CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules *

Wenmin Li Jiawei Han Jian Pei†
School of Computing Science, Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
E-mail: {wli, han, peijian}@cs.sfu.ca

Abstract

Previous studies propose that associative classification has high classification accuracy and strong flexibility at handling unstructured data. However, it still suffers from the huge set of mined rules and sometimes biased classification or overfitting since the classification is based on only single high-confidence rule.

In this study, we propose a new associative classification method, CMAR, *i.e.,* **C***lassification based on* **M***ultiple* **A***ssociation* **R***ules. The method extends an efficient frequent pattern mining method,* FP-growth*, constructs a class distribution-associated* FP-tree*, and mines large database efficiently. Moreover, it applies a* CR-tree *structure to store and retrieve mined association rules efficiently, and prunes rules effectively based on confidence, correlation and database coverage. The classification is performed based on a weighted χ^2 analysis using multiple strong association rules. Our extensive experiments on* 26 *databases from UCI machine learning database repository show that* CMAR *is consistent, highly effective at classification of various kinds of databases and has better average classification accuracy in comparison with CBA and C4.5. Moreover, our performance study shows that the method is highly efficient and scalable in comparison with other reported associative classification methods.*

1 Introduction

Building accurate and efficient classifiers for large databases is one of the essential tasks of data mining and machine learning research. Given a set of cases with class labels as a *training set*, *classification* is to build a model (called *classifier*) to predict future data objects for which the class label is unknown.

Previous studies have developed heuristic/greedy search techniques for building classifiers, such as decision trees [10], rule learning [2], naïve-Bayes classification [4], and statistical approaches [8]. These techniques induces a representative subset of rules (e.g., a decision tree or a set of rules) from training data sets for quality prediction.

Recent studies propose the extraction of a set of high quality association rules from the training data set which satisfy certain user-specified frequency and confidence thresholds. Effective and efficient classifiers have been built by careful selection of rules, e.g., CBA [9], CAEP [3], and ADT [11]. Such a method takes the most effective rule(s) from among all the rules mined for classification. Since association rules explore highly confident associations among multiple variables, it may overcome some constraints introduced by a decision-tree induction method which examines one variable at a time. Extensive performance studies [6, 9, 3, 11] show that association-based classification may have better accuracy in general. However, this approach may also suffer some weakness as shown below.

On one hand, *it is not easy to identify the most effective rule at classifying a new case.* Some method, such as [9, 3, 11], simply selects a rule with a maximal user-defined measure, such as confidence. As we will see later, such a selection may not always be the right choice in many cases. Such a simple pick may affect the classification accuracy.

On the other hand, *a training data set often generates a huge set of rules. It is challenging to store, retrieve, prune, and sort a large number of rules efficiently for classification.* Many studies [1, 5] have indicated the inherent nature of a combinatorial explosive number of frequent patterns and hence association rules that could be generated when the support threshold is small (i.e., when rare cases are also be included in the consideration). To achieve high accuracy, a classifier may have to handle a large set of rules, including storing those generated by association mining methods, retrieving the related rules, and pruning and sorting a large number of rules.

Can we solve the above two problems? To solve the first problem, that is, to predict a new case accurately, instead of applying only a single rule, one may consider a small subset of the most related, highly confident rules and

*The work was supported in part by the Natural Sciences and Engineering Research Council of Canada (grant NSERC-A3723), and the Networks of Centres of Excellence of Canada (grant NCE/IRIS-3).

†Contact author.

make a collective and all-around decision. Intuitively, that would help us avoid bias, exceptions, and overfitting of too small data sets. To overcome the second problem, the efficiency and scalability problem of association-based classification, one needs to develop efficient methods for storing and retrieving rules. This may help improve efficiency as well as the accuracy of classification since more rules can be stored and considered. This is the motivation of this research.

In this paper, we develop a new technique, *CMAR*, for accurate and efficient classification and make the following contributions.

First, *instead of relying on a single rule for classification,* CMAR *determines the class label by a set of rules.* Given a new case for prediction, *CMAR* selects a small set of high confidence, highly related rules and analyzes the correlation among those rules. To avoid bias, we develop a new technique, called *weighted* χ^2, which derives a good measure on how strong the rule is under both conditional support and class distribution. An extensive performance study shows that *CMAR* in general has higher prediction accuracy than CBA [9] and C4.5 [10].

