule-based algorithms like FOIL [30], RIPPER [13], and CPAR [33] discover a set of classification rules one-rule-at-a-time and employ a sequential covering methodology to eliminate from the training set the positive instances that are covered by each newly discovered rule. This *rule induction* process is done in a greedy fashion as it employs various heuristics (e.g., information gain) to determine how each rule would be extended. Due to this heuristic rule-induction process and the sequential covering framework, the final set of discovered rules are not guaranteed to be the best possible. For example, due to the removal of some training instances, the information gain is computed based on the incomplete information; thus, the variable (or literal) chosen by these algorithms to extend the current rule will be no longer the globally optimal one. Moreover, for multi-class problems, these algorithms need to be applied multiple times, each time mining the rules for one class. If the training database is large and contains many classes, the algorithms will be inefficient.

Since the introduction of association rule mining [2], many association-based classifiers have been proposed [19], [24], [7], [4], [8], [25], [16], [5], [34], [15]. Some typical examples like CBA [28] and CMAR [27] adopt efficient association rule mining algorithms (e.g., Apriori [3] and FP-growth [22]) to first mine a large number of high-confidence rules satisfying a user-specified minimum support and confidence thresholds and then use various sequential-covering-based schemes to select from them a set of high-quality rules to be used for classification. Since these schemes defer the selection step only after a large intermediate set of high-confidence rules have been identified, they tend to achieve somewhat better accuracy than the heuristic rule induction schemes [33]. However, the drawback of these approaches is that the number of initial rules is usually extremely large, significantly increasing the rule discovery and selection time.

In this paper we propose a new classification algorithm, HARMONY[1], which can overcome the problems of both the rule-induction-based and the association-rule-based algorithms. HARMONY directly mines for each training instance one of the highest confidence classification rules that it supports and satisfies a user-specified minimum support constraint, and builds the classification model from the union of these rules over the entire set of instances. Thus HARMONY employs an *instance-centric* rule generation framework and is guaranteed to find and include the best possible rule for each training instance. Moreover, since each training instance usually supports many of the discovered rules, the overall classifier can better generalize to new instances and thus achieve better classification performance.

To achieve high computational efficiency, HARMONY mines the classification rules for all the classes simultaneously and directly mines the final set of classification rules by pushing deeply some effective pruning methods into the projection-based frequent itemset mining framework. All these pruning methods preserve the completeness of the resulting rule-set in the sense that they only remove from consideration rules that are guaranteed not to be of high quality. We have performed numerous performance studies with various databases and shown that HARMONY can achieve better accuracy while

Jianyong Wang is with the Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China. E-mail: jianyong@tsinghua.edu.cn.

George Karypis is with the Department of Computer Science and Engineering, Digital Technology Center and Army HPC Research Center, University of Minnesota, Minneapolis, MN 55455, U.S.A. Email: karypis@cs.umn.edu.

[1] HARMONY stands for **H**ighest confidence cl**A**ssification **R**ule **M**ining f**O**r i**N**stance-centric classif**Y**ing.

maintaining high efficiency.

The rest of the paper is organized as follows. Section II introduces some basic definitions and notations. Section III describes the problem formulation. Section IV introduces some related work. Section V discusses in detail the HARMONY algorithm and some extensions to the algorithm. The performance study is presented in Section VI. Finally, the paper concludes with Section VII.

II. Notations and Definitions

A *training database* $TrDB$ is a set of training instances, where each training instance, denoted as a triple $\langle tid, X, cid\rangle$, contains a set of items (i.e., X) and is associated with a unique training instance identifier tid, and a class identifier $cid \in \{c_1, c_2, ..., c_k\}$ (A class identifier is also called a class label, and we assume there are totally k distinct class labels in $TrDB$). Table I illustrates an example training database, which contains totally eight instances and two classes. Let $I=\{i_1, i_2, \ldots, i_n\}$ be the complete set of distinct items appearing in $TrDB$. An *itemset* Y is a non-empty subset of I and is called an *l-itemset* if it contains l items. An itemset $\{x_1, \ldots, x_l\}$ is also denoted by $x_1 \cdots x_l$. A training instance $\langle tid, X, cid\rangle$ is said to *contain* itemset Y if $Y \subseteq X$. The number of instances in $TrDB$ containing itemset Y is called the (absolute) *support* of itemset Y, denoted by sup_Y. The number of instances containing itemset Y and associated with a class label c_i (where $i \in \{1, 2, ..., k\}$) is called the support of $Y \cup \{c_i\}$, denoted by $sup_Y^{c_i}$. A classification rule has the form: '$Y \rightarrow c_i : sup_Y^{c_i}, conf_Y^{c_i}$', where Y is called the body, c_i the head, $sup_Y^{c_i}$ the support, and $conf_Y^{c_i} = \frac{sup_Y^{c_i}}{sup_Y}$ the confidence of the rule, respectively. In addition, we use $|TrDB|$ to denote the number of instances in database $TrDB$, and for brevity, we sometimes use the instance identifier tid to denote an instance $\langle tid, X, cid\rangle$.

TABLE I
An example training database $TrDB$.

Instance identifier	Set of items	Class identifier
01	a, c, e, g	1
02	b, d, e, f	0
03	d, e, f	0
04	a, b, c, e	1
05	a, c, e	1
06	b, d, e	0
07	a, b, e	1
08	a, b, d, e	0

Given a minimum support threshold, *min_sup*, an itemset Y is *frequent* if $sup_Y \geq$*min_sup*. A frequent itemset Y supported by any training instance $\langle t_j, X_j, c_i\rangle$ ($1 \leq j \leq |TrDB|$ and $1 \leq i \leq k$) is also called a frequent covering itemset of instance t_j, and '$Y \rightarrow c_i : sup_Y^{c_i}, conf_Y^{c_i}$' is called a frequent covering rule of instance t_j. Among the frequent covering rules of any instance t_j, those with the highest confidence are called the *Highest Confidence Covering Rules* w.r.t. instance t_j. We denote a Highest Confidence Covering Rule w.r.t. instance t_j by $HCCR_{t_j}$, and use $HCCR_{t_j}^{sup}$ and $HCCR_{t_j}^{conf}$ to denote its support and confidence.

III. Problem Definition

The goal of this paper is to design an accurate and efficient rule-based classifier with good scalability, which should be able to overcome the problems of both the traditional rule-based and the recently proposed association-based classifiers. As mentioned in Section I, instead of using the sequential database covering to select the rules, our solution mines a set of high quality rules in an instance-centric manner and can assure that at least one of the highest confidence frequent covering rules (if there is any) w.r.t. any training instance is included in the final result set of classification rules.

Specifically, given a training database $TrDB$ and a minimum support threshold *min_sup*, the problem of this study is to find one of the highest confidence frequent covering rules for each of the training instances in $TrDB$, and build a classifier from these classification rules. Note the input training database must be in the form that is consistent with the corresponding definition in Section II, otherwise, the training database should be first converted to that form. For example, a numerical database should be first discretized into a categorical one in order to use HARMONY to build the model. In addition, although this study mainly focuses on mining any one of the highest confidence frequent covering rules for each training instance, it is straightforward to revise HARMONY to mine the complete set of the highest confidence frequent covering rules or K highest confidence frequent covering rules for each training instance as discussed in Section V-D.2.

IV. Related Work

There are two classes of algorithms that are directly related to this work. One is the traditional rule-induction-based methods and the other is the recently proposed association-rule-based methods. Both of these classes share the same idea of trying to find a set of classification rules to build the model. The rule-induction-based classifiers like C4.5 [29], FOIL [30], RIPPER [13], and CPAR [33] use various heuristics such as information gain (including Foil gain) and gini index to identify the best variable (or literal) by which to grow the current rule, and many of them follow a sequential database covering paradigm to speed up rule induction. The association-based classifiers adopt another approach to find the set of classification rules. They first use some efficient association rule mining algorithms to discover the complete (or a large intermediate) set of association rules, from which the final set of classification rules can be chosen based on various types of sequential database covering techniques. Some typical examples of association-based (or *Emerging Pattern*-based) methods include CBA [28], CAEP [17], CMAR [27], ARC-BC [5], and DeEPs [26].

In contrast to the rule-induction-based algorithms, HARMONY does not apply any heuristic pruning methods and the sequential database covering approach. Instead, it follows an instance-centric framework and mines the covering rules with the highest confidence for each instance, which can achieve better accuracy. At the same time, by maintaining the currently best rules for each training instance and pushing deeply several effective pruning methods into the projection-based frequent

itemset mining framework [22], [1], [20], HARMONY directly mines the final set of classification rules, which avoids the time consuming rule generation and selection process used in several association-based classifiers [28], [27], [5].

The idea of directly mining a set of high confidence rules is similar to those in [7], [15]. The author of [7] investigated a brute-force technique for mining the set of high-confidence classification rules, and proposed several effective pruning strategies to control the combinatorial explosion in the number of rule candidates. The FARMER algorithm [15] finds the interesting rule groups for microarray databases. It mines the rules in a row enumeration space, and fully exploits some pruning methods to prune the search space based on the user-specified constraints like minimum support, confidence, and chi-square. Unlike [7], [15], HARMONY does not need the user to specify the minimum confidence and/or chi-square. Instead, it mines for each training instance one of the highest confidence frequent rules that it covers. In addition, by maintaining the currently best classification rules for each instance, HARMONY is able to incorporate some new pruning methods under the unpromising item (or conditional database) pruning framework, which has been proven very effective in pushing deeply the length-decreasing support constraint or tough block constraints into closed itemset mining [31], [20]. In addition, recently we noticed that a similar approach [14] to this research was independently proposed at the same time frame, and showed its high accuracy in classifying gene expression data.

