Appeared in *KDD-98*, New York, Aug 27-31, 1998

Integrating Classification and Association Rule Mining

Bing Liu Wynne Hsu Yiming Ma

Department of Information Systems and Computer Science
National University of Singapore
Lower Kent Ridge Road, Singapore 119260
{liub, whsu, mayiming}@iscs.nus.edu.sg

Abstract

Classification rule mining aims to discover a small set of rules in the database that forms an accurate classifier. Association rule mining finds all the rules existing in the database that satisfy some minimum support and minimum confidence constraints. For association rule mining, the target of discovery is not pre-determined, while for classification rule mining there is one and only one pre-determined target. In this paper, we propose to integrate these two mining techniques. The integration is done by focusing on mining a special subset of association rules, called *class association rules* (CARs). An efficient algorithm is also given for building a classifier based on the set of discovered CARs. Experimental results show that the classifier built this way is, in general, more accurate than that produced by the state-of-the-art classification system C4.5. In addition, this integration helps to solve a number of problems that exist in the current classification systems.

Introduction

Classification rule mining and association rule mining are two important data mining techniques. Classification rule mining aims to discover a small set of rules in the database to form an accurate classifier (e.g., Quinlan 1992; Breiman et al 1984). Association rule mining finds all rules in the database that satisfy some minimum support and minimum confidence constraints (e.g., Agrawal and Srikant 1994). For association rule mining, the target of mining is not pre-determined, while for classification rule mining there is one and only one pre-determined target, i.e., the class. Both classification rule mining and association rule mining are indispensable to practical applications. Thus, great savings and conveniences to the user could result if the two mining techniques can somehow be integrated. In this paper, we propose such an integrated framework, called *associative classification*. We show that the integration can be done efficiently and without loss of performance, i.e., the accuracy of the resultant classifier.

The integration is done by focusing on a special subset of association rules whose right-hand-side are restricted to the classification class attribute. We refer to this subset of rules as the *class association rules* (CARs). An existing association rule mining algorithm (Agrawal and Srikant 1994) is adapted to mine all the CARs that satisfy the minimum support and minimum confidence constraints. This adaptation is necessary for two main reasons:

1. Unlike a transactional database normally used in association rule mining (Agrawal and Srikant 1994) that does not have many associations, classification data tends to contain a huge number of associations. Adaptation of the existing association rule mining algorithm to mine only the CARs is needed so as to reduce the number of rules generated, thus avoiding combinatorial explosion (see the evaluation section).
2. Classification datasets often contain many continuous (or numeric) attributes. Mining of association rules with continuous attributes is still a major research issue (Srikant and Agrawal 1996; Yoda et al 1997; Wang, Tay and Liu 1998). Our adaptation involves discretizing continuous attributes based on the classification pre-determined class target. There are many good discretization algorithms for this purpose (Fayyad and Irani 1993; Dougherty, Kohavi and Sahami 1995).

Data mining in the proposed associative classification framework thus consists of three steps:

- discretizing continuous attributes, if any
- generating all the class association rules (CARs), and
- building a classifier based on the generated CARs.

This work makes the following contributions:

1. It proposes a new way to build accurate classifiers. Experimental results show that classifiers built this way are, in general, more accurate than those produced by the state-of-the-art classification system C4.5 (Quinlan 1992).
2. It makes association rule mining techniques applicable to classification tasks.
3. It helps to solve a number of important problems with the existing classification systems.

Let us discuss point 3 in greater detail below:

- The framework helps to solve the *understandability* problem (Clark and Matwin 1993; Pazzani, Mani and Shankle 1997) in classification rule mining. Many rules produced by standard classification systems are difficult to understand because these systems use domain independent biases and heuristics to generate a small set of rules to form a classifier. These biases, however, may not be in agreement with the existing knowledge of the human user, thus resulting in many generated rules that make no sense to the user, while many understandable rules that exist in the data are left undiscovered. With the new framework, the problem of finding understandable rules is reduced to a post-processing task (since we generate all the rules). Techniques such as those in (Liu and Hsu 1996; Liu, Hsu and Chen 1997) can be employed to help the user identify understandable rules.

- A related problem is the discovery of *interesting* or *useful* rules. The quest for a small set of rules of the existing classification systems results in many interesting and useful rules not being discovered. For example, in a drug screening application, the biologists are very interested in rules that relate the color of a sample to its final outcome. Unfortunately, the classification system (we used C4.5) just could not find such rules even though such rules do exist as discovered by our system.
- In the new framework, the database can reside on disk rather than in the main memory. Standard classification systems need to load the entire database into the main memory (e.g., Quinlan 1992), although some work has been done on the scaling up of classification systems (Mahta, Agrawal and Rissanen 1996).