Second, *to improve both accuracy and efficiency,* CMAR *employs a novel data structure,* CR-tree, *to compactly store and efficiently retrieve a large number of rules for classification. CR-tree* is a prefix tree structure to explore the sharing among rules, which achieves substantial compactness. *CR-tree* itself is also an index structure for rules and serves rule retrieval efficiently.

Third, *to speed up the mining of complete set of rules,* CMAR *adopts a variant of recently developed* FP-growth *method. FP-growth* is much faster than *Apriori*-like methods used in previous association-based classification, such as [9, 3, 11], especially when there exist a huge number of rules, large training data sets, and long pattern rules.

The remaining of the paper is arranged as follows. Section 2 revisits the general idea of associative classification. Section 3 devotes to the generation of rules for classification. Section 4 discusses how to classify a new data object using the generated rules. The experimental results on classification accuracy and the performance study on efficiency and scalability are reported in Section 5. The paper is concluded in Section 6.

2 Associative Classification

Suppose a data object $obj = (a_1, \ldots, a_n)$ follows the schema $(A_1, \ldots, A_n)$, where A_1, ..., A_n are called *attributes*. Attributes can be categorical or continuous. For a categorical attribute, we assume that all the possible values are mapped to a set of consecutive positive integers. For a continuous attribute, we assume that its value range is discretized into intervals, and the intervals are also mapped to consecutive positive integers. By doing so, all the attributes are treated uniformly in this study.

Let $C = \{c_1, \ldots, c_m\}$ be a finite set of *class labels*. A *training data set* is a set of data objects such that, for each object obj, there exists a class label $c_{obj} \in C$ associated with it. A *classifier* $\mathcal{C}$ is a function from $(A_1, \ldots, A_n)$ to C. Given a data object obj, $\mathcal{C}(obj) \in C$ returns a class label.

In general, given a training data set, the task of **classification** is to build a classifier from the training data set such that it can be used to predict class labels of unknown objects with high accuracy.

Besides many different approach for classification, such as decision tree approach, naive Bayesian approach, k-nearest neighbors approach, neural network approach, a new approach is to explore association relationships between object conditions and class labels [9]. The idea is natural since it utilizes frequent patterns and association relationships between cases and class labels in training data set to do classification. If strong associations among some frequent patterns and class labels can be observed in training data set, the future object of similar patterns can be classified.

In general, a **pattern** $P = a_{i_1} \ldots a_{i_k}$ is a set of attribute-values such that for $(1 \leq j \leq k)$, $a_{i_j} \in A_{i_j}$, and $i_j \neq i_{j'}$ for $j' \neq j$. A data object obj is said to **match** pattern $P = a_{i_1} \ldots a_{i_k}$ if and only if for $(1 \leq j \leq k)$, obj has value a_{i_j} in attribute A_{i_j}.

Given a training data set T, let c be a class label. For rule $R : P \rightarrow c$, the number of data objects in T matching pattern P and having class label c is called the **support** of R, denoted as $sup(R)$. The ratio of the number of objects matching pattern P and having class label c versus the total number of objects matching pattern P is called the **confidence** of R, denoted as $conf(R)$.

For example, if 95% of customers who have no job cannot get a credit limit more than $\$3000$, i.e., the confidence of rule $R : \textit{no\_job} \rightarrow \textit{credit\_limit\_less\_than\_3000}$ is 95%, then we can use rule R to classify future data objects. To avoid noise, a rule is used for classification only if it has enough support. Given a support threshold and a confidence threshold, **associative classification** method finds the complete set of class-association rules (CAR) passing the thresholds. When a new (unknown) object comes, the classifier selects the rule which matches the data object and has the highest confidence and uses it to predict the class label of the new object.

Recent studies show that associative classification is intuitive and effective and has good classification accuracy in many cases. In most existing associative classification methods [9, 3, 11], the rule with the highest confidence is used for classification. However, such a decision may not always be the correct one.

For example, suppose we want to determine the credit limit of a customer with attribute values (*no_job*, *investment_immigrant*, oversea_asset$\geq 500k$). The top-3 most confident rules matching the customer are as follows.

- Rule R_1: *no_job* $\rightarrow$ *credit_limit_*3000^- (support:

3000, confidence: 95%);

- Rule R_2: *investment_immigrant* $\rightarrow$ *credit_limit_*3000^+ (support: 5000, confidence: 93%); and
- Rule R_3: *oversea_asset* $\geq$ $500k$ $\rightarrow$ *credit_limit_*3000^+ (support: 8000, confidence: 91%).