V. HARMONY: An Instance-Centric Classifier

In this section, we will describe in detail the HARMONY algorithm. We first elaborate on how to adapt the traditional projection-based frequent itemset mining framework to efficiently enumerate the classification rules, then we focus on how to push deeply some effective pruning methods into the rule enumeration framework and give the whole algorithm. Finally we will discuss several important extensions to HARMONY.

A. Classification Rule Enumeration

The projection-based itemset enumeration framework has been widely used in many frequent itemset mining algorithms [22], [1], [20], and will be used by HARMONY as the basis in enumerating the classification rules. Given a training database $TrDB$ and a minimum support *min_sup*, HARMONY first computes the frequent items by scanning $TrDB$ once, and sorts them to get a list of frequent items (denoted by *f_list*) according to a certain ordering scheme. Assume the *min_sup* is 3 and the lexicographical ordering is the default ordering scheme, the *f_list* computed from Table I is $\{a, b, c, d, e\}$. HARMONY applies the *divide-and-conquer* method plus the *depth-first search strategy*. In our example, HARMONY first mines the rules whose body contains item 'a', then mines the rules whose body contains 'b' but no 'a', ..., and finally mines the rules whose body contains only 'e'. In mining the rules with item 'a', item 'a' is treated as the current prefix, and its conditional database (denoted by $TrDB|_a$) is built and the *divide-and-conquer* method is applied recursively with the depth-first search strategy. To build conditional database $TrDB|_a$, HARMONY first identifies the instances in $TrDB$ containing 'a' and removes the infrequent items, then sorts the left items in each instance according to the *f_list* order, finally $TrDB|_a$ is built as $\{\langle 01, ce, 1\rangle, \langle 04, bce, 1\rangle, \langle 05, ce, 1\rangle, \langle 07, be, 1\rangle, \langle 08, be, 0\rangle\}$ (infrequent items 'd' and 'g' are removed). Following the *divide-and-conquer* method, HARMONY first mines the rules with prefix 'ab', then mines rules with prefix 'ac' but no 'b', and finally mines rules with prefix 'ae' but no 'b' nor 'c'.

During the mining process, when HARMONY gets a new prefix, it will generate a set of classification rules w.r.t. the training instances covered by the prefix. For each training instance, it always maintains one of its currently highest confidence rules mined so far. Assume the current prefix P is 'a' (i.e., P='a'). As shown in the above example, P covers five instances with $tids$ 01, 04, 05, 07, and 08. HARMONY computes the covering rules according to the class distribution w.r.t. the prefix P. In this example, sup_P=5, sup_P^0=1, sup_P^1=4, and HARMONY generates two classification rules:

Rule 1: $a \rightarrow 0 : 1, \frac{1}{5}$
Rule 2: $a \rightarrow 1 : 4, \frac{4}{5}$

Rule 1 covers the instance with tid 08, while Rule 2 covers the instances with $tids$ 01, 04, 05 and 07. Up to this point, we have $HCCR_{01}$ = $HCCR_{04}$ = $HCCR_{05}$ = $HCCR_{07}$ = Rule 2, and $HCCR_{08}$ = Rule 1.

1) Ordering of the Local Items: In the above rule enumeration process, we used the lexicographical ordering as an illustration to sort the set of local frequent items in order to get the *f_list*. Many frequent itemset mining algorithms either adopt item support descending order [22] or support ascending order [20] as the ordering scheme. However, because we are interested in the highest confidence rules w.r.t. the training instances, both the support descending order and ascending order may not be the most efficient and effective ways. As a result, we propose the following three new ordering schemes as the alternatives.

Let the current prefix be P, its support be sup_P, the support and confidence of the classification rule w.r.t. prefix P and class label c_i, '$P \rightarrow c_i$', be $sup_P^{c_i}$ and $conf_P^{c_i}$, respectively, the set of local frequent items be $\{x_1, x_2, ..., x_m\}$, the number of prefix P's conditional instances containing item x_j $(1 \leq j \leq m)$ and associated with class label c_i $(1 \leq i \leq k)$ be $sup_{P\cup\{x_j\}}^{c_i}$, and the support of $P \cup \{x_j\}$ be $sup_{P\cup\{x_j\}}$ $=\sum_{i=1}^{k} sup_{P\cup\{x_j\}}^{c_i}$.

Maximum confidence descending order. Given a local item x_j $(1 \leq j \leq m)$ w.r.t. P, we can compute k rules with body $P \cup \{x_j\}$, among which, the i-th rule with rule head c_i is:

$$P \cup \{x_j\} \rightarrow c_i \ : \ sup_{P\cup\{x_j\}}^{c_i}, \ \frac{sup_{P\cup\{x_j\}}^{c_i}}{sup_{P\cup\{x_j\}}}$$

The highest confidence among the k rules with body $P \cup \{x_j\}$ is called the maximum confidence of local item x_j, and is defined as the following:

$$\frac{\max_{\ \forall i, 1\leq i\leq k} \ sup_{P\cup\{x_j\}}^{c_i}}{sup_{P\cup\{x_j\}}} \tag{1}$$

To mine the highest confidence covering rules as quickly as possible, a good heuristic is to sort the local frequent items in their maximum confidence descending order.

Entropy ascending order. The widely used entropy to some extent measures the purity of a cluster of instances. If the entropy of the set of instances containing $P \cup \{x_j\}$ $(1 \leq j \leq m)$ is small, it is highly possible to generate some high confidence rules with body $P \cup \{x_j\}$. Thus another good ordering heuristic is to rank the set of local frequent items in their entropy ascending order, and the entropy w.r.t. item x_j is defined as follows:

$$-\frac{1}{\log k}\sum_{i=1}^{k}\left(\frac{sup_{P\cup\{x_j\}}^{c_i}}{sup_{P\cup\{x_j\}}}\right)\log\left(\frac{sup_{P\cup\{x_j\}}^{c_i}}{sup_{P\cup\{x_j\}}}\right) \quad (2)$$

Correlation coefficient ascending order. Both the maximum confidence descending order and entropy ascending order do not consider the class distribution of the conditional database w.r.t. prefix P, which may cause some problems in some cases. Let us see an example. Assume the number of class labels k=2, $sup_P^{c_1} = 12$, and $sup_P^{c_2} = 6$, then we can get two rules with body P as follows:

Rule 3: $P \rightarrow c_1 : 12, \frac{12}{18}$
Rule 4: $P \rightarrow c_2 : 6, \frac{6}{18}$

Suppose there are two local items, x_1 and x_2, and $sup_{P\cup\{x_1\}}^{c_1}$=2, $sup_{P\cup\{x_1\}}^{c_2}$=1, $sup_{P\cup\{x_2\}}^{c_1}$=1, and $sup_{P\cup\{x_2\}}^{c_2}$=2. According to Equation 1 and Equation 2, the maximum confidence and entropy w.r.t. item x_1 are equal to the corresponding maximum confidence and entropy w.r.t. x_2. Thus we cannot determine which one of x_1 and x_2 should be ranked higher. However, because the conditional database $TrDB|_{P\cup\{x_1\}}$ has the same class distribution as conditional database $TrDB|_P$, we cannot generate rules with body $P \cup \{x_1\}$ and a confidence higher than those with body P (i.e., Rule 3 and Rule 4). The two rules with body $P \cup \{x_1\}$ are shown as the following.

Rule 5: $P \cup \{x_1\} \rightarrow c_1 : 2, \frac{2}{3}$
Rule 6: $P \cup \{x_1\} \rightarrow c_2 : 1, \frac{1}{3}$

If we examine the rules generated from prefix itemset $P \cup \{x_2\}$ as shown in Rule 7 and Rule 8, we can see Rule 8 has higher confidence than Rule 4, and can be used to replace Rule 4 for the instances covered by Rule 8. In this case , item x_2 should be ranked before item x_1.

Rule 7: $P \cup \{x_2\} \rightarrow c_1 : 1, \frac{1}{3}$
Rule 8: $P \cup \{x_2\} \rightarrow c_2 : 2, \frac{2}{3}$

This example suggests that the more similar the class distribution between conditional databases $TrDB|_P$ and $TrDB|_{P\cup\{x_j\}}$ $(1 \leq j \leq m)$, the lower is the possibility to generate higher confidence rules from $TrDB|_{P\cup\{x_j\}}$. Because the correlation coefficient is a good metric in measuring the similarity between two vectors (the larger the coefficient, the more similar the two vectors), it can be used to rank the local items. In HARMONY, the correlation coefficient ascending order is adopted to sort the local items.

Let $\overline{sup_P}$ be $\frac{1}{k}\sum_{i=1}^{k} sup_P^{c_i}$, $\overline{sup_{P\cup\{x_j\}}}$ be $\frac{1}{k}\sum_{i=1}^{k} sup_{P\cup\{x_j\}}^{c_i}$, σ_P be $\sqrt{\frac{1}{k}\sum_{i=1}^{k}(sup_P^{c_i})^2 - \overline{sup_P}^2}$, $\sigma_{P\cup\{x_j\}}$ be $\sqrt{\frac{1}{k}\sum_{i=1}^{k}(sup_{P\cup\{x_j\}}^{c_i})^2 - \overline{sup_{P\cup\{x_j\}}}^2}$, the correlation coefficient between prefix P and $P \cup \{x_j\}$ $(1 \leq j \leq m)$ is defined as follows.

$$\frac{\frac{1}{k}\sum_{i=1}^{k}(sup_P^{c_i} \times sup_{P\cup\{x_j\}}^{c_i} - \overline{sup_P} \times \overline{sup_{P\cup\{x_j\}}})}{\sigma_P \times \sigma_{P\cup\{x_j\}}} \quad (3)$$

B. Search Space Pruning

Unlike the association-based algorithms, HARMONY directly mines the final set of classification rules. By maintaining the current highest confidence among the covering rules for each training instance during the mining process, some effective pruning methods can be proposed to improve the algorithm efficiency.