Problem Statement

Our proposed framework assumes that the dataset is a normal relational table, which consists of N cases described by l distinct attributes. These N cases have been classified into q known classes. An attribute can be a categorical (or discrete) or a continuous (or numeric) attribute.

In this work, we treat all the attributes uniformly. For a categorical attribute, all the possible values are mapped to a set of consecutive positive integers. For a continuous attribute, its value range is discretized into intervals, and the intervals are also mapped to consecutive positive integers. With these mappings, we can treat a data case as a set of (*attribute*, *integer-value*) pairs and a class label. We call each (*attribute*, *integer-value*) pair an *item*. Discretization of continuous attributes will not be discussed in this paper as there are many existing algorithms in the machine learning literature that can be used (see (Dougherty, Kohavi and Sahami 1995)).

Let D be the dataset. Let I be the set of all items in D, and Y be the set of class labels. We say that a data case $d \in D$ contains $X \subseteq I$, a subset of items, if $X \subseteq d$. A class association rule (CAR) is an implication of the form $X \rightarrow y$, where $X \subseteq I$, and $y \in Y$. A rule $X \rightarrow y$ holds in D with confidence c if c% of cases in D that contain X are labeled with class y. The rule $X \rightarrow y$ has support s in D if s% of the cases in D contain X and are labeled with class y.

Our objectives are (1) to generate the complete set of CARs that satisfy the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) constraints, and (2) to build a classifier from the CARs.

Generating the Complete Set of CARs

The proposed algorithm is called algorithm CBA (Classification Based on Associations). It consists of two parts, a *rule generator* (called CBA-RG), which is based on algorithm Apriori for finding association rules in (Agrawal and Srikant 1994), and a *classifier builder* (called CBA-CB). This section discusses CBA-RG. The next section discusses CBA-CB.

Basic concepts used in the CBA-RG algorithm

The key operation of CBA-RG is to find all *ruleitems* that have support above *minsup*. A *ruleitem* is of the form:

$<condset, y>$

where *condset* is a set of items, $y \in Y$ is a class label. The support count of the *condset* (called *condsupCount*) is the number of cases in D that contain the *condset*. The support count of the *ruleitem* (called *rulesupCount*) is the number of cases in D that contain the *condset* and are labeled with class y. Each *ruleitem* basically represents a rule:

$condset \rightarrow y$,

whose *support* is (*rulesupCount* / $|D|$) *100%, where $|D|$ is the size of the dataset, and whose *confidence* is (*rulesupCount / condsupCount*)*100%.

Ruleitems that satisfy *minsup* are called *frequent ruleitems*, while the rest are called *infrequent ruleitems*. For example, the following is a *ruleitem*:

$<\{(A, 1), (B, 1)\}, (class, 1)>$,

where A and B are attributes. If the support count of the *condset* {(A, 1), (B, 1)} is 3, the support count of the *ruleitem* is 2, and the total number of cases in D is 10, then the support of the *ruleitem* is 20%, and the confidence is 66.7%. If *minsup* is 10%, then the *ruleitem* satisfies the *minsup* criterion. We say it is frequent.

For all the *ruleitems* that have the same *condset*, the *ruleitem* with the highest confidence is chosen as the *possible rule* (PR) representing this set of *ruleitems*. If there are more than one *ruleitem* with the same highest confidence, we randomly select one *ruleitem*. For example, we have two *ruleitems* that have the same *condset*:

1. $<\{(A, 1), (B, 1)\}, (class: 1)>$.
2. $<\{(A, 1), (B, 1)\}, (class: 2)>$.

Assume the support count of the *condset* is 3. The support count of the first *ruleitem* is 2, and the second *ruleitem* is 1. Then, the confidence of *ruleitem* 1 is 66.7%, while the confidence of *ruleitem* 2 is 33.3% With these two *ruleitems*, we only produce one PR (assume $|D| = 10$):

(A, 1), (B, 1) $\rightarrow$ (class, 1) [*supt* = 20%, *confd*= 66.7%]

If the confidence is greater than *minconf*, we say the rule is *accurate*. The set of *class association rules* (CARs) thus consists of all the PRs that are both frequent and accurate.