So, given such a customer, what class label should we predict?

A conventional associative classification method, like CBA, may predict *credit_limit_*3000^- according to rule R_1 only, since it has the highest confidence. However, a closer look at rules R_2 and R_3 may suggest that we reconsider the decision seriously. The three rules have very similar confidence, but R_2 and R_3 have stronger support. The decision based on rules R_2 and R_3 seems to be more reliable.

The above example indicates that to make a reliable and accurate prediction, the most confident rule may not always be the best choice, and a thorough, detailed, and all-around measure analysis based on multiple rules may lead to better quality prediction.

3 Generating Rules for Classification

In this section, we develop a new associative-classification method, called *CMAR*, which performs **C**lassification based on **M**ultiple **A**ssociation **R**ules.

CMAR consists of two phases: *rule generation* and *classification*.

In the first phase, *rule generation*, *CMAR* computes the *complete* set of rules in the form of $R: P \rightarrow c$, where P is a pattern in the training data set, and c is a class label such that $sup(R)$ and $conf(R)$ pass the given support and confidence thresholds, respectively. Furthermore, *CMAR* prunes some rules and only selects a subset of high quality rules for classification.

In the second phase, *classification*, for a given data object obj, *CMAR* extracts a subset of rules matching the object and predicts the class label of the object by analyzing this subset of rules.

In this section, we develop methods to generate rules for classification. The second phase, classification, will be discussed in Section 4.

3.1 Mining Class-Association Rules Passing Support and Confidence Thresholds

To find rules for classification, *CMAR* first mines the training data set to find the complete set of rules passing certain support and confidence thresholds. This is a typical frequent pattern or association rule mining task [1]. To make mining highly scalable and efficient, *CMAR* adopts a variant of *FP-growth* method [5]. *FP-growth* is a frequent pattern mining algorithm which is faster than conventional *Apriori*-like methods, especially in the situations where there exist large data sets, low support threshold, and/or long patterns. The general idea of mining rules in *CMAR* is shown in the following example.

Example 1 (Mining class-association rules) Given a training data set T as shown in Table 1. Let the support threshold is 2 and confidence threshold is 50%. *CMAR* mines class-association rules as follows.

Row-id	A	B	C	D	Class label
1	a_1	b_1	c_1	d_1	A
2	a_1	b_2	c_1	d_2	B
3	a_2	b_3	c_2	d_3	A
4	a_1	b_2	c_3	d_3	C
5	a_1	b_2	c_1	d_3	C

Table 1. A training data set.

First, *CMAR* scans the training data set T once, find the set of attribute values happening at least twice in T. The set is $F = \{a_1, b_2, c_1, d_3\}$ and is called *frequent item set*. All other attribute values, which fail the support threshold, cannot play any role in the class-association rules, and thus can be pruned.

Then, *CMAR* sorts attribute values in F in support descending order, i.e., *F-list* $= a_1 - b_2 - c_1 - d_3$. Then, *CMAR* scans the training data set again to construct an *FP-tree*, as shown in Figure 1(a).

FP-tree is a prefix tree w.r.t. *F-list*. For each tuple in the training data set, attributes values appearing in *F-list* are extracted and sorted according to *F-list*. For example, for the first tuple, (a_1, c_1) are extracted and inserted in the tree as the left-most branch in the tree. the class label is attached to the last node in the path.

Tuples in the training data set share prefixes. For example, the second tuple carries attribute values (a_1, b_2, c_1) in *F-list* and shares a common prefix a_1 with the first tuple. So, it also shares the a_1 sub-path with the left-most branch.

All nodes with same attribute value are linked together as a queue started from the *header table*.

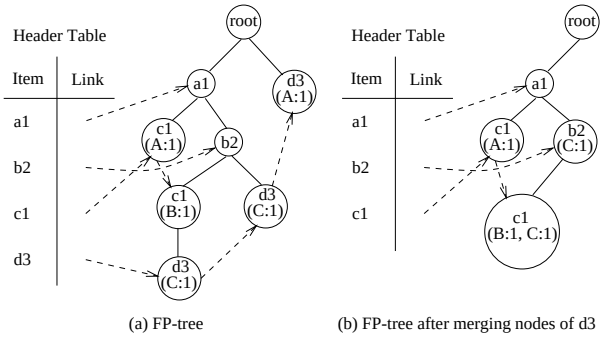

Figure 1. *FP-tree* **in Example 1.**

Third, based on *F-list*, the set of class-association rules can be divided into 4 subsets without overlap: (1) the ones having d_3; (2) the ones having c_1 but no d_3; (3) the ones having b_2 but no d_3 nor c_1; and (4) the ones having only a_1. *CMAR* finds these subsets one by one.