1) Support Equivalence Item Elimination: Given the current prefix P, among its set of local frequent items $\{x_1, x_2, ..., x_m\}$, some may have the same support as P. We call them support equivalence items and can be safely pruned according to the following Lemma 1.

Lemma 1: (***Support equivalence item pruning***) Any local item x_j w.r.t. prefix P can be safely pruned if it satisfies $sup_{P\cup\{x_j\}} = sup_P$.

Proof. Because $sup_{P\cup\{x_j\}} = sup_P$ holds, $TrDB|_P$ and $TrDB|_{P\cup\{x_j\}}$ contain the same set of conditional instances; thus, their class distributions are also the same and the following equation must hold:

$\forall i, 1 \leq i \leq k, sup_{P\cup\{x_j\}}^{c_i} = sup_P^{c_i}$

Given any itemset, Y, which can be used to extend P (Y can be empty), can also be used to extend $P \cup \{x_j\}$, and the following must hold:

$\forall i, 1 \leq i \leq k, sup_{P\cup\{x_j\}\cup Y}^{c_i} = sup_{P\cup Y}^{c_i}$

We can further have the following equation:

$\forall i, 1 \leq i \leq k, \frac{sup_{P\cup\{x_j\}\cup Y}^{c_i}}{sup_{P\cup\{x_j\}\cup Y}} = \frac{sup_{P\cup Y}^{c_i}}{sup_{P\cup Y}}$

This means the confidence of the rule '$P\cup\{x_j\}\cup Y \rightarrow c_i$' is equal to the confidence of the rule '$P\cup Y \rightarrow c_i$', and we cannot generate higher confidence rules from prefix $P \cup \{x_j\} \cup Y$ in comparison with the rules with body $P \cup Y$. Thus, item x_j can be safely pruned. ■

Note $P \cup Y$ is a subset of $P \cup \{x_j\} \cup Y$, by pruning item x_j, we prefer the more generic classification rules. A similar strategy was adopted in [7], [15].

2) Unpromising Item Elimination: Given the current prefix P, any one of its local frequent items, x_j $(1 \leq j \leq m)$, any itemset Y that can be used to extend $P \cup \{x_j\}$ (where Y can be empty and $P \cup \{x_j\} \cup Y$ is frequent), and any class label c_i $(1 \leq i \leq k)$, the following equation must hold:

$$conf_{P\cup\{x_j\}\cup Y}^{c_i} = \frac{sup_{P\cup\{x_j\}\cup Y}^{c_i}}{sup_{P\cup\{x_j\}\cup Y}} \leq \frac{sup_{P\cup\{x_j\}\cup Y}^{c_i}}{min\_sup} \leq \frac{sup_{P\cup\{x_j\}}^{c_i}}{min\_sup}$$

Because $conf_{P\cup\{x_j\}\cup Y}^{c_i} \leq 1$ also holds, we have the following equation:

$$conf^{c_i}_{P\cup\{x_j\}\cup Y} \leq \min\{1, \frac{sup^{c_i}_{P\cup\{x_j\}}}{min\_sup}\} \quad (4)$$

Lemma 2: (***Unpromising item pruning***) For any conditional instance $\langle t_l, X_l, c_i\rangle \in TrDB|_{P\cup\{x_j\}}$ ($\forall l,\ 1 \leq l \leq |TrDB|_{P\cup\{x_j\}}|$, and $1 \leq i \leq k$), if the following always holds, item x_j is called an unpromising item and can be safely pruned.

$$HCCR^{conf}_{t_l} \geq \min\{1, \frac{sup^{c_i}_{P\cup\{x_j\}}}{min\_sup}\} \quad (5)$$

Proof. By combining Equation 4 and Equation 5 we get that for any itemset Y (Y can be empty) the following must hold:

$$conf^{c_i}_{P\cup\{x_j\}\cup Y} \leq HCCR^{conf}_{t_l}$$

This means that any rule mined by growing prefix $P \cup \{x_j\}$ will have a confidence that is no greater than the current highest confidence covering rules (with the same rule head) of any conditional instance in $TrDB|_{P\cup\{x_j\}}$; thus, item x_j can be safely pruned. ■

3) Unpromising Conditional Database Elimination: Given the current prefix P, any itemset Y (where Y can be empty and $P \cup Y$ is frequent), any class label c_i ($1 \leq i \leq k$), the confidence of rule '$P \cup Y \rightarrow c_i$', $conf^{c_i}_{P\cup Y}$, must satisfy the following equation:

$$conf^{c_i}_{P\cup Y} = \frac{sup^{c_i}_{P\cup Y}}{sup_{P\cup Y}} \leq \frac{sup^{c_i}_{P\cup Y}}{min\_sup} \leq \frac{sup^{c_i}_{P}}{min\_sup}$$

In addition, because $conf^{c_i}_{P\cup Y} \leq 1$ also holds, we have the following equation:

$$conf^{c_i}_{P\cup Y} \leq \min\{1, \frac{sup^{c_i}_{P}}{min\_sup}\} \quad (6)$$

Lemma 3: (***Unpromising conditional database pruning***) For any conditional instance $\langle t_l, X_l, c_i\rangle \in TrDB|_P$ ($\forall l, 1 \leq l \leq |TrDB|_P|$, and $1 \leq i \leq k$), if the following always holds, the conditional database $TrDB|_P$ can be safely pruned.

$$HCCR^{conf}_{t_l} \geq \min\{1, \frac{sup^{c_i}_{P}}{min\_sup}\} \quad (7)$$

Proof. By combining Equation 6 and Equation 7 we can get that for any itemset Y (Y can be empty) and $\forall l, 1 \leq l \leq |TrDB|_P|$, $\langle t_l, X_l, c_i\rangle \in TrDB|_P$ ($1 \leq i \leq k$), the following must hold:

$$conf^{c_i}_{P\cup Y} \leq HCCR^{conf}_{t_l}$$

This means that any rule mined by growing prefix P will have a confidence that is no greater than the current highest confidence rules (with the same rule head) of any conditional instance in $TrDB|_P$; thus, the whole conditional database $TrDB|_P$ can be safely pruned. ■

C. The algorithm

After we described how to enumerate the classification rules, and how to design the local item ordering scheme and some effective search space pruning methods in order to accelerate the mining of the highest confidence covering rules in terms of each training instance, we introduce the integrated HARMONY algorithm in this section.

The HARMONY algorithm is shown in ALGORITHM 1. It consists of three sub-algorithms: RULEMINER() takes as input the training database $TrDB$ and the minimum support *min_sup*, and outputs the set of highest confidence covering classification rules, $HCCR$; BUILDMODEL() takes $HCCR$ as input and outputs a classification model, CM; NEWINSTANCECLASSIFICATION() classifies a new test instance ti using the model CM.

ALGORITHM 1: HARMONY(*TrDB, min_sup, ti*)

INPUT: (1) *TrDB*: *a training database*, (2) *min sup*: *a minimum support threshold*, and (3) *ti*: *a new test instance.*
OUTPUT: (1) *HCCR*: *the set of the highest confidence frequent covering rules w.r.t. each instance in TrDB*, (2) *CM*: a classification model, (3) *PCL*: the predicted class label(s) w.r.t. test instance *ti*.

01. *HCCR* ← RULEMINER(*TrDB, min_sup*);
02. *CM* ← BUILDMODEL(*HCCR*);
03. *PCL* ← NEWINSTANCECLASSIFICATION(*CM, ti*).

1) Classification Rule Generation: In Section V-A and Section V-B we introduced how to efficiently enumerate the classification rules under the *divide-and-conquer* and *depth-first search* paradigm, and proposed several pruning methods to speed up the enumeration of the highest confidence covering rules. By integrating the pruning methods with the rule enumeration, we get the classification rule generation algorithm, as shown in the RULEMINER() algorithm.

The RULEMINER() algorithm first initializes the highest confidence classification rules w.r.t. each training instance to empty (lines 01-02), then enumerates the classification rules by calling subroutine *ruleminer*(∅, *TrDB*) (line 03). Subroutine *ruleminer*() takes as input a prefix itemset pi and its corresponding conditional database cdb. For each conditional instance, it checks if a classification rule with higher confidence can be computed from the current prefix pi, if so, it replaces the corresponding instance's current highest confidence rule with the new rule (lines 04-07). It then finds the frequent local items by scanning cdb (line 08), prunes invalid items based on the *support equivalence item pruning* method and the *unpromising item pruning* method (lines 09-10). If the set of valid local items is empty or the whole conditional database cdb can be pruned based on the *unpromising conditional database* pruning method, it returns directly (lines 11-13). Otherwise, it sorts the left frequent local items according to the correlation coefficient ascending order (line 14), and grows the current prefix (line 16), builds the conditional database for the new prefix (line 17), and recursively calls itself to mine the highest confidence rules from the new prefix (line 18).

ALGORITHM 1.1: RULEMINER(*TrDB, min_sup*)

INPUT: (1) *TrDB*: *a training database*, and (2) *min sup*: *a minimum support threshold.*
OUTPUT: (1) *HCCR*: *the set of the highest confidence frequent covering rules w.r.t. each instance in TrDB.*

01. *for all* $t_i \in TrDB$

02. $HCCR_{t_i} \leftarrow \emptyset$;
03. *call* **ruleminer**($\emptyset$, *TrDB*).