The CBA-RG algorithm

The CBA-RG algorithm generates all the frequent *ruleitems* by making multiple passes over the data. In the first pass, it counts the support of individual *ruleitem* and determines whether it is frequent. In each subsequent pass, it starts with the seed set of *ruleitems* found to be frequent in the previous pass. It uses this seed set to generate new possibly frequent *ruleitems*, called *candidate ruleitems*. The actual supports for these candidate *ruleitems* are calculated during the pass over the data. At the end of the pass, it determines which of the candidate *ruleitems* are actually frequent. From this set of frequent *ruleitems*, it produces the rules (CARs).

Let *k-ruleitem* denote a *ruleitem* whose *condset* has k items. Let F_k denote the set of frequent *k-ruleitems*. Each element of this set is of the following form:

$$<(condset, condsupCount), (y, rulesupCount)>.$$

Let C_k be the set of candidate *k-ruleitems*. The CBA-RG algorithm is given in Figure 1.

```
1   F_1 = {large 1-ruleitems};
2   CAR_1 = genRules(F_1);
3   prCAR_1 = pruneRules(CAR_1);
4   for (k = 2; F_k-1 ≠ ∅; k++) do
5       C_k = candidateGen(F_k-1);
6       for each data case d ∈ D do
7           C_d = ruleSubset(C_k, d);
8           for each candidate c ∈ C_d do
9               c.condsupCount++;
10              if d.class = c.class then c.rulesupCount++
11          end
12      end
13      F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
14      CAR_k = genRules(F_k);
15      prCAR_k = pruneRules(CAR_k);
16  end
17  CARs = ∪_k CAR_k;
18  prCARs = ∪_k prCAR_k;
```

Figure 1: The CBA-RG algorithm

Line 1-3 represents the first pass of the algorithm. It counts the item and class occurrences to determine the frequent 1-*ruleitems* (line 1). From this set of 1-*ruleitems*, a set of CARs (called CAR_1) is generated by *genRules* (line 2) (see previous subsection). CAR_1 is subjected to a pruning operation (line 3) (which can be optional). Pruning is also done in each subsequent pass to CAR_k (line 15). The function *pruneRules* uses the pessimistic error rate based pruning method in C4.5 (Quinlan 1992). It prunes a rule as follows: If rule r's pessimistic error rate is higher than the pessimistic error rate of rule r^- (obtained by deleting one condition from the conditions of r), then rule r is pruned. This pruning can cut down the number of rules generated substantially (see the evaluation section).

For each subsequent pass, say pass k, the algorithm performs 4 major operations. First, the frequent *ruleitems* $F_{k\text{-}1}$ found in the (k-1)*th* pass are used to generate the candidate *ruleitems* C_k using the *condidateGen* function (line 5). It then scans the database and updates various support counts of the candidates in C_k (line 6-12). After those new frequent *ruleitems* have been identified to form F_k (line 13), the algorithm then produces the rules CAR_k using the *genRules* function (line 14). Finally, rule pruning is performed (line 15) on these rules.

The *candidateGen* function is similar to the function *Apriori-gen* in algorithm Apriori. The *ruleSubset* function takes a set of candidate *ruleitems* C_k and a data case d to find all the *ruleitems* in C_k whose *condsets* are supported by d. This and the operations in line 8-10 are also similar to those in algorithm Apriori. The difference is that we need to increment the support counts of the *condset* and the *ruleitem* separately whereas in algorithm Apriori only one count is updated. This allows us to compute the confidence of the *ruleitem*. They are also useful in rule pruning.

The final set of class association rules is in *CARs* (line 17). Those remaining rules after pruning are in *prCARs* (line 18).

Building a Classifier

This section presents the CBA-CB algorithm for building a classifier using CARs (or prCARs). To produce the best classifier out of the whole set of rules would involve evaluating all the possible subsets of it on the training data and selecting the subset with the right rule sequence that gives the least number of errors. There are 2^m such subsets, where m is the number of rules, which can be more than 10,000, not to mention different rule sequences. This is clearly infeasible. Our proposed algorithm is a heuristic one. However, the classifier it builds performs very well as compared to that built by C4.5. Before presenting the algorithm, let us define a total order on the generated rules. This is used in selecting the rules for our classifier.

Definition: Given two rules, r_i and r_j, $r_i \succ r_j$ (also called r_i precedes r_j or r_i has a higher precedence than r_j) if

1. the confidence of r_i is greater than that of r_j, or
2. their confidences are the same, but the support of r_i is greater than that of r_j, or
3. both the confidences and supports of r_i and r_j are the same, but r_i is generated earlier than r_j;

Let R be the set of generated rules (i.e., CARs or pCARs), and D the training data. The basic idea of the algorithm is to choose a set of high precedence rules in R to cover D. Our classifier is of the following format:

$$<r_1, r_2, \ldots, r_n, default\_class>,$$

where $r_i \in R$, $r_a \succ r_b$ if $b > a$. *default_class* is the default class. In classifying an unseen case, the first rule that satisfies the case will classify it. If there is no rule that applies to the case, it takes on the default class as in C4.5. A naive version of our algorithm (called M1) for building such a classifier is shown in Figure 2. It has three steps:

Step 1 (line 1): Sort the set of generated rules R according to the relation "$\succ$". This is to ensure that we will choose the highest precedence rules for our classifier.