Fourth, to find the subset of rules having d_3, *CMAR* traverses nodes having attribute value d_3 and look "upward" to collect a d_3-projected database, which contains three tuples: $(a_1, b_2, c_1, d_3) : C$, $(a_1, b_2, d_3) : C$ and $d_3 : A$. It contains all the tuples having d_3. The problem of finding all frequent patterns having d_3 in the whole training set can be reduced to mine frequent patterns in d_3-projected database.

Recursively, in d_3-projected database, a_1 and b_2 are the frequent attribute values, i.e., they pass support threshold. (In d_3-projected database, d_3 happens in every tuple and thus is trivially frequent. We do not count d_3 as a local frequent attribute value.) We can mine the projected database recursively by constructing *FP-tree*s and projected databases. Please see [5] for more details.

It happens that, in d_3-projected database, a_1 and b_2 always happen together and thus a_1b_2 is a frequent pattern. a_1 and b_2 are two subpatterns of a_1b_2 and have same support count as a_1b_2. To avoid triviality, we only adopt frequent pattern $a_1b_2d_3$. Based on the class label distribution information, we generate rule $a_1b_2d_3 \rightarrow C$ with support 2 and confidence 100%.

After search for rules having d_3, all nodes of d_3 are merged into their parent nodes, respectively. That is, the class label information registered in a d_3 node is registered in its parent node. The *FP-tree* is shrunk as shown in Figure 1(b). Please note that this tree-shrinking operation is done at the same scan of collecting the d_3-projected database.

The remaining subsets of rules can be mined similarly.

There are two major differences in the rule mining in *CMAR* and the standard *FP-growth* algorithm.

On one hand, CMAR *finds frequent patterns and generates rules in one step.*

Conventionally, association rules must be mined in two steps [1]. This is also the case for traditional associative classification methods [9]. First, all the frequent patterns (i.e., patterns passing support threshold) are found. Then, all the association rules satisfying the confidence threshold are generated based on the mined frequent patterns.

The difference of *CMAR* from other associative classification methods is that for every pattern, *CMAR* maintains the distribution of various class labels among data objects matching the pattern. This is done without any overhead in the procedure of counting (conditional) databases. Thus, once a frequent pattern (i.e., pattern passing support threshold) is found, rules about the pattern can be generated immediately. Therefore, *CMAR* has no separated rule generation step.

On the other hand, CMAR *uses class label distribution to prune.*

For any frequent pattern P, let c be the most dominant class in the set of data objects matching P. If the number of objects having class label c and matching P is less than the support threshold, there is no need to search any super-pattern (superset) P' of P since any rule in the form of $P' \rightarrow c$ cannot satisfy the support threshold either.

3.2 Storing Rules in *CR-tree*

Once a rule is generated, it is stored in a *CR-tree*, which is a prefix tree structure. We demonstrate the general idea of *CR-tree* in the following example.

Example 2 (*CR-tree*) After mining a training data set, four rules are found as shown in Table 2.

Rule-id	Rule	Support	Confidence
1	$abc \rightarrow A$	80	80%
2	$abcd \rightarrow A$	63	90%
3	$abe \rightarrow B$	36	60%
4	$bcd \rightarrow C$	210	70%

Table 2. Rules found in a training data set.

A *CR-tree* is built for the set of rules, as shown in Figure 2, while the construction process is explained as follows.

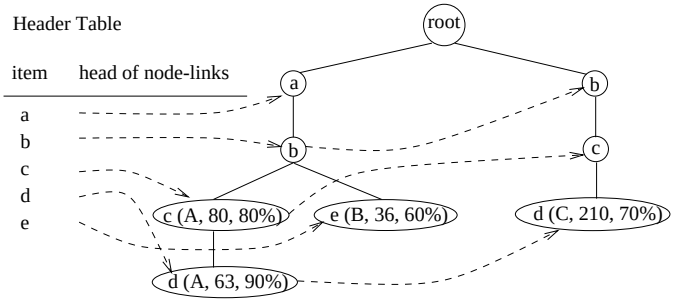

Figure 2. A *CR-tree* for rules in Example 2.