SUBROUTINE 1.1 : **ruleminer**(*pi*, *cdb*)

INPUT: (1) *pi*: *a prefix itemset*, and (2) *cdb*: *the conditional database w.r.t. prefix pi.*
04. *if*($pi \neq \emptyset$)
05. *for all* $\langle t_l, X_l, c_j \rangle \in cdb$
06. *if*($HCCR_{t_l}^{conf} < \frac{sup_{pi}^{c_j}}{sup_{pi}}$)
07. $HCCR_{t_l} \leftarrow$ *rule* '$pi \rightarrow c_j$';
08. $I \leftarrow$ *find_frequent_items(cdb,min_sup);*
09. $S \leftarrow$ *support_equivalence_item_pruning(I);* $I \leftarrow I - S$;
10. $S \leftarrow$ *unpromising_item_pruning(I, cdb);* $I \leftarrow I - S$;
11. *if*($I \neq \emptyset$)
12. *if(unpromising_conditional_database_pruning(I,pi,cdb))*
13. *return;*
14. *correlation_coefficient_asscending_ordering(I);*
15. *for all* $x \in I$ *do*
16. $pi' \leftarrow pi \cup \{x\}$;
17. $cdb' \leftarrow$ *build_cond_database*(pi', cdb);
18. *call* **ruleminer**(pi', cdb');

2) Building the Classification Model: After the set of highest confidence covering rules have been mined, it will be straightforward to build the classification model. HARMONY first groups the set of highest confidence covering rules into k groups according to their rule heads (i.e., class labels), where k is the total number of distinct class labels in the training database. Within the same group of rules, HARMONY sorts the rules in their confidence descending order, and for the rules with the same confidence, sorts them in support descending order. In this way, HARMONY prefers the rules with higher confidence, and the rules with higher support if the confidence is the same. The BUILDMODEL algorithm is shown in ALGORITHM 1.2.

ALGORITHM 1.2: BUILDMODEL(*HCCR*)

INPUT: (1) *HCCR*: *the set of highest confidence covering rules.*
OUTPUT: (1) *CM*: *the classification model (i.e., k groups of ranked rules).*
01. *Cluster_rules_into_k_groups(HCCR);*//according to class label
02. *for each group of rules*
03. *Sort_rules();*//in confidence and support descending order

ALGORITHM 1.3: NEWINSTANCECLASSIFICATION(*CM*, *ti*)

INPUT: (1) *CM*: *the classification model*, (2) *ti*: *a test instance.*
OUTPUT: (1) *PCL*: *a predicted class label (or a set of class labels).*
01. *for j=1 to k* //CM_j: the j-th group of rules in CM
//SCR_j: the score for *ti* computed from CM_j
02. $SCR_j \leftarrow$*ComputeScore*(CM_j, *ti*);
03. *PCL* $\leftarrow$ *PredictClassLabel(SCR).*

3) New Instance Classification: After the classification model, *CM*, has been built, it can be used to classify a new test instance, *ti*, using the NEWINSTANCECLASSIFICATION algorithm shown in ALGORITHM 1.3. HARMONY first computes a score w.r.t. ti for each group of rules in CM (lines 01-02), and predicts for ti a class label or a set of class labels if the underlying classification is a multi-class multi-label problem (i.e., each instance can be associated with several class labels).

Scoring function. In HARMONY, the score for a certain group of rules is defined in three different ways. The first scoring function is called *HIGHEST*, which computes the score as the highest confidence among the covering rules w.r.t. test instance ti (by a 'covering rule', we mean its rule body is a subset of ti). The second method is based on the *ALL* function. It is the default scoring function in HARMONY and computes the score as the sum of the confidences of all the covering rules w.r.t. ti. The third function is called *TOP-K*, where K is a user-specified parameter. It computes the score for a group of rules as the sum of the top K highest confidences of the covering rules w.r.t. ti. The *HIGHEST* and *ALL* functions can be thought of as two special cases of the *TOP-K* function when K is set at 1 and $+\infty$. For a multi-class single-label classification problem, HARMONY simply chooses the class label with the highest score as the predicted class label. However, for a multi-class multi-label classification problem, the prediction is a little complicated.

Multi-class multi-label classification. In [5], the *dominant factor*-based method was proposed to predict the class labels for a multi-class multi-label classification problem and works as follows. Given a user-specified *dominant factor* γ, let the class label with the highest score be c_{max} and the corresponding highest score w.r.t. test instance ti be $SCORE_{ti}^{c_{max}}$, then any class label whose corresponding score is no smaller than $SCORE_{ti}^{c_{max}} \times \gamma$ is a predicted class label for ti. This method has been verified to be effective in practice [5]. However, in many imbalanced classification problems, the average confidence of each group of classification rules may be quite different from each other, this uniform *dominant factor*-based method will not work well. A large *dominant factor* may lead to low recalls (i.e., the percentage of the total test instances for the given class label that are correctly classified) for the classes with low average rule confidences, while a small *dominant factor* can lead to low precisions (i.e., the percentage of predicted instances for the given class label that are correctly classified) for the classes with high average rule confidences. To overcome this problem, HARMONY adopts a *weighted dominant factor*-based method. Let the average confidence of the group of classification rules w.r.t. class label c_k be $conf_{c_k}^{avg}$, the score w.r.t. instance ti and class label c_k be $SCORE_{ti}^{c_k}$. Instance ti is predicted to belong to class c_k if it satisfies the equation:

$$SCORE_{ti}^{c_k} \geq SCORE_{ti}^{c_{max}} \times \gamma \times (\frac{conf_{c_k}^{avg}}{conf_{c_{max}}^{avg}})^{\delta}$$

Here, δ ($\delta \geq 0$) is called the *score differentia factor* and the larger the δ, the more the difference of the *weighted dominant factors* (i.e., $\gamma \times (\frac{conf_{c_k}^{avg}}{conf_{c_{max}}^{avg}})^{\delta}$) among different class labels. It is set to 1 by default in HARMONY.

D. Extensions to HARMONY

1) Using Class-Specific Support Threshold: The classification rule enumeration algorithm described in Section V-C.1 assumes a uniform minimum support as an input, which may cause some problems for the imbalanced training databases. By an imbalanced training database, we mean the class distribution is not balanced, that is, some classes may contain a much larger number of instances than the other classes. If a large minimum support is used as input, the algorithm will encounter difficulties in mining high confidence rules for the small classes, while a small minimum support will

lead to the overfitting problem for some large classes. This intuition suggests we should use different minimum supports for different size classes.

HARMONY provides two ways in specifying class-specific minimum supports. The first way allows the user to directly specify a minimum support for each class (in the following we will use min_sup_i to denote the minimum support of the i-th class). However, when there exist a lot of classes in the database, to specify a proper minimum support for each class is not an easy task. As a result, in the second option, HARMONY requires the user to provide a minimum support, ***min_sup***, which corresponds to the minimum support of the smallest class, and it will automatically compute a minimum support for each class from ***min_sup*** and the class distribution. Let the number of training instances w.r.t. class c_i be $|c_i|$, then min_sup_i is computed as follows:

$$min\_sup_i = \textbf{\textit{min\_sup}} \times (\frac{|c_i|}{\min_{\forall j, 1 \leq j \leq k} |c_j|})^{\xi}$$

Here, ξ ($\xi \geq 0$) is called the *support differentia factor*. In HARMONY, ξ is set to 0 by default, which can be used to compute a uniform minimum support for all the classes.

By using class-specific minimum supports, Lemma 2 and Lemma 3 still applies, but we need to replace ***min_sup*** with min_sup_i in Equation 5 and Equation 7. For example, Equation 5 should be changed to the following form:

$$HCCR_{t_l}^{conf} \geq \min\{1, \frac{sup_{P \cup \{x_j\}}^{c_i}}{min\_sup_i}\}$$

In addition, to make the algorithm work, we also need to require the line 06 of SUBROUTINE 1.1 satisfy $sup_{pi} \geq min\_sup_j$.

2) Mining K-Rules for Each Instance: A training instance may support multiple highest-confidence classification rules, but the above classification rule enumeration algorithm described in Section V-C.1 only reports the first discovered one. Usually this arrangement can assure the set of final rules is large enough to build an accurate classifier in the case that the training database contains a large number of training instances. However, the set of final rules generated in this way may not be sufficient if the database is small. To overcome this problem, HARMONY provides an option to mine K highest-confidence rules w.r.t. a training instance if it supports multiple highest-confidence rules, where K is a user-specified parameter.

In order to mine K-rules for each instance, Equation 5 needs to be changed to the following form:

$$(HCCR_{t_l}^{conf} > \min\{1, \frac{sup_{P \cup \{x_j\}}^{c_i}}{min\_sup}\}) \vee$$

$$(HCCR_{t_l}^{conf} = \min\{1, \frac{sup_{P \cup \{x_j\}}^{c_i}}{min\_sup}\}) \wedge (n >= K)$$

Where n is the number of highest confidence classification rules discovered so far w.r.t. t_l.

