CPAR: Classification based on Predictive Association Rules *

Xiaoxin Yin Jiawei Han
University of Illinois at Urbana-Champaign
{xyin1, hanj}@cs.uiuc.edu

Abstract

Recent studies in data mining have proposed a new classification approach, called associative classification, which, according to several reports, such as [7, 6], achieves higher classification accuracy than traditional classification approaches such as C4.5. However, the approach also suffers from two major deficiencies: (1) it generates a very large number of association rules, which leads to high processing overhead; and (2) its confidence-based rule evaluation measure may lead to overfitting.

In comparison with associative classification, traditional rule-based classifiers, such as C4.5, FOIL and RIPPER, are substantially faster but their accuracy, in most cases, may not be as high. In this paper, we propose a new classification approach, *CPAR* (*Classification based on Predictive Association Rules*), which combines the advantages of both associative classification and traditional rule-based classification. Instead of generating a large number of candidate rules as in associative classification, CPAR adopts a greedy algorithm to generate rules directly from training data. Moreover, CPAR generates and tests more rules than traditional rule-based classifiers to avoid missing important rules. To avoid overfitting, CPAR uses *expected accuracy* to evaluate each rule and uses the *best k rules* in prediction.

1 Introduction

In recent years, a new approach called *associative classification* [7, 6] is proposed to integrate association rule mining [1] and classification. It uses association rule mining algorithm, such as Apriori [1] or FPgrowth [5], to generate the complete set of association rules. Then it selects a small set of high quality rules and uses this rule set for prediction. The experiments in [7, 6] show that this approach achieves higher accuracy than traditional classification approaches such as C4.5 [8]. However, associative classification suffers from efficiency due to the facts that it often generates a very large number of rules in association rule mining, and also it takes efforts to select high quality rules from among them.

In this paper, we propose a novel approach called *CPAR* (Classification based on Predictive Association Rules). CPAR inherits the basic idea of FOIL [9] in rule generation and integrates the features of associative classification in predictive rule analysis. In comparison with associative classification, CPAR has the following advantages: (1) CPAR generates a much smaller set of high-quality predictive rules directly from the dataset; (2) to avoid generating redundant rules, CPAR generates each rule by considering the set of "already-generated" rules; and (3) when predicting the class label of an example, CPAR uses the best k rules that this example satisfies.

Moreover, CPAR employs the following features to further improve its accuracy and efficiency: (1) CPAR uses dynamic programming to avoid repeated calculation in rule generation; and (2) when generating rules, instead of selecting only the best literal, all the close-to-the-best literals are selected so that important rules will not be missed.

CPAR generates a smaller set of rules, with higher quality and lower redundancy in comparison with associative classification. As a result, CPAR is much more time-efficient in both rule generation and prediction but achieves as high accuracy as associative classification.

The outline of this paper is as follows. section 2 reviews the general ideas of rule-based classification. Section 3 describes the rule generation process in CPAR. Section 4 discusses how to predict class labels for unseen examples using the rules generated. The experimental results and performance study are presented in section 5, and we conclude the study in section 6.

2 Rule-Based Classification

Let T be a set of tuples. Each tuple t in T follows the scheme $(\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k)$, where $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k$ are k attributes. Each continuous attribute is first discretized into a categorical attribute.

*The work was supported in part by U.S. National Science Foundation NSF IIS-02-09199, a research fund from Univ. of Illinois, and an IBM Faculty Award.