A *CR-tree* has a *root node*. All the attribute values appearing at the left hand side of rules are sorted according to their frequency, i.e., the most frequently appearing attribute value goes first.

The first rule, $abc \rightarrow A$ is inserted into the tree as a path from root node. The class label as well as the support and confidence of the rule, denoted as $(A, 80, 80\%)$, are registered at the last node in the path, i.e., node c for this rule.

The second rule, $abcd \rightarrow A$, shares a *prefix* abc with the first rule. Thus, it is inserted into the tree by extending a new node d to the path formed by the first rule. Again, the class label, support and confidence of the rule are registered at the last node, i.e., d.

The third and fourth rules can be inserted similarly. All the nodes with the same attribute value are linked together by *node-link* to a queue. The head of each queue is stored in a *Header table.*

To store the original rule set, 13 cells are needed for the left hand sides of the rules. Using *CR-tree*, only 9 nodes are needed.

As can be seen from the above example, the *CR-tree* structure has some advantages as follows.

CR-tree is a compact structure. It explores potential sharing among rules and thus can save a lot of space on storing rules. Our experimental results show that, in many cases, about $50-60\%$ of space can be saved using *CR-tree*.

CR-tree itself is an index for rules. For example, if we want to retrieve all the rules having attribute value b and d in the set of rules in Example 2, we only need to traverse node-links of d, which starts at the header table, and keep looking upward for b.

Once a *CR-tree* is built, rule retrieval becomes efficient. That facilitates the pruning of rules and using rules for classification dramatically.

3.3 Pruning Rules

The number of rules generated by class-association rule mining can be huge. To make the classification effective and also efficient, we need to prune rules to delete redundant and noisy information.

According to the facility of rules on classification, a global order of rules is composed. Given two rules R_1 and R_2, R_1 is said **having higher rank** than R_2, denoted as $R_1 > R_2$, if and only if (1) $conf(R_1) > conf(R_2)$; (2) $conf(R_1) = conf(R_2)$ but $sup(R_1) > sup(R_2)$; or (3) $conf(R_1) = conf(R_2)$, $sup(R_1) = sup(R_2)$ but R_1 has fewer attribute values in its left hand side than R_2 does. In addition, a rule $R_1 : P \rightarrow c$ is said a **general rule** w.r.t. rule $R_2 : P' \rightarrow c'$, if and only if P is a subset of P'.

CMAR employs the following methods for rule pruning.

First, *using general and high-confidence rule to prune more specific and lower confidence ones*. Given two rules R_1 and R_2, where R_1 is a general rule w.r.t. R_2. *CMAR* prunes R_2 if R_1 also has higher rank than R_2. The rationale is that we only need to consider general rules with high confidence, and thus more specific rules with low confidence should be pruned.

This pruning is pursued when the rule is inserted into the *CR-tree*. When a rule is inserted into the tree, retrieval over the tree is triggered to check if the rule can be pruned or it can prune other rules that are already inserted. Our experimental results show that this pruning is effective.

Second, *selecting only positively correlated rules.* For each rule $R : P \rightarrow c$, we test whether P is positively correlated with c by χ^2 testing. Only the rules that are positively correlated, i.e., those with χ^2 value passing a significance level threshold, are used for later classification. All the other rules are pruned.

Algorithm 1 (Selecting rules based on database coverage)

Input: a set of rules and a coverage threshold δ

Output: a subset of rules for classification

Method:

1. Sort rules in the rank descending order;
2. For each data object in the training data set, set its cover-count to 0;
3. While both the training data set and rule set are not empty, for each rule R in rank descending order, find all data objects matching rule R. If R can correctly classify at least one object then select R and increase the cover-count of those objects matching R by 1. A data object is removed if its cover-count passes coverage threshold δ;

Figure 3. Selecting rules based on database coverage.

The rationale of this pruning is that we use the rules reflecting strong implications to do classification. By removing those rules not positively correlated, we prune noise.

This pruning happens when a rule is found. Since the distribution of class labels w.r.t. frequent patterns is kept track during the rule mining, the χ^2 testing is done almost for free.

Third, *pruning rules based on database coverage. CMAR* selects a subset of high quality rules for classification. That is achieved by pruning rules based on database coverage. *CMAR* uses a *coverage threshold* [7] to select database coverage, as shown in Figure 3.

The database coverage method used in *CMAR* is similar to that in CBA. The major difference is that, instead of removing one data object from the training data set immediately after it is covered by some selected rule, we let it stay there until it is covered by at least δ rules. That allows more selected rules. When classifying a new data object, it may have more rules to consult and may have better chance to be accurately predicted.