Step 2 (line 2-13): Select rules for the classifier from R following the sorted sequence. For each rule r, we go through D to find those cases *covered* by r (they satisfy the conditions of r) (line 5). We mark r if it correctly classifies a case d (line 6). d.id is the unique identification number of d. If r can correctly classify at least one case (i.e., if r is marked), it will be a potential rule in our classifier (line 7-8). Those cases it covers are then removed from D (line 9). A default class is also selected (the majority class in the remaining data), which means that if we stop selecting more rules for our classifier C this class will be the default class of C (line 10). We then compute and record the total number of errors that are made by the current C and the default class (line 11). This is the sum of the number of errors that have been made by all the selected rules in C and the number of errors to be made by the default class in the training data. When there is no rule or no training case left, the rule selection process is completed.

Step 3 (line 14-15): Discard those rules in C that do not improve the accuracy of the classifier. The first rule at which there is the least number of errors recorded on D is the cutoff rule. All the rules after this rule can be discarded because they only produce more errors. The undiscarded rules and the default class of the last rule in C form our classifier.

```
1   R = sort(R);
2   for each rule r ∈ R in sequence do
3       temp = ∅;
4       for each case d ∈ D do
5           if d satisfies the conditions of r then
6               store d.id in temp and mark r if it correctly
                    classifies d;
7       if r is marked then
8           insert r at the end of C;
9           delete all the cases with the ids in temp from D;
10          selecting a default class for the current C;
11          compute the total number of errors of C;
12      end
13  end
14  Find the first rule p in C with the lowest total number
        of errors and drop all the rules after p in C;
15  Add the default class associated with p to end of C,
        and return C (our classifier).
```

Figure 2. A naïve algorithm for CBA-CB: M1

This algorithm satisfies two main conditions:

Condition 1. Each training case is covered by the rule with the highest precedence among the rules that can cover the case. This is so because of the sorting done in line 1.

Condition 2. Every rule in C correctly classifies at least one remaining training case when it is chosen. This is so due to line 5-7.

This algorithm is simple, but is inefficient especially when the database is not resident in the main memory because it needs to make many passes over the database. Below, we present an improved version of the algorithm (called M2), whereby only slightly more than one pass is made over D. The key point is that instead of making one pass over the remaining data for each rule (in M1), we now find the best rule in R to cover each case. M2 consists of three stages (see (Liu, Hsu and Ma 1998) for more details):

Stage 1. For each case d, we find both the highest precedence rule (called *cRule*) that correctly classifies d, and also the highest precedence rule (called *wRule*) that wrongly classifies d. If *cRule* $\succ$ *wRule*, the case should be covered by *cRule*. This satisfies Condition 1 and 2 above. We also mark the *cRule* to indicate that it classifies a case correctly. If *wRule* $\succ$ *cRule*, it is more complex because we cannot decide which rule among the two or some other rule will eventually cover d. In order to decide this, for each d with *wRule* $\succ$ *cRule*, we keep a data structure of the form: <*dID*, y, *cRule*, *wRule*>, where *dID* is the unique identification number of the case d, y is the class of d. Let A denote the collection of <*dID*, y, *cRule*, *wRule*>'s, U the set of all *cRule*s, and Q the set of *cRule*s that have a higher precedence than their corresponding *wRule*s. This stage of the algorithm is shown in Figure 3.

The *maxCoverRule* function finds the highest precedence rule that covers the case d. C_c (or C_w) is the set of rules having the same (or different) class as d. d.id and d.class represent the identification number and the class of d respectively. For each *cRule*, we also remember how many cases it covers in each class using the field *classCasesCovered* of the rule.

```
1   Q = ∅; U = ∅; A = ∅;
2   for each case d ∈ D do
3       cRule = maxCoverRule(C_c, d);
4       wRule = maxCoverRule(C_w, d);
5       U = U ∪ {cRule};
6       cRule.classCasesCovered[d.class]++;
7       if cRule ≻ wRule then
8           Q = Q ∪ {cRule};
9           mark cRule;
10      else A = A ∪ <d.id, d.class, cRule, wRule>
11  end
```