DEFINITION 2.1. **(literal)** *A literal p is an attribute-value pair, taking the form of $(\mathbf{A}_i, v)$, in which $\mathbf{A}_i$ is an attribute and v a value. A tuple t satisfies a literal $p = (\mathbf{A}_i, v)$ if and only if $t_i = v$, where t_i is the value of the i^{th} attribute of t.*

DEFINITION 2.2. **(rule)** *A rule r, which takes the form of "$p_1 \wedge p_2 \wedge \cdots \wedge p_l \rightarrow c$," consists of a conjunction of literals $p_1, p_2, \ldots, p_l$, associated with a class label c. A tuple t satisfies rule r's body if and only if it satisfies every literal in the rule. If t satisfies r's body, r predicts that t is of class c. If a rule contains zero literal, its body is satisfied by any tuple.*

Two important association rule-based classifiers are CBA [7] and CMAR [6]. CBA first generates all the association rules *with certain support and confidence thresholds* as candidate rules. It then selects a small set of rules from them to form a classifier. When predicting the class label for an example, the best rule (i.e., with the highest confidence) whose body is satisfied by the example is chosen for prediction.

CMAR generates and evaluates rules in a similar way as CBA (but uses a more efficient FPtree structure [5]). A major difference is that it uses multiple rules in prediction, using weighted χ^2. Their experiments show that CMAR outperforms CBA in accuracy.

When datasets contain a large number of rows and/or columns, both rule generation and rule selection in CBA and CMAR are time consuming. In this paper, a novel approach is proposed to overcome this problem: Instead of generating candidate rules, a small set of predictive rules are directly generated from the dataset based on the rule prediction and coverage analysis.

3 Rule Generation

The basic idea of CPAR is from FOIL, which is introduced in section 3.1. The rule generation algorithm of CPAR is developed step-by-step in sections 3.2 and 3.3.

3.1 FOIL: A Brief Introduction. FOIL (First Order Inductive Learner), proposed by Ross Quinlan in 1993 [9], is a greedy algorithm that learns rules to distinguish positive examples from negative ones. FOIL repeatedly searches for the current best rule and removes all the positive examples covered by the rule until all the positive examples in the data set are covered. The algorithm FOIL is presented in Figure 1. For multi-class problems, FOIL is applied on each class: for each class, its examples are used as positive examples and those of other classes as negative ones. The rules for all classes are merged together to form the result rule set.

When selecting literals, *Foil Gain* is used to measure the information gained from adding this literal to the current rule. Suppose there are $|P|$ positive examples and $|N|$ negative examples satisfying the current rule r's body. After literal p is added to r, there are $|P^*|$ positive and $|N^*|$ negative examples satisfying the new rule's body. Then the Foil gain of p is defined as,

$$(3.1)\ gain(p) = |P^*|\left(\log\frac{|P^*|}{|P^*|+|N^*|} - \log\frac{|P|}{|P|+|N|}\right)$$

where $gain(p)$ is the number of bits saved in representing all the positive examples by adding p to r.

ALGORITHM 3.1. **FOIL**

Input: Training set $D = P \cup N$. (P and N are the sets of all positive and negative examples, respectively.)

Output: A set of rules for predicting class labels for examples.

```
Procedure FOIL
  rule set R ← Φ
  while |P| > 0
    N' ← N, P' ← P
    rule r ← empty_rule
    while |N'| > 0 and r.length < max_rule_length
      find the literal p that brings most gain
            according to P' and N'
      append p to r
      remove from P' all examples not satisfying r
      remove from N' all examples not satisfying r
    end
    R ← R ∪ {r}
    remove from P all examples satisfying r's body
  end
  return R
```

Figure 1: The FOIL algorithm

LEMMA 3.1. (**The running time of FOIL**). FOIL takes $O(nkm|R|)$ time, where there are n examples, each having k attributes, and each attribute having m values on average, and FOIL generates $|R|$ rules.

Proof. *There are km possible literals. When searching for the first literal of a rule, there are $O(n)$ examples left. So it takes $O(nkm)$ time to identify the first literal. Suppose each literal covers β of all remaining examples. When searching for the i^{th} literal, there are $O(\beta^i n)$ examples left ($0 \leq \beta \leq 1$). So searching for one rule takes $O(nkm)$ time, and generating $|R|$ rules takes $O(nkm|R|)$ time.*

3.2 Predictive Rule Mining. In this section we propose *Predictive Rule Mining (PRM)*, an algorithm which modifies FOIL to achieve higher accuracy and efficiency. One reason that FOIL does not achieve as high accuracy is that it generates a very small number of rules. In PRM, after an example is correctly covered by a rule, instead of removing it, its weight is decreased

by multiplying a factor. This "weighted" version of FOIL produces more rules and each positive example is usually covered more than once.