This pruning is pursued when the rule mining process finishes. It is the last pruning of rules.

4 Classification Based on Multiple Rules

After a set of rules is selected for classification, as discussed in Section 3, *CMAR* is ready to classify new objects. Given a new data object, *CMAR* collects the subset of rules matching the new object from the set of rules for classification. In this section, we discuss how to determine the class label based on the subset of rules.

Trivially, if all the rules matching the new object have the same class label, *CMAR* just simply assigns that label

to the new object.

If the rules are not consistent in class labels, *CMAR* divides the rules into groups according to class labels. All rules in a group share the same class label and each group has a distinct label. *CMAR* compares the effects of the groups and yields to the strongest group.

To compare the strength of groups, we need to measure the "*combined effect*" of each group. Intuitively, if the rules in a group are highly positively correlated and have good support, the group should have strong effect.

There are many possible ways to measure the combined effect of a group of rules. For example, one can use the strongest rule as a representative. That is, the rule with highest χ^2 value is selected. However, simply choosing the rule with highest χ^2 value may be favorable to minority classes, as illustrated in the following example.

Example 3 In a credit card application approval case, there are two rules.

$R_1 : job = no \rightarrow rejected(support = , confidence = 60\%)$, and

$R_2 : education = university \rightarrow approved(sup = 200, confidence = 99.5\%)$

The observed and expected contingencies are shown in Figure 4.

R_1	approved	rejected	total
job=yes	438	32	470
job=no	12	18	30
total	450	50	500

The observed contingency of rule R_1.

R_2	approved	rejected	total
ed=univ	199	1	200
ed≠univ	251	49	300
total	450	50	500

The observed contingency of rule R_2.

R_1	approved	rejected	total
job=yes	423	47	470
job=no	27	3	30
total	450	50	500

The expected contingency of rule R_1.

R_2	approved	rejected	total
ed=univ	180	20	200
ed≠univ	270	30	300
total	450	50	500

The expected contingency of rule R_2.

Figure 4. Observed and expected contingencies for rules.

Based on the observed and expected values, the χ^2 values for R_1 and R_2 are 88.4 and 33.6, respectively. For a customer having no job and with university education, we may predict her application would be rejected using rule R_1, if the choice of rules is based on only χ^2 values.

However, rule R_2 is intuitively much better than R_1 since R_2 has much higher support and confidence.

Another alternative is to use the compound of correlation of rules as measure. For example, we can sum up χ^2 values in a group as the measure of the group. However, this method suffers from the same problem that it may favors minority too much.

A better way is to integrate both information of correlation and popularity into the measure. After empirical verification, *CMAR* adopts a *weighted* χ^2 measure [7] as follows.

For each rule $R : P \rightarrow c$, let $sup(c)$ be the number of data objects in training data set that associated with class label c and $|T|$ the number of data objects in the training data set. We define $max\chi^2$ for rule R as follows.

$$max\chi^2 = (min\{sup(P), sup(c)\} - \frac{sup(P)sup(c)}{|T|})^2 |T| e$$

where

$$e = \frac{1}{sup(P)sup(c)} + \frac{1}{sup(P)(|T| - sup(c))} + \frac{1}{(|T| - sup(P))sup(c)} + \frac{1}{(|T| - sup(P))(|T| - sup(c))}$$

$max\chi^2$ computes the upper bound of χ^2 value of the rule w.r.t. other settings are fixed. Then, for each group of rules, the *weighted* χ^2 measure of the group is defined as $\sum \frac{\chi^2 \chi^2}{max\chi^2}$.

As can be seen, we use the ratio of χ^2 value against its upper bound (i.e., $max\chi^2$) to overcome the bias of χ^2 value favoring minority class. Please note that, theoretically, it is hard to verify the soundness or effect of measures on strength of groups of rules. Instead, we explore the effect of measures empirically, and according to our experimental results, weighted χ^2 value is the best from among a good set of candidate measure formulas that can be worked out by us.

5 Experimental Results and Performance Study

To evaluate the accuracy, efficiency and scalability of *CMAR*, we have performed an extensive performance study. In this section, we report our experimental results on comparing *CMAR* against two popular classification methods: CBA [9] and C4.5 [10]. It shows that *CMAR* outperforms both CBA and C4.5 in terms of average accuracy, efficiency and scalability.