Figure 3: CBA-CB: M2 (Stage 1)

Stage 2. For each case d that we could not decide which rule should cover it in Stage 1, we go through d again to find all rules that classify it wrongly and have a higher precedence than the corresponding *cRule* of d (line 5 in Figure 4). That is the reason we say that this method makes only slightly more than one pass over D. The details are as follows (Figure 4):

```
1   for each entry <dID, y, cRule, wRule> ∈ A do
2       if wRule is marked then
3           cRule.classCasesCovered[y]--;
4           wRule.classCasesCovered[y]++;
5       else wSet = allCoverRules(U, dID.case, cRule);
6           for each rule w ∈ wSet do
7               w.replace = w.replace ∪ {<cRule, dID, y>};
8               w.classCasesCovered[y]++;
9           end
10          Q = Q ∪ wSet
11      end
12  end
```

Figure 4: CBA-CB: M2 (Stage 2)

If *wRule* is marked (which means it is the *cRule* of at least one case) (line 2), then it is clear that *wRule* will cover the case represented by *dID*. This satisfies the two conditions. The numbers of data cases, that *cRule* and *wRule* cover, need to be updated (line 3-4). Line 5 finds all the rules that wrongly classify the *dID* case and have higher precedences than that of its *cRule* (note that we only need to use the rules in U). This is done by the function *allCoverRules*. The rules returned are those rules that may replace *cRule* to cover the case because they have higher precedences. We put this information in the *replace* field of each rule (line 7). Line 8 increments the count of *w*.classCasesCovered[*y*] to indicate that rule *w* may cover the case. *Q* contains all the rules to be used to build our classifier.

Stage 3. Choose the final set of rules to form our classifier (Figure 5). It has two steps:

Step 1 (line 1-17): Choose the set of potential rules to form the classifier. We first sort *Q* according to the relation "$\succ$". This ensures that Condition 1 above is satisfied in the final rule selection. Line 1 and 2 are initializations. The *compClassDistr* function counts the number of training cases in each class (line 1) in the initial training data. *ruleErrors* records the number of errors made so far by the selected rules on the training data.

In line 5, if rule *r* no longer correctly classifies any training case, we discard it. Otherwise, *r* will be a rule in our classifier. This satisfies Condition 2. In line 6, *r* will try to replace all the rules in *r.replace* because *r* precedes them. However, if the *dID* case has already been covered by a previous rule, then the current *r* will not replace *rul* to cover the case. Otherwise, *r* will replace *rul* to cover the case, and the *classCasesCovered* fields of *r* and *rul* are updated accordingly (line 7-9).

For each selected rule, we update *ruleErrors* and *classDistr* (line 10-11). We also choose a default class (i.e., *defaultClass*), which is the majority class in the remaining training data, computed using *classDistr* (line 12). After the default class is chosen, we also know the number of errors (called *defaultError*) that the default class will make in the remaining training data (line 13). The total number of errors (denoted by *totalErrors*) that the selected rules in *C* and the default class will make is *ruleErrors* + *defaultErrors* (line 14).

Step 2 (line 18-20): Discard those rules that introduce more errors, and return the final classifier *C* (this is the same as in M1).

```
1  classDistr = compClassDistri(D);
2  ruleErrors = 0;
3  Q = sort(Q);
4  for each rule r in Q in sequence do
5    if r.classCasesCovered[r.class] ≠ 0 then
6        for each entry <rul, dID, y> in r.replace do
7            if the dID case has been covered by a
               previous r then
8              r.classCasesCovered[y]--;
9            else  rul.classCasesCovered[y]--;
10       ruleErrors = ruleErrors + errorsOfRule(r);
11       classDistr = update(r, classDistr);
12       defaultClass = selectDefault(classDistr);
13       defaultErrors = defErr(defaultClass, classDistr);
14       totalErrors = ruleErrors + defaultErrors;
15       Insert <r, default-class, totalErrors> at end of C
16   end
17 end
18 Find the first rule p in C with the lowest totalErrors,
      and then discard all the rules after p from C;
19 Add the default class associated with p to end of C;
20 Return C without totalErrors and default-class;
```

Figure 5: CBA-CB: M2 (Stage 3)

Empirical Evaluation

We now compare the classifiers produced by algorithm CBA with those produced by C4.5 (tree and rule) (Release 8). We used 26 datasets from UCI ML Repository (Merz and Murphy 1996) for the purpose. The execution time performances of CBA-RG and CBA-CB are also shown.