The most time consuming part of FOIL is evaluating every literal when searching for the one with the highest gain. In fact, similar to [4], to calculate the gain, we only need to know the information stored in a data structure called *PNArray*.

DEFINITION 3.1. **(PNArray)** *A PNArray stores the following information corresponding to rule r.*

1. *P and N: the numbers of positive and negative examples satisfying r's body.*
2. *$P(p)$ and $N(p)$: for each possible literal p, the numbers of positive and negative examples satisfying the body of rule r', the rule constructed by appending p to r.*

If there exists a PNArray corresponding to the current rule, the gain of every literal can be calculated, with the best one found, in $O(km)$ time, independent of the size of the dataset. The PNArray for the empty rule can be easily computed from the training set. We can scan every attribute of every tuple and increase the number of positive (or negative) examples satisfying every predicate. It takes $O(nk)$ time to compute this initial PNArray. If one adds/removes an example to/from the dataset, it takes $O(k)$ time to update the PNArray. By using PNArray, the algorithm Predictive Rule Mining (Figure 2) achieves much higher efficiency than FOIL on large datasets.

LEMMA 3.2. *Predictive rule mining presented in Algorithm 3.2 takes $O(nk|R|)$ time.*

Proof. *During the process of building a rule, it removes each example at most once. Moreover, it takes $O(k)$ time to update the PNArray when removing an example. So it takes $O(nk)$ time to build a rule. After building a rule, it changes the weights of at most n examples, and it takes $O(nk)$ time to update the PNArray. Therefore, Predictive Rule Mining (PRM) takes $O(nk|R|)$ time.*

3.3 Rule Generation in CPAR. In associative classification, association rule mining is used to generate candidate rules, which includes all conjunctions of literals that meet the support threshold. Then a subset of rules are selected from the candidates. This subset is built by combining the best δ rules for every example ($\delta = 1$ in [7], and $\delta = 4$ in [6]). This can be considered as doing exhaustive search in rule generation.

PRM has higher efficiency but lower accuracy than associative classification. In PRM, every rule is generated from the remaining dataset. Suppose an example t in the remaining dataset is covered by a rule r that is just generated. We are not sure whether r is the best rule for t because (1) r is generated by greedy algorithm; and (2) r is generated from the remaining dataset instead of the whole dataset. In PRM, we also generate at least a certain number of rules for each example (depending on the weight decay factor α). However, these several rules are not necessarily the best rules for the reasons illustrated above.

When selecting literals during the rule building process, PRM selects only the best literal and ignores all the others. There are often a few literals with a similar gain. Thus there are usually many rules with similar accuracy based on the remaining dataset. The "best" rule among them may not be the best rule based on the whole dataset. However, PRM selects only one of them, which may lead to missing some important rules.

Here a novel approach called CPAR is proposed. It stands in the middle between exhaustive and greedy algorithms and combines the advantages of both. CPAR builds rules by adding literals one by one, which is similar to PRM. However, instead of ignoring all literals except the best one, CPAR keeps all close-to-the-best literals during the rule building process. By doing so, CPAR can select more than one literal at the same time and build several rules simultaneously.