All the experiments are performed on a 600MHz Pentium PC with 128M main memory, running Microsoft Windows/NT. CBA and C4.5 were implemented by their

authors, respectively. In the experiments, the parameters of the three methods are set as follows.

All C4.5 parameters are default values. We test both C4.5 decision tree method and rule method. Since the rule method has better accuracy, we only report the accuracy for rule method.

For CBA, we set support threshold to 1% and confidence threshold to 50% and disable the limit on number of rules. Other parameters remain default.

For *CMAR*, the support and confidence thresholds are set as same as CBA. The database coverage threshold is set to 4 and the confidence difference threshold to 20%.

All reports of the runtime include both CPU time and I/O time.

We test 26 data sets from UCI Machine Learning Repository. We use C4.5's *shuffle* utility to shuffle the data sets. Also, we adopt the same method used by CBA to discretize continuous attributes.

Data set	# attr	# cls	# rec	C4.5	CBA	*CMAR*
Anneal	38	6	898	94.8	97.9	97.3
Austral	14	2	690	84.7	84.9	86.1
Auto	25	7	205	80.1	78.3	78.1
Breast	10	2	699	95	96.3	96.4
Cleve	13	2	303	78.2	82.8	82.2
Crx	15	2	690	84.9	84.7	84.9
Diabetes	8	2	768	74.2	74.5	75.8
German	20	2	1000	72.3	73.4	74.9
Glass	9	7	214	68.7	73.9	70.1
Heart	13	2	270	80.8	81.9	82.2
Hepatic	19	2	155	80.6	81.8	80.5
Horse	22	2	368	82.6	82.1	82.6
Hypo	25	2	3163	99.2	98.9	98.4
Iono	34	2	351	90	92.3	91.5
Iris	4	3	150	95.3	94.7	94
Labor	16	2	57	79.3	86.3	89.7
Led7	7	10	3200	73.5	71.9	72.5
Lymph	18	4	148	73.5	77.8	83.1
Pima	8	2	768	75.5	72.9	75.1
Sick	29	2	2800	98.5	97	97.5
Sonar	60	2	208	70.2	77.5	79.4
Tic-tac	9	2	958	99.4	99.6	99.2
Vehicle	18	4	846	72.6	68.7	68.8
Waveform	21	3	5000	78.1	80	83.2
Wine	13	3	178	92.7	95	95
Zoo	16	7	101	92.2	96.8	97.1
Average				83.34	84.69	85.22

Table 3. The comparison of C4.5, CBA and *CMAR* on accuracy.

As can be seen from the table, *CMAR* outperforms both C4.5 and CBA on accuracy. Furthermore, out of the 26 data sets, *CMAR* achieves the best accuracy in 13 ones. In another word, *CMAR* wins in 50% of test data sets. In some data sets, e.g. Lymph, *CMAR* wins the second place over 5% in accuracy.

There are two important parameters, *database coverage threshold* and *confidence difference threshold*, in *CMAR*. As discussed before, these two thresholds control the number of rules selected for classification.

In general, if the set of rules is too small, some effective rules may be missed. On the other hand, if the rule set is too large, the training data set may be overfit. Thus, we need to test the sensitivities of the two thresholds w.r.t. classification accuracy.

As an example, we test different database coverage threshold values on the *Sonar* data set from UCI Machine Learning Database Repository. The results are shown in Figure 5, where the confidence difference threshold is set to 0. On the other hand, we test different confidence difference threshold values on the *Sonar* data set. The results are shown in Figure 6, where the database coverage threshold is set to 1.

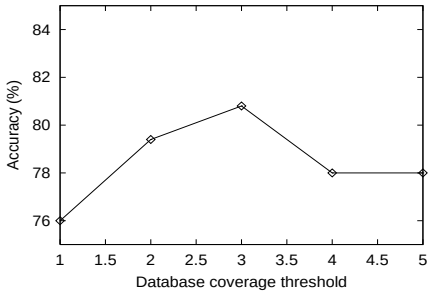

Figure 5. The effect of coverage threshold on accuracy.

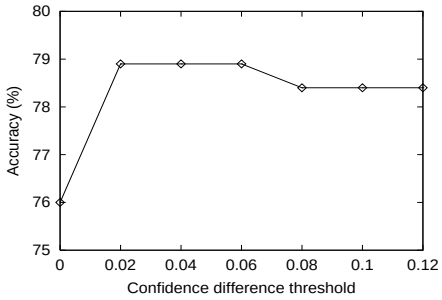

Figure 6. The effect of confidence difference threshold on accuracy.