Similarly, Equation 7 should have the following form:

$$(HCCR_{t_l}^{conf} > \min\{1, \frac{sup_{P}^{c_i}}{\textbf{\textit{min\_sup}}}\}) \vee$$

$$(HCCR_{t_l}^{conf} = \min\{1, \frac{sup_{P}^{c_i}}{\textbf{\textit{min\_sup}}}\}) \wedge (n >= K)$$

In addition, it is evident that the condition of line 06 in SUBROUTINE 1.1 should be rewritten to the following form:

$$(HCCR_{t_l}^{conf} < \frac{sup_{pi}^{c_j}}{sup_{pi}}) \vee (HCCR_{t_l}^{conf} = \frac{sup_{pi}^{c_j}}{sup_{pi}}) \wedge (n < K)$$

3) Traditional Definition of a Frequent Rule: The above classification rule enumeration algorithm described in Section V-C.1 can also be adapted to accord with the more traditional definition of an association rule, that is, instead of only requiring the rule body be frequent, it requires the entire rule be frequent. To simply achieve this goal, we also need to require the line 06 of SUBROUTINE 1.1 satisfy $sup_{pi}^{c_j} \geq min\_sup$ (or $sup_{pi}^{c_j} \geq min\_sup_j$ in the case of applying class-specific support threshold).

Adapting the algorithm to the traditional definition of a classification rule also enables us to design some search space pruning methods. Let the current prefix be P, a local item of P, x_j, is called infrequent and can be safely pruned according to Lemma 4.

Lemma 4: (***Infrequent item pruning***) Item x_j is called an infrequent item w.r.t. prefix P and can be safely pruned from P's conditional database if it satisfies the following equation:

$$\max_{\forall i, 1 \leq i \leq k} sup_{P \cup \{x_j\}}^{c_i} < \textbf{\textit{min\_sup}} \quad (8)$$

Proof. Follows easily from the traditional definition of a classification rule. ■

4) Maximum Support Threshold: Some dense databases contain some highly frequent items, which appear in almost all the training instances. From the classification point of view, these items are indifferentiable and cannot be used to generate high quality classification rules. Removing these items usually does not hurt the classification accuracy, but it can significantly improve the algorithm efficiency. Thus, in HARMONY there is an option for the user to specify a maximum support threshold, max_sup, in order to remove the overly frequent items.

VI. Empirical Results

A. Test Environment and Databases

We implemented the HARMONY algorithm in C and performed a thorough experimental study. We first evaluated HARMONY as a frequent itemset mining algorithm to show the effectiveness of the pruning methods, the algorithm efficiency and scalability. Then we compared HARMONY with some well-known classifiers on both categorical and text databases. All the experiments except the scalability test were performed on a 1.8GHz Linux machine with 1GB memory.

The UCI Databases. Many previous studies used some small databases to evaluate both the accuracy and efficiency of a classifier. For example, most of the 26 databases used in [28], [27], [33] only contain several hundred instances, which means the test databases contain too few test instances (i.e., only a few tens) if the 10-fold cross validation is adopted to evaluate the classification accuracy. In this paper, we mainly focus on

TABLE II
LARGE UCI DATABASE CHARACTERISTICS.

Database	# instances	# items	# classes
adult	48842	131	2
chess	28056	66	18
connect	67557	66	3
led7	3200	24	10
letRegcog	20000	106	26
mushroom	8124	127	2
nursery	12960	32	5
pageBlocks	5473	55	5
penDigits	10992	90	10
waveform	5000	108	3

TABLE III
SMALL UCI DATABASE CHARACTERISTICS.

Database	# instances	# items	# classes
anneal	798	106	6
auto	205	142	7
breast	699	48	2
glass	214	52	7
heart	303	53	5
hepatitus	155	58	2
horseColic	368	94	2
ionosphere	351	104	2
iris	150	23	3
pimaIndians	768	42	2
ticTacToe	958	29	2
wine	178	68	3
zoo	101	43	7

relatively large databases (by large, we mean the database should contain no fewer than 1000 instances), although we also report the comparison results for some small databases.

In [12], the author used 23 UCI databases to compare FOIL and CPAR algorithms [2]. Among these 23 databases, 10 of them are large databases and the left 13 databases are small ones. The characteristics of these two classes of databases are summarized in Table II and Table III, respectively. All the 23 databases were obtained from the author of [12] and the 10-fold cross validation is used for comparison with FOIL and CPAR. Because databases *connect* and *ionosphere* are too dense, during the 10-fold cross validation in our experiments HARMONY only used the items whose supports are no greater than 20,000 and 190 for *connect* and *ionosphere* respectively, to generate classification rules (i.e., max_sup=20,000 and max_sup=190 for these two databases respectively).

TABLE IV
TOP 10 TOPICS IN REUTERS-21578.

Category Name	# train labels	# test labels
acq	1650	719
corn	181	56
crude	389	189
earn	2877	1087
grain	433	149
interest	347	131
money-fx	538	179
ship	197	89
trade	369	118
wheat	212	71
total	7193	2787

Text Databases. We also used two text databases in our empirical evaluation. The first database is the popularly used 'ModeApte' split version of the reuters-21578 collection, which was preprocessed and provided by the authors of [11], and both the database and its description are available at [10]. After preprocessing, it contains totally 8575 distinct terms, 9603 training documents, and 3299 test documents. Like many other studies [23], [18], [5], [11], we are more interested in the top 10 most common categories (i.e., topics). These ten largest categories form 6488 training documents and 2545 test documents. A small portion of the training and test documents are associated with multiple category labels (that is, *reuter-21578* is a multi-class multi-label database). In our experiments, we treated each one of the training documents with multiple labels as multiple documents, each one with a distinct label. The top 10 categories and their corresponding number of labels in the training and test databases are described in Table IV. The second text database is *sports*, which was obtained from San Jose Mercury (TREC). In our experiments, we removed some highly frequent terms, and finally it contains totally 8580 documents, 7 classes, and about 1748 distinct terms. The seven classes and their corresponding number of documents are shown in Table V.

[2]The numerical attributes in these databases have been discretized by the author, and the discretization technique is different from those used in [28], [27], [33]; thus, the performance reported here may be different from the previous studies even for the same algorithm and the same database.

TABLE V
CLASS DISTRIBUTION IN *sports* DATABASE.

Class Name	Number of labels
baseball	3412
basketball	1410
football	2346
hockey	809
boxing	122
bicycle	145
golf	336
total	8580

B. Experimental Results

1) Evaluate HARMONY as a Frequent Itemset Mining Algorithm: To mine the highest confidence covering rule(s) for each instance, a naïve method is like the association-based classifiers: first use an efficient association rule mining algorithm to compute the complete set of classification rules, from which the set of the highest confidence covering rules w.r.t. each instance can be selected. Our empirical results show that this method is usually inefficient if the database is large and a more efficient way is to push some effective pruning methods into the frequent itemset mining framework and to directly mine the final set of classification rules.

Ordering of the local items. In HARMONY, we provide three options for item ordering, that is, CoRrelation coefficient Ascending order (denoted by CRA), Entropy Ascending order (denoted by EA), and Maximum Confidence Descending order (denoted by MCD). We first evaluated the effectiveness of these item ordering schemes against Support Descending order (denoted by SD) that is popularly adopted in frequent itemset mining. Our experiments on many databases have shown that the three newly proposed item ordering schemes are always more efficient that the traditional support descending ordering scheme. In addition, these schemes also lead to slightly different classification accuracy. This is partly because different item ordering schemes may mine a different highest confidence covering rule w.r.t. a certain training instance, which may have different supports, although their confidences are the same. The experimental results also show that although the correlation coefficient ascending ordering scheme is not always the winner, on average it is more efficient and has better

accuracy than the other schemes. As a result, in HARMONY, it is chosen as the default option for item ordering. Table VI shows the comparison result for *sports* database at absolute support of 200. We can see that the correlation coefficient ascending ordering scheme is more efficient and also has slightly higher classification accuracy (which was measured using 10-fold cross validation) than other schemes.

TABLE VI
ITEM ORDERING TEST ON *sports* DATABASE.

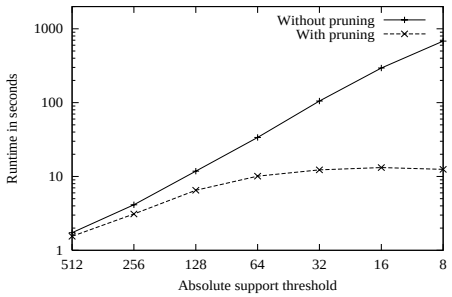
Ordering scheme	CRA	EA	MCD	SD
Runtime(in seconds)	55.7	88.6	110.8	156.3
Accuracy(in %)	85.57	85.51	85.53	85.52

Effectiveness of the pruning methods. We also evaluated the effectiveness of the pruning methods. Figure 1a shows the results for database *penDigits* with absolute support threshold varying from 512 to 8. At first glance of Equation 5 and Equation 7, the *unpromising item* and *conditional database* pruning methods seem to be less effective at lower support, however this is not the case when considering more covering rules with higher confidence can be found at lower support and can be used to more quickly raise the currently maintained highest confidences. As we can see from Figure 1a, if we turn off the pruning methods used in HARMONY (denoted by 'without pruning'), it can become over an order of magnitude slower at low support.

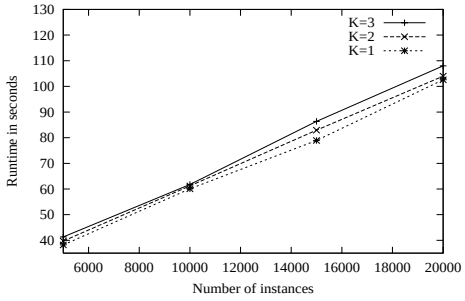
Scalability test. Figure 1b shows the results of the scalability test performed on an Intel Pentium IV processor computer with 256 MB memory using database *letRecog* with relative support set at 0.5% . The three curves in Figure 1b were generated by setting parameter K at values 1, 2, and 3, and using one fourth, one half, three fourths, and the whole of database *letRecog*, as the input databases with different base size, respectively. We can see that HARMONY has linear scalability in the runtime with increasing number of instances.