ALGORITHM 3.2. **Predictive Rule Mining (PRM)**

Input and output: The same as Algorithm 1.

```
Procedure Predictive Rule Mining
    set the weight of every example to 1
    rule set R ← Φ
    totalWeight ← TotalWeight(P)
    A ← Compute PNArray from D
    while TotalWeight(P) > δ · totalWeight
        N' ← N, P' ← P, A' ← A
        rule r ← emptyrule
        while true
            find best literal p according to A'
            if gain(p) < min_gain then break
            append p to r
            for each example t in P' ∪ N' not satisfying
            r's body
                    remove t from P' or N'
                    change A' according to the removal of t
            end
        end
        R ← R ∪ {r}
        for each example t in P satisfying r's body
            t.weight ← α · t.weight
            change A according to the weight decreased
        end
    end
    return R
```

Figure 2: The Predictive Rule Mining algorithm

The following is a detailed description of the rule generation algorithm of CPAR. Suppose at a certain step in the process of building a rule, after finding the best literal p, another literal q that has similar gain as p (e.g., differ by at most 1%) is found. Besides continuing building the rule by appending p to r, q is also appended to the current rule r to create a new rule r', which is pushed into the queue. Each time when a new rule is to be built, the queue is first checked. If it is not empty, a rule is extracted from it and is taken as the current rule. This forms the depth-first-search in rule generation.

Example. Figure 3 shows an example of how CPAR generates rules. After the first literal ($A_1 = 2$) is selected, two literals ($A_2 = 1$) and ($A_3 = 1$) are found to have similar gain, which is higher than other literals. Literal ($A_2 = 1$) is first selected and a rule is generated along this direction. After that, the rule ($A_1 = 2$, $A_3 = 1$) is taken as the current rule. Again two literals with similar gain ($A_4 = 2$) and ($A_2 = 1$) are selected and a rule is generated along each of the two directions. In this way, three rules are generated:

($A_1 = 2$, $A_2 = 1$, $A_4 = 1$).
($A_1 = 2$, $A_3 = 1$, $A_4 = 2$, $A_2 = 3$).
($A_1 = 2$, $A_3 = 1$, $A_2 = 1$).

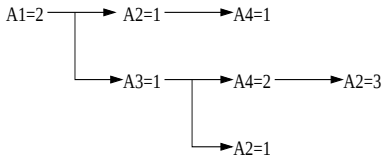

Figure 3: Some rules generated by CPAR.

LEMMA 3.3. *CPAR's rule generation takes $O(nk|R|)$ time.*

Proof. *When examining whether an example t satisfies a rule r's body, we examine whether t satisfies r's literals one by one. Thus it takes constant expected time to determine whether an example satisfies a rule's body. In CPAR's rule generation algorithm, when a rule is extracted from the queue, we need to find examples satisfying this rule's body from the remaining examples and calculate the PNArray for them, which takes $O(nk)$ time. With Lemma 3.2, we know that it still takes $O(nk)$ to build a rule and takes $O(nk|R|)$ time to build the rule set.*

4 Prediction Using Rules

4.1 Rule Evaluation. Before making any prediction, every rule needs to be evaluated to determine its prediction power. For a rule $r =$ "$p_1 \wedge p_2 \wedge \cdots \wedge p_l \rightarrow c$", we define its *expected accuracy* as the probability that an example satisfying r's body belongs to class c, or $Prob(t \in c \mid t\ satisfies\ r's\ body)$.

We use the *Laplace expected error estimate* [2] to estimate the accuracy of rules, which is defined as follows. The expected accuracy of a rule is given by:

$$LaplaceAccuracy = (n_c + 1)/(n_{tot} + k) \quad (4.2)$$

where k is the number of classes, n_{tot} is the total number of examples satisfying the rule's body, among which n_c examples belong to c, the predicted class of the rule.

4.2 Classification. Given a rule set containing rules for each class, we use the best k rules of each class for prediction, with the following procedure: (1) select all the rules whose bodies are satisfied by the example; (2) from the rules selected in step (1), select the best k rules for each class; and (3) compare the average expected accuracy of the best k rules of each class and choose the class with the highest expected accuracy as the predicted class.

We use multiple rules in prediction because (1) the accuracy of rules cannot be precisely estimated, and (2) one cannot expect that any single rule can perfectly predict the class label of every example satisfying its body. Moreover, we use the best k rules instead of all the rules because there are different number of rules for different classes, and we do not want to use lower ranked rules