From the figures, one can see that the peaks of accuracy are achieved at the middle of both curves. That is, there are optimal settings for both thresholds. However, according to our experimental results, there seems no way to pre-determine the best threshold values. Fortunately, both curves are quite plain. That means the accuracy is not very sensitive to the two thresholds values.

CR-tree is a compact structure to store rules. To test

the effectiveness of *CR-tree*, we compare the main memory usage of CBA and *CMAR* on large test data sets. The results are shown in Table 4.

Dataset	# attr	# cls	# rec	CBA mem (M)	*CMAR* mem (M)	saving (%)
Auto	25	7	205	488	101	79.30%
Hypo	25	2	3163	330	90	72.73%
Iono	34	2	351	334	86	74.25%
Sick	29	2	2800	321	85	73.52%
Sonar	60	2	208	590	88	85.09%
Vehicle	18	4	846	590	88	85.09%
Average				393.33	90	77.12%

Table 4. The comparison of CBA and *CMAR* on main memory usage.

Please note that, in this experiment, we disable the limitation of number of rules in CBA. In such a setting, CBA and *CMAR* generate all the rules necessary for classification and thus are compared in a fair base. From the table, one can see that, on average, *CMAR* achieves 77.12% saving on main memory usage.

The saving in main memory usage can be explained from two apsects.

First, *CMAR* uses *CR-tree*. The compactness of *CR-tree* brings significant gain in storing a large set of rules where many items in the rules can be shared.

On the other hand, *CR-tree* is also an index structure of rules. Before a rule is inserted into a *CR-tree*, *CMAR* checks if there is a general rule or some more specific rules in the tree. If so, related pruning is pursued immediately. Such a pruning techique also contributes to the saving of main memory.

To test the scalability of *CMAR*, we compare the runtime of CBA and *CMAR* on six data sets. The results are shown in Figure 5. Again, we disable the limit on number of rules in CBA. In the experiments, CBA spends a large portion of runtime on I/O.

Dataset	# attr	# cls	# rec	CBA runtime	*CMAR* runtime
Auto	25	7	205	612s	408s
Hypo	25	2	3163	92s	19s
Iono	34	2	351	150s	89s
Sick	29	2	2800	74s	13s
Sonar	60	2	208	226s	145s
Vehicle	18	4	846	284s	192s

Table 5. The runtime of CBA and *CMAR*.

As can be seen from the table, *CMAR* is faster than CBA in many cases. Please be note that the machine we use for testing is with relatively small size of main memory (128M). Both CBA and *CMAR* can be expected running significantly faster if more main memory is available.

6 Conclusions

In this paper, we examined two major challenges in associative classification: (1) efficiency at handling huge number of mined association rules, and (2) effectiveness at predicting new class labels with high classification accuracy. We proposed a novel associative classification method, *CMAR*, i.e., **C**lassification based on **M**ultiple **A**ssociation **R**ules. The method has several distinguished features: (1) its classification is performed based on a weighted χ^2 analysis enforced on multiple association rules, which leads to better overall classification accuracy, (2) it prunes rules effectively based on confidence, correlation and database coverage, and (3) its efficiency is achieved by extension of an efficient frequent pattern mining method, *FP-growth*, construction of a class distribution-associated *FP-tree*, and applying a *CR-tree* structure to store and retrieve mined association rules efficiently.

Our experiments on 26 databases in UCI machine learning database repository show that *CMAR* is consistent, highly effective at classification of various kinds of databases and has better average classification accuracy in comparison with CBA and C4.5, and is more efficient and scalable than other associative classification methods.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB'94*, Chile, Sept. 1994.

[2] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[3] G. Dong, X. Zhang, L. Wong, and J. Li. Caep: Classification by aggregating emerging patterns. In *DS'99 (LNCS 1721)*, Japan, Dec. 1999.

[4] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[5] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD'00*, Dallas, TX, May 2000.

[6] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *ICDE'97*, England, April 1997.

[7] W. Li. Classification based on multiple association rules. *M.Sc. Thesis*, Simon Fraser University, April 2001.

[8] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 39, 2000.

[9] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD'98*, New York, NY, Aug. 1998.

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

[11] K. Wang, S. Zhou, and Y. He. Growing decision tree on support-less association rules. In *KDD'00*, Boston, MA, Aug. 2000.