Efficiency test. As we mentioned above, the traditional frequent (closed) itemset mining algorithms can be revised to mine the complete set of high confidence classification rules, from which a subset of high quality rules can be further identified. Our efficiency tests for HARMONY in comparison with FPgrowth* and FPclose, two recently developed efficient frequent/closed itemset mining algorithms [21], show that such a method is not realistic at low support, while our experiments demonstrate that the classification accuracy is usually higher at low support.

Figure 2 shows the comparison results for database *sports*. As we can see, although at high support, both FPgrowth* and FPclose are faster than HARMONY, once we continue to lower the support, they will be much slower. For example, at absolute support of 100, HARMONY is several orders of magnitude faster than FPgrowth* and FPclose. Figure 2b shows the classification accuracy at different support thresholds using the 10-fold cross validation. We can see that HARMONY can achieve higher accuracy at lower support like 100. It is also interesting to see that the accuracy at a too low support 50 is worse than that at support 100 for this database, due to the 'overfitting' problem.

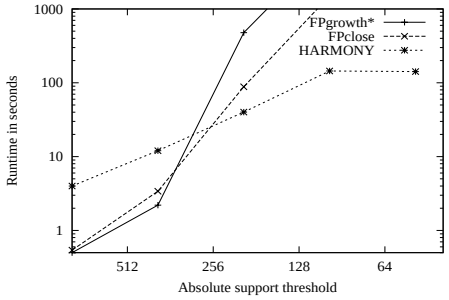


a) Pruning (*penDigits*)

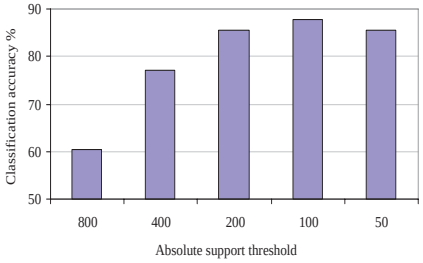


b) Scalability (*letRecog, min_sup=0.5%*)

Fig. 1. Pruning and scalability test.



a) *Runtime comparison*



b) *Classification accuracy*

Fig. 2. Efficiency test (*sports*).

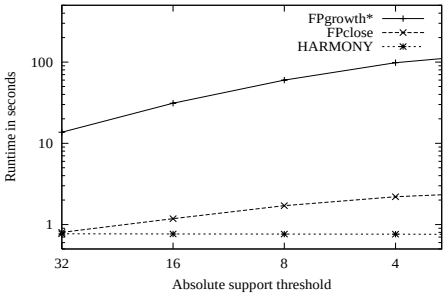
Figure 3a shows similar comparison results for categorical database *mushroom*. HARMONY is faster than both FPgrowth* and FPclose at absolute support lower than 32. Figure 3b shows that HARMONY has better accuracy at low support threshold.

2) Classification Evaluation: **The reuters–21578 (*ModApte*) text database.** For a multi-class multi-label database like *reuters-21578*, most previous studies used

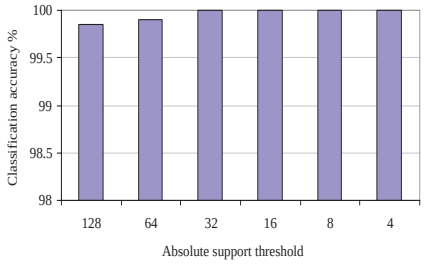
TABLE VII

BREAKEVEN PERFORMANCE ON *Reuters-21578* WITH SOME WELL-KNOWN CLASSIFIERS.

Categories	HARMONY *min_sup=60*	HARMONY *min_sup=70*	HARMONY *min_sup=80*	Findsim	NBayes	BayesNets	Trees	SVM (linear)	ARC-BC δ=50, min_sup=10%
acq	**95.3**	**95.3**	**95.3**	64.7	87.8	88.3	89.7	93.6	90.9
corn	78.2	78.6	75.2	48.2	65.3	76.4	**91.8**	90.3	69.6
crude	85.7	85.0	88.0	70.1	79.5	79.6	85.0	**88.9**	77.9
earn	**98.1**	**98.2**	97.6	92.9	95.9	95.8	97.8	98.0	92.8
grain	91.8	90.4	90.1	67.5	78.8	81.4	85.0	**94.6**	68.8
interest	77.3	76.6	75.1	63.4	64.9	71.3	67.1	**77.7**	70.5
money-fx	**80.5**	**81.9**	**82.1**	46.7	56.6	58.8	66.2	74.5	70.5
ship	**86.9**	82.9	82.8	49.2	85.4	84.4	74.2	85.6	73.6
trade	**88.4**	**88.0**	**86.1**	65.1	63.9	69.0	72.5	75.9	68.0
wheat	62.8	60.6	58.7	68.9	69.7	82.7	**92.5**	91.8	84.8
micro-avg	**92.0**	91.7	91.4	64.6	81.5	85.0	88.4	**92.0**	82.1



a) Runtime comparison



b) Classification accuracy

Fig. 3. Efficiency test (*mushroom*).

TABLE VIII

RUNTIME ON *reuters-21578* DATABASE.

Runtime	HARMONY (*min_sup=60*)	HARMONY (*min_sup=70*)	HARMONY (*min_sup=80*)
training	72.6	51.1	37.6
testing	0.363	0.337	0.309

the breakeven point of precision and recall to measure the classifier performance [6], [23], [18], [32], [9], [5], [11], which is defined as the point at which precision is equal to the recall. To our best knowledge, the best breakeven performance for the *reuters-21578* database is the linear SVM [18]. For comparison with earlier results, we first found the overall breakeven point in terms of all top 10 categories by adjusting the dominant factor γ, then reported the average of precision and recall for each category as their corresponding breakeven performance [18].

Table VII shows the comparison results with some previous results. The results for Findsim (i.e., Find-Similar), NBayes (i.e., Naïve-Bayes), Bayes-Nets, Trees (i.e., Decision-Trees), and LinearSVM were obtained from [18], while the results for ARC-BC are reported as given in [5]. The micro-avg is the overall breakeven performance over all 10 categories. For HARMONY, we used three different uniform absolute support thresholds, 60, 70, and 80, respectively. From Table VII we can see that both HARMONY and LinearSVM have similar breakeven performance and perform much better than all the other classifiers, including Find-Similar, Naïve-Bayes, Bayes-Nets, Decision-Trees, and the association-based classifier ARC-BC. Among the 10 categories, HARMONY achieves the best performance at support of 60 for five categories, *acq*, *earn*, *money-fx*, *ship*, and *trade*. While LinearSVM performs best for another three categories, *crude*, *grain*, and *interest*. Decision-Trees also performs good and has the best performance for two small categories, *corn* and *wheat*. SVM is very well known for classifying high dimensional text databases. Our results show that HARMONY can achieve similar performance to SVM. Table VIII shows the runtime in seconds for HARMONY at three different support thresholds. We can see that HARMONY is very efficient in classifying the *reuters–21578* text database. For example, at absolute support of 60, it takes 72.6 seconds to build the model from 7193 training documents, and 0.363 seconds to classify 2545 test documents.

TABLE X

EFFECTIVENESS OF THE *score differentia factor* δ (CLASS-SPECIFIC *min_sup*, *reuters-21578*).

δ	0	0.3	0.6	0.9	1.2
γ	0.4987	0.5029	0.517	0.544	0.569
micro-avg	91.2	91.6	92.2	92.4	92.1

The micro-avg values for HARMONY in Table VII were computed by using a uniform minimum support for all 10 categories and our experiments show that choosing a uniform minimum absolute support which is smaller than 60 does not help in further improving classification accuracy. However, if we choose some proper class-specific support thresholds for different categories, HARMONY can still achieve better performance. In Table IX, the second row shows the corresponding minimum support chosen for the 10 categories, while the third row shows the breakeven performance. From the results we can see that with these class-specific support thresholds, HARMONY achieves a better micro-avg value, 92.4. This example illustrates that adopting different support thresholds for different classes does achieve better results.

The micro-avg values for HARMONY in Table VII and Table IX were computed by setting the *score differentia factor* at its default value 1. By choosing different *differ-*

TABLE IX
BREAKEVEN PERFORMANCE ON THE *Reuters-21578* DATABASE WITH CLASS-SPECIFIC *min_sup*.

Categories	acq	corn	crude	earn	grain	interest	money-fx	ship	trade	wheat	micro-avg
min_sup	55	60	60	75	60	55	70	50	60	45	-
Breakeven	95.6	78.2	86.6	97.9	90.7	76.8	83.5	88.5	89.3	68.1	92.4

entia factor values, HARMONY may have different micro-avg performance. Table X shows the micro-avg performance and the corresponding dominant factor γ for HARMONY with the class-specific support thresholds shown in Table IX and by varying the parameter of *score differentia factor* δ from 0 to 1.2. $\delta = 0$ means the weighted dominant factor-based scoring method degenerates to the dominant factor-based method used in [5]. By adopting a proper value of δ, the weighted dominant factor-based scoring method can achieve a better micro-avg performance. For example, by setting δ at 0.9, the overall precision equals the overall recall at $\gamma = 0.544$, and the corresponding micro-avg breakeven performance for HARMONY is 92.4, which is higher than the corresponding micro-avg at $\delta = 0$ (i.e., 91.2). As our experiments show usually a *differentia factor* value around 1 can achieve good accuracy, in HAMONY we set a default value of 1 for *differentia factor*.