In our experiments, *minconf* is set to 50%. For *minsup*, it is more complex. *minsup* has a strong effect on the quality of the classifier produced. If *minsup* is set too high, those possible rules that cannot satisfy *minsup* but with high confidences will not be included, and also the CARs may fail to cover all the training cases. Thus, the accuracy of the classifier suffers. From our experiments, we observe that once *minsup* is lowered to 1-2%, the classifier built is more accurate than that built by C4.5. In the experiments reported below, we set *minsup* to 1%. We also set a limit of 80,000 on the total number of candidate rules in memory (including both the CARs and those dropped-off rules that either do not satisfy *minsup* or *minconf*). 16 (marked with a * in Table 1) of the 26 datasets reported below cannot be completed within this limit. This shows that classification data often contains a huge number of associations.

Discretization of continuous attributes is done using the Entropy method in (Fayyad and Irani 1993). The code is taken from MLC++ machine learning library (Kohavi et al 1994). In the experiments, all C4.5 parameters had their default values. All the error rates on each dataset are obtained from 10-fold cross-validations. The experimental results are shown in Table 1. The execution times here are based on datasets that reside in the main memory.

Column 1: It lists the names of the 26 datasets. See (Liu, Hsu and Ma 1998) for the description of the datasets.

Column 2: It shows C4.5rules' mean error rates over ten complete 10-fold cross-validations using the original datasets (i.e., without discretization). We do not show C4.5 tree's detailed results because its average error rate over the 26 datasets is higher (17.3).

Column 3: It shows C4.5rules' mean error rate after discretization. The error rates of C4.5 tree are not used here as its average error rate (17.6) is higher.

Column 4: It gives the mean error rates of the classifiers built using our algorithm with *minsup* = 1% over the ten cross-validations, using both CARs and *infrequent* rules (dropped off rules that satisfy *minconf*). We use infrequent rules because we want to see whether they affect the classification accuracy. The first value is the error rate of the classifier built with rules that are not subjected to pruning in rule generation, and the second value is the error rate of the classifier built with rules that are subjected to pruning in rule generation.

Column 5: It shows the error rates using only CARs in our classifier construction without or with rule pruning (i.e., prCARs) in rule generation.

It is clear from these 26 datasets that CBA produces more accurate classifiers. On average, the error rate decreases from 16.7% for C4.5rules (without discretization) to 15.6-15.8% for CBA. Furthermore, our system is superior to

Table 1: Experiment Results

Datasets	c4.5rules w/o discr.	c4.5rules discr.	CBA (CARs + infreq)		CBA (CARs)		No. of CARs		Run time (sec) (CBA-RG)		Run time (sec) (CBA-CB)		No. of Rules in C
			w/o pru.	pru.	w/o pru	pru.	w/o pru.	pru.	w/o pru	pru.	M1	M2	
anneal*	5.2	6.5	1.9	1.9	3.2	3.6	65081	611	14.33	14.36	0.08	0.06	34
australian*	15.3	13.5	13.5	13.4	13.2	13.4	46564	4064	5.00	5.05	0.20	0.22	148
auto*	19.9	29.2	21.0	23.1	24.0	27.2	50236	3969	3.30	3.55	0.12	0.06	54
breast-w	5.0	3.9	3.9	3.9	4.2	4.2	2831	399	0.30	0.33	0.02	0.03	49
cleve*	21.8	18.2	18.1	19.1	16.7	16.7	48854	1634	4.00	4.30	0.04	0.06	78
crx*	15.1	15.9	14.3	14.3	14.1	14.1	42877	4717	4.90	5.06	0.43	0.30	142
diabetes	25.8	27.6	24.8	25.5	24.7	25.3	3315	162	0.25	0.28	0.03	0.01	57
german*	27.7	29.5	27.2	26.5	25.2	26.5	69277	4561	5.60	6.00	1.04	0.28	172
glass	31.3	27.5	27.4	27.4	27.4	27.4	4234	291	0.20	0.22	0.02	0.00	27
heart	19.2	18.9	19.6	19.6	18.5	18.5	52309	624	4.70	4.60	0.04	0.03	52
hepatitis*	19.4	22.6	15.1	15.1	15.1	15.1	63134	2275	2.80	2.79	0.09	0.05	23
horse*	17.4	16.3	18.2	17.9	18.7	18.7	62745	7846	3.2	3.33	0.35	0.19	97
hypo*	0.8	1.2	1.6	1.6	1.9	1.7	37631	493	45.60	45.30	1.02	0.40	35
ionosphere*	10.0	8.0	7.9	7.9	8.2	8.2	55701	10055	3.75	4.00	0.56	0.41	45
iris	4.7	5.3	7.1	7.1	7.1	7.1	72	23	0.00	0.00	0.00	0.00	5
labor	20.7	21.0	17.0	17.0	17.0	17.0	5565	313	0.17	0.20	0.00	0.00	12
led7	26.5	26.5	27.8	27.8	27.8	27.8	464	336	0.40	0.45	0.11	0.10	71
lymph*	26.5	21.0	20.3	18.9	20.3	19.6	40401	2965	2.70	2.70	0.07	0.05	36
pima	24.5	27.5	26.9	27.0	27.4	27.6	2977	125	0.23	0.25	0.04	0.02	45
sick*	1.5	2.1	2.8	2.8	2.7	2.7	71828	627	32.60	33.40	0.62	0.40	46
sonar*	29.8	27.8	24.3	21.7	24.3	21.7	57061	1693	5.34	5.22	0.30	0.12	37
tic-tac-toe	0.6	0.6	0.0	0.0	0.0	0.0	7063	1378	0.62	0.71	0.12	0.08	8
vehicle*	27.4	33.6	31.3	31.2	31.5	31.3	23446	5704	6.33	6.33	1.40	0.40	125
waveform*	21.9	24.6	20.2	20.2	20.4	20.6	9699	3396	13.65	13.55	2.72	1.12	386
wine	7.3	7.9	8.4	8.4	8.4	8.4	38070	1494	2.34	2.65	0.11	0.04	10
zoo*	7.8	7.8	5.4	5.4	5.4	5.4	52198	2049	2.73	2.70	0.61	0.32	7
Average	16.7	17.1	15.6	15.6	15.7	15.8	35140	2377	6.35	6.44	0.39	0.18	69

C4.5rules on 16 of the 26 datasets. We also observe that without or with rule pruning the accuracy of the resultant classifier is almost the same. Thus, those prCARs (after pruning) are sufficient for building accurate classifiers. Note that when compared with the error rate (17.1) of C4.5rules after discretization, CBA is even more superior.

Let us now see the rest of the columns, which give the number of rules generated and the run time performances of our system (running on 192MB DEC alpha 500).

Column 6: It gives the average numbers of rules generated by algorithm CBA-RG in each cross-validation. The first value is the number of CARs. The second value is the number of prCARs (after pruning). We see that the number of rules left after pruning is much smaller.

Column 7: It gives the average time taken to generate the rules in each cross-validation. The first value is the time taken when no pruning is performed. The second value is the time taken when pruning is used. With pruning, algorithm CBA-RG only runs slightly slower.

Column 8: It shows the average times taken to build each classifier using only prCARs. The first value is the running time of method 1 (M1), and the second value is that of method 2 (M2). We see that M2 is much more efficient than M1.

Column 9: It gives the average number of rules in the classifier built by CBA-CB using prCARs. There are generally more rules in our classifier than that produced by C4.5 (not shown here). But this is not a problem as these rules are only used to classify future cases. Understandable and useful rules can be found in CARs (or prCARs). These rules may or may not be generated by C4.5 since C4.5 does not generate all the rules.

Below, we summarize two other important results.

- Although we cannot find all the rules in 16 of the 26 datasets using the 80,000 limit, the classifiers constructed with the discovered rules are already quite accurate. In fact, when the limit reaches 60,000 in the 26 datasets (we have experimented with many different limits), the accuracy of the resulting classifiers starts to stabilize. Proceeding further only generate rules with many conditions that are hard to understand and difficult to use.
- We also ran the CBA algorithm with the datasets on disk rather than in the main memory, and increased the number of cases of all datasets by up to 32 times (the largest dataset reaches 160,000 cases). Experimental results show that both CBA-RG and CBA-CB (M2) have linear scaleup.

Related Work

Our system is clearly different from the existing classification systems, such as C4.5 (Quinlan 1992) and CART (Breiman et al 1984), which only produce a small set of biased rules. The proposed technique aims to generate the complete set of potential classification rules.

Several researchers, e.g., Schlimmer (1993), Webb (1993), and Murphy and Pazzani (1994) have tried to build classifiers by performing extensive search. None of them, however, uses association rule mining techniques.

Our classifier building technique is related to the covering method in (Michalski 1980). The covering method works as follows. For each class in turn, it finds the best rule for the class and then removes those training cases covered by the rule. The strategy is then recursively applied to the remaining cases in the class until no more training case is left. Searching for the rule is done heuristically. In (Webb 1993; Quinlan and Cameron-Jones 1995), large scale beam-search is used to search for the best rule. The results were, however, not encouraging. Our method is different. We first find all the rules, and then select the best rules to cover the training cases. Test results show that our classifier performs better than that built by C4.5. Our best rules are "global" best rules because they are generated using all the training data, while the best rules of the covering method are "local" best rules because it removes those covered cases after each best rule is found. We also implemented the covering method in our framework, but the performance is not good. We believe it is because "local" best rules tend to overfit the data.