The *sports* text database. From Table VII we see that HARMONY has similar performance to SVM for the *reuters-21578* text database, we then compared the two algorithms using the *sports* text database which was obtained from San Jose Mercury. We split it into two parts: the training set contains 5718 instances and the test set contains 2852 instances. We first tried the *polynomial*, *radial basis function*, and *sigmoid* kernels for SVM, however, SVM cannot finish within a reasonable time slot for the *sports* database (for example, *sigmoid* SVM could not terminate after running for more than 80,000 seconds). Thus, here we only report the performance for linear SVM. Table V shows the comparison results for HARMONY and linear SVM. In the experiments we chose four different minimum supports for HARMONY, 75, 100, 125, and 150, while for linear SVM, we chose four values for parameter C (i.e., the trade-off between training error and margin), 0.25, 0.5, 1, and 2, respectively. Under these parameter settings, HARMONY and SVM have similar runtime efficiency. From Table XI we see that SVM has slightly better accuracy than HARMONY.

TABLE XI
ACCURACY COMPARISON BETWEEN HARMONY AND SVM (*sports*).

HARMONY	94.2	94.9	94.3	94.1
(*min_sup*)	(75)	(100)	(125)	(150)
SVM	95.79	95.79	95.76	95.72
(*C*)	(2.0)	(1.0)	(0.5)	(0.25)

The UCI databases. We evaluated HARMONY on the UCI databases in comparison with FOIL, CPAR, and SVM. FOIL and CPAR are two well-known algorithms for classifying categorical data. The results in [33] show that CPAR has better accuracy than c4.5 [29] and ripper [13], and has comparable accuracy to the association-based algorithms CMAR [27] and CBA [28], but is orders of magnitude faster; thus, we will do not compare HARMONY with c4.5, ripper, and the association-based algorithms. The results for FOIL and CPAR were provided by Frans Coenen and are available at [12]. Because most databases we used contain more than two class labels, when comparing with SVM, we used $SVM^{multiclass}$ (Version: 1.01), which is an implementation of the multi-class Support Vector Machine and is available at $http://www.cs.cornell.edu/People/tj/svm\_light/svm\_multiclass.html$. In the experiments, we ran SVM with its default setting. All the results including the accuracy and runtime are computed using the 10-fold cross validation. The reported accuracy is the corresponding average value of the 10-fold cross validation results, while the runtime is the total runtime of the 10-fold cross validation, including both training and testing time. In the experiments, we fixed the absolute support threshold at 50 for HARMONY with all 10 UCI databases, and at 10 for all 13 small UCI databases.

Table XII shows the accuracy comparison results, which reveal that HARMONY has much better overall accuracy than FOIL and CPAR, and has comparable accuracy with SVM. The average accuracy of HARMONY over all 10 UCI databases is about 5% higher than FOIL, 10% higher than CPAR, and 2% higher than SVM. SVM performs very well for the databases with few class labels, like *adult*, *connect*, and *waveform*, but has much worse accuracy than HARMONY for the databases with many class labels, like *chess* and *letRecog*. Compared with SVM, HARMONY has reasonably stable and good performance over all 10 UCI databases. Note in the experiments we fixed the minimum support at 50 for all 10 UCI databases. If we choose some tuned supports, HARMONY can achieve better performance than what we reported here for some databases. For example, if we choose the minimum support at 5 for the *chess* database, HARMONY has an accuracy of 58.43%, which is over 13% higher than the accuracy at support 50, while it only becomes about two times slower.

TABLE XII
ACCURACY COMPARISON ON 10 LARGE *UCI* DATABASES (*min_sup=50* FOR HARMONY).

Database	FOIL	CPAR	SVM	HARMONY
adult	82.5	76.7	**84.16**	81.9
chess	42.6	32.8	29.83	**44.87**
connect	65.7	54.3	**72.5**	68.05
led7	62.3	71.2	73.78	**74.56**
letRecog	57.5	59.9	67.76	**76.81**
mushroom	99.5	98.8	99.67	**99.94**
nursery	91.3	78.5	91.35	**92.83**
pageBlocks	**91.6**	76.2	91.21	**91.6**
penDigits	88.0	83.0	93.2	**96.23**
waveform	75.6	75.4	**83.16**	80.46
average	75.66	70.68	78.663	**80.725**

Table XIII compares the runtime (in seconds) of the four algorithms. Note that FOIL and CPAR were implemented in java and were tested on a different machine from that

TABLE XIII
RUNTIME COMPARISON ON 10 LARGE *UCI* DATABASES (*min_sup=50* FOR HARMONY).

Database	FOIL	CPAR	SVM	HARMONY
adult	10251.0	**809.0**	2493.1	1395.5
chess	10122.8	1736.0	13289.4	**11.34**
connect	35572.5	24047.1	74541.1	**85.44**
led7	11.5	5.7	17.12	**1.29**
letRecog	4365.6	**764.0**	17825.2	778.91
mushroom	38.3	15.4	16.6	**8.78**
nursery	73.1	51.7	322.4	**6.21**
pageBlocks	43.1	15.5	11.2	**2.5**
penDigits	821.1	101.9	512.7	**82.6**
waveform	295.3	38.1	**36.2**	130.0
total	61594.3	27584.4	109065.02	**2502.57**

TABLE XIV
COMPARISON OF # RULES ON 10 LARGE *UCI* DATABASES (*min_sup=50* FOR HARMONY).

Database	FOIL	CPAR	HARMONY
adult	331.8	183.1	6431.3
chess	1116.7	1504.8	2881.5
connect	285.8	816.1	6664.0
led7	80.6	31.4	268.7
letRecog	560.9	643.0	2255.8
mushroom	16.2	30.8	95.9
nursery	57.4	83.6	391.64
pageBlocks	123.1	56.2	78.1
penDigits	204.6	166.9	1434.5
waveform	159.7	114.3	958.6
average	293.68	363.02	2146.0

of HARMONY and SVM. As a result, their runtime cannot be directly compared to those reported for HARMONY and SVM but they only provide an overall idea on the relative computational requirements of the various schemes. Table XIII shows that on average the runtime of HARMONY is over an order of magnitude smaller than those of FOIL, CPAR, and SVM. For some large databases like *chess*, the runtime of HARMONY can be over two orders of magnitude smaller than those of FOIL and CPAR, and over three orders of magnitude smaller than that of SVM. Table XIV compares the number of classification rules discovered by three rule-based algorithms, FOIL, CPAR, and HARMONY. We can see that on average, HARMONY finds many more rules than both FOIL and CPAR. The reason why HARMONY finds more rules is that it mines classification rules in an instance-centric manner: it guarantees that at least one of the highest confidence covering rules for each instance is discovered.

Table XV depicts the accuracy comparison among FOIL, CPAR, SVM, and HARMONY on 13 small UCI databases, from which we can see that on average HARMONY and FOIL have similar classification accuracy and both perform a little better than CPAR and SVM. In the experiments we fixed the minimum absolute support at 10 on all 13 small UCI databases for HARMONY, if we use tuned minimum support, HARMONY can achieve better accuracy for most databases. In addition, because these databases contain a small number of training instances, the number of classification rules mined by HARMONY may not be sufficient to build an accurate classification model; thus, we implemented a variant of HARMONY based on the discussion in Section V-D.2, which mines K highest confidence frequent covering rules for each training instance if it supports no fewer than K such rules. By varying K parameter from 1 to 5, and choosing the minimum absolute support from $\{5, 10, 15\}$, we got a set of classification results, among which the best results for 13 small UCI databases are shown in Table XVI. Similarly, we ran SVM with radial basis function (i.e., *rbf*) kernel and the tuned best accuracy was reported by choosing the value of parameter γ from $\{0.5, 0.75, 1.0\}$. We can see that with tuned parameters, HARMONY achieves better overall classification accuracy than FOIL, CPAR, Linear SVM, and *rbf* SVM, on the other hand, *rbf* SVM achieves better classification accuracy than HARMONY for databases *anneal*, *glass*, *wine*, and *zoo*, and FOIL has better accuracy than other classifiers for database *ticTacToe*. Table XVI also shows that HARMONY achieves its highest accuracy with different minimum supports for different databases. Currently HARMONY cannot dynamically adjust the minimum support to get the best classification accuracy, instead one needs to manually choose a proper minimum support in order for HARMONY to get good classification results. However, this issue is not unique to HARMONY, many classifiers (such as FOIL, CPAR, and SVM) also require a user to manually specify some input parameter values.

TABLE XV
ACCURACY COMPARISON ON 13 SMALL *UCI* DATABASES (*min_sup=10* FOR HARMONY).

Database	FOIL	CPAR	SVM	HARMONY
anneal	**96.9**	90.2	83.83	91.51
auto	46.1	48.0	55.5	**61**
breast	94.4	94.8	**96.8**	92.42
glass	49.3	48.0	46.0	**49.8**
heart	57.4	51.1	**60.36**	56.46
hepatitus	77.5	76.5	81.83	**83.16**
horseColic	**83.5**	82.3	83.31	82.53
ionosphere	89.5	**92.9**	89.44	92.03
iris	94.0	**94.7**	94.67	93.32
pimaIndians	73.8	**75.6**	74.18	72.34
ticTacToe	**96.0**	72.2	70.78	92.29
wine	86.4	92.5	**94.895**	91.94
zoo	**96.0**	**96.0**	86.0	93.0
average	80.06	78.06	78.28	**80.82**

TABLE XVI
TUNED ACCURACY ON 13 SMALL *UCI* DATABASES FOR HARMONY AND SVM.

Database	*min_sup* $\in \{5, 10, 15\}$	K rules $\in [1, 5]$	HARMONY	SVM (*rbf*) $\gamma \in \{0.5, 0.75, 1.0\}$
anneal	5	4	95.65	**97.26**
auto	10	1	**61.5**	58.9
breast	15	2	**96.14**	95.09
glass	10	1	49.8	**50.53**
heart	15	1	**58.4**	57.46
hepatitus	15	5	**85.99**	85.5
horseColic	5	4	**84.64**	84.06
ionosphere	10	2	**93.45**	89.43
iris	5	5	**95.99**	93.33
pimaIndians	5	5	**73.79**	71.06
ticTacToe	10	4	**94.09**	88.52
wine	10	5	94.9	**97.25**
zoo	5	1	96	**97**
average	N/A	N/A	**83.1**	81.95

TABLE XVII
EFFECTIVENESS OF THE *support differentia factor* ξ (*min_sup=10, hepatitus*).

ξ	0	0.2	0.4	0.6	0.8	1	1.2	1.4
accuracy	83.16	85.33	85.33	83.5	84.33	84.83	84.16	83.83

We also evaluated the effectiveness of the *support differentia factor* ξ using one imbalanced database, *hepatitus*. In Table XVII, we set *min_sup* at 10 and varied ξ from 0 to 1.4. We can see that the *support differentia factor* based class-specific support threshold method is effective in improving the

accuracy for this database. For example, with $\xi = 0.2\ (or\ 0.4)$, HARMONY has an accuracy of 85.33%, which is about 2% higher than that at the default value $\xi = 0$. Our tests with several databases suggest us that usually with a small *support differentia factor* value (e.g., $0 < \xi \leq 1$), HARMONY can achieve a good classification accuracy for imbalanced databases.

VII. Conclusion

Designing accurate, efficient, and scalable classifiers is an important research topic in data mining, and the rule-based classifiers have been proven very effective in classifying the categorical or high-dimensional sparse data. However, to achieve high accuracy, a good rule-based classifier needs to find a sufficient number of high quality classification rules and use them to build the model. In this paper, we proposed an instance-centric classification rule mining paradigm and designed an accurate classifier, HARMONY. Several effective pruning methods and search strategies have also been proposed, which can be pushed deeply into the projection-based frequent itemset enumeration framework. Our performance study shows that HARMONY has high accuracy and efficiency in comparison with many well known classifiers for both the categorical data and the high dimensional text data. It also has good scalability in terms of the base size.

Acknowledgements

We are grateful to Frans Coenen and Shane Bergsma for providing us the UCI databases and the reuters-21578 database, respectively, and Osmar R. Zaïane for answering our questions related to the ARC-BC algorithm. We also thank the anonymous ICDE'05 and SDM'05 reviewers whose comments helped a lot in improving this paper. This work was supported in part by NSF CCR-9972519, EIA-9986042, ACI-9982274, ACI-0133464, and ACI-0312828; the Digital Technology Center at the University of Minnesota; and by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement number DAAD19-01-2-0014. The content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Super-computing Institute. Jianyong Wang was funded in part by National Natural Science Foundation of China (NSFC) under Grant No. 60573061 and Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList), and this paper is a major-value added version of a conference paper that appeared in the 2005 SIAM International Conference on Data Mining (SDM'05).

References

[1] R. Agarwal, C. Aggarwal, V. Prasad. *A Tree Projection Algorithm for Generation of Frequent Item Sets*, Journal of Parallel and Distributed Computing. 61(3), 2001.
[2] R. Agrawal, T. Imielinski, A. Swami. *Mining Association Rules between Sets of Items in Large Databases*, SIGMOD'93.
[3] R. Agrawal, R. Srikant. *Fast Algorithms for Mining Association Rules*, VLDB'94.
[4] K. Ali, S Manganaris, R. Srikant. *Partial Calssification Using Association Rules*, KDD'97.
[5] M. Antonie, O. Zaiane. *Text Document Categorization by Term Association*, ICDM'02.
[6] C. Apte, F. Damerau, S.M. Weiss. *Towards Language Independent Automated Learning of Text Categorization Models*, SIGIR'94.
[7] R.J. Bayardo. *Brute-force Mining of High-confidence Classification rules*, KDD'97.
[8] R.J. Bayardo, R. Agrawal. *Mining the most interesting rules*, KDD'99.
[9] R. Bekkerman, R. EI-Yaniv, N. Tishby, Y. Winter. *On Feature Distribution Clustering for Text Categorization*, SIGIR'01.
[10] S. Bergsma. *The Reuters-21578 (ModApte) dataset*, Department of Computer Science, University of Alberta. Available at http://www.cs.ualberta.ca/~bergsma/650/.
[11] S. Bergsma, D. Lin. *Title Similarity-Based Feature Weighting for Text Categorization*, CMPUT 650 Research Project Report, Department of Computer Science, University of Alberta.
[12] F. Coenen. (2004) The LUCS-KDD Implementations of the FOIL, PRM, and CPAR algorithms, http://www.csc.liv.ac.uk/~frans/KDD/Software/FOIL_PRM_CPAR/foilPrmCpar.html, Computer Science Department, University of Liverpool, UK.
[13] W. Cohen. *Fast effective rule induction*, ICML'95.
[14] G. Cong, K. Tan, A. Tung, X. Xin. *Mining Top-k Covering Rule Groups for Gene Expression Data*, SIGMOD'05.
[15] G. Cong, X. Xu, F. Pan, A. Tung, J. Yang. *FARMER: Finding Interesting Rule Groups in Microarray Datasets*, SIGMOD'04.
[16] M. Deshpande, G. Karypis. *Using Conjunction of Attribute Values for Classification*, CIKM'02.
[17] G. Dong, X. Zhang, L. Wong, J. Li. *CAEP: Classification by aggregating emerging patterns*, DS'99.
[18] S. Dumais, J. Platt, D. Heckerman, M. Sahami. *Inductive Learning Algorithms and Representations for Text Categorization*, CIKM'98.
[19] T. Fukuda, Y. Morimoto, S. Motishita. *Constructing Efficient Decision Trees by Using Optimized Numeric Association Rules*, VLDB'96.
[20] K. Gade, J. Wang, G. Karypis. *Efficient Closed Pattern Mining in the Presence of Tough Block Constraints*, KDD'04.
[21] G. Grahne, J. Zhu. *Efficiently Using Prefix-trees in Mining Frequent Itemsets*, FIMI'03.
[22] J. Han, et al. *Mining Frequent Patterns without Candidate Generation*, SIGMOD'00.
[23] T. Joachims. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*, ECML'98.
[24] B. Lent, A. Swami, J. Widom. *Clustering Association Rules*, ICDE'97.
[25] N. Lesh, M. Zaki, M. Ogihara. *Mining Features for Sequence Classification*, KDD'99.
[26] J. Li, G. Dong, K. Ramamohanarao, L. Wong. *DeEPs: A New Instance-based Discovery and Classification System*, Machine Learning, 54(2), 2004.
[27] W. Li, J. Han, J. Pei. *CMAR: Accurate and Efficient Classification based on multiple class-association rules*, ICDM'01.
[28] B. Liu, W. Hsu, Y. Ma. *Integrating Classification and Association Rule Mining*, KDD'98.
[29] J. Quinlan. *C4.5: Programs for Machine Learning*, Morgan Kaufman, 1993.
[30] J. Quinlan, R. Cameron-Jones. *FOIL: A Midterm Report*, ECML'93.
[31] J. Wang, G. Karypis. *BAMBOO: Accelerating Closed Itemset Mining by Deeply Pushing the Length-Decreasing Support Constraint*, SDM'04.
[32] Y. Yang. *An Evaluation of Statistical Approaches to Text Categorization*, Information Retrieval, Vol. 1, No. 1-2, 1999.
[33] X. Yin, J. Han. *CPAR: Classification based on Predictive Association Rules*, SDM'03.
[34] M. Zaki, C. Aggarwal. *XRULES: An Effective Structural Classifier for XML Data*, KDD'03.

Jianyong Wang received the PhD degree in computer science in 1999 from the Institute of Computing Technology, the Chinese Academy of Sciences. Since then, he has worked as an assistant professor in the Department of Computer Science and Technology, Peking University in the areas of distributed systems and Web search engines, and visited the School of Computing Science at Simon Fraser University, the Department of Computer Science at the University of Illinois at Urbana-Champaign, and the Digital Technology Center and Department of Computer Science and Engineering at the University of Minnesota, mainly working in the area of data mining. He is currently an associate professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is a member of the IEEE Computer Society, and the ACM SIGKDD.

George Karypis received his Ph.D. degree in computer science at the University of Minnesota and he is currently an Associate Professor at the Department of Computer Science and Engineering at the University of Minnesota. His research interests spans the areas of parallel algorithm design, data mining, bioinformatics, information retrieval, applications of parallel processing in scientific computing and optimization, sparse matrix computations, parallel preconditioners, and parallel programming languages and libraries. His research has resulted in the development of software libraries for serial and parallel graph partitioning (METIS and ParMETIS), hypergraph partitioning (hMETIS), for parallel Cholesky factorization (PSPASES), for collaborative filtering-based recommendation algorithms (SUGGEST), clustering high dimensional datasets (CLUTO), and finding frequent patterns in diverse datasets (PAFI). He has coauthored over one hundred journal and conference papers on these topics and a book title “Introduction to Parallel Computing” (Publ. Addison Wesley, 2003, 2nd edition). In addition, he is serving on the program committees of many conferences and workshops on these topics, is an associate editor of the IEEE Transactions on Parallel and Distributed Systems, and on the editorial board of the International Journal of Data Mining and Bioinformatics.