This research is closely related to association rule mining (Agrawal and Srikant 1994). The Apriori algorithm in (Agrawal and Srikant 1994) has been adapted for discovering CARs. In CBA-RG, we do not use *itemset* (a set of items) as in algorithm Apriori. Instead, we use *ruleitem*, which consists of a *condset* (a set of items) and a class. We also use the rule pruning technique in (Quinlan 1992) to prune off those non-predictive and overfitting rules. This is not used in association rule mining. In addition, we build an accurate classifier that can be used for prediction from the generated rules. The association rule discovery is not concerned with building classifiers.

(Bayardo 1997) uses an association rule miner to generate high-confidence classification rules (*confidence* > 90%). (Ali, Manganaris and Srikant 1997) uses an association rule miner to form rules that can describe individual classes. Both works are not concerned with building a classifier from the rules. This paper has shown that it is possible to build an accurate classifier for prediction from the set of generated rules.

Conclusion

This paper proposes a framework to integrate classification and association rule mining. An algorithm is presented to generate all class association rules (CARs) and to build an accurate classifier. The new framework not only gives a new way to construct classifiers, but also helps to solve a number of problems that exist in current classification systems. In our future work, we will focus on building more accurate classifiers by using more sophisticated techniques and to mine CARs without pre-discretization.

References

Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules. *VLDB-94*, 1994.

Ali, K., Manganaris, S. and Srikant, R. 1997. Partial classification using association rules. *KDD-97*, 115-118.

Bayardo, R. J. 1997. Brute-force mining of high-confidence classification rules. *KDD-97*, 123-126.

Breiman, L, Friedman, J., Olshen, R. and Stone C. 1984. *Classification and regression trees*. Belmont: Wadsworth.

Clark, P. and Matwin, S. 1993. Using qualitative models to guide induction learning. *ICML-93,* 49-56.

Dougherty, J., Kohavi, R. Sahami, M. 1995. Supervised and unsupervised discretization of continuous features. *ICML-95*.

Fayyad, U. M. and Irani, K. B. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. *IJCAI-93*, 1022-1027.

Kohavi, R., John, G., Long, R., Manley, D., and Pfleger, K. 1994. MLC++: a machine learning library in C++. *Tools with artificial intelligence*, 740-743.

Liu, B. and Hsu, W. 1996. Post-analysis of learned rules. *AAAI-96*, 828-834.

Liu, B., Hsu, W. and Chen, S. 1997. Using general impressions to analyze discovered classification rules. *KDD-97*, 31-36.

Liu, B., Hsu, W. and Ma, Y. 1998. Building an accurate classifier using association rules. Technical report.

Mahta, M., Agrawal, R. and Rissanen, J. 1996. SLIQ: A fast scalable classifier for data mining. *Proc. of the fifth Int'l Conference on Extending Database Technology*.

Merz, C. J, and Murphy, P. 1996. UCI repository of machine learning database. [http://www.cs.uci.edu/~mlearn/MLRepository.html].

Michalski, R. 1980. Pattern recognition as rule-guided induction inference. *IEEE Transaction On Pattern Analysis and Machine Intelligence* 2, 349-361.

Murphy, P. and Pazzani, M. 1994. Exploring the decision forest: an empirical investigation of Occam's razor in decision tree induction. *J. of AI Research* 1:257-275.

Pazzani, M., Mani, S. and Shankle, W. R. 1997. Beyond concise and colorful: learning intelligible rules. *KDD-97*.

Quinlan, J. R. 1992. *C4.5*: *program for machine learning*. Morgan Kaufmann.

Quinlan, R. and Cameron-Jones, M. 1995. Oversearching and layered search in empirical learning. *IJCAI-95*.

Schlimmer, J 1993. Efficiently inducing determinations: a complete and systematic search algorithm that uses optimal pruning. *ICML-93*, 268-275.

Srikant, R. and Agrawal, R. 1996. Mining quantitative association rules in large relational tables. *SIGMOD-96*.

Wang, K., Tay, W. and Liu, B. 1998. An interestingness-based interval merger for numeric association rules. *KDD-98*.

Yoda, K., Fukuda, T. Morimoto, Y. Morishita, S. and Tokuyama, T. 1997. Computing optimized rectilinear regions for association rules. *KDD-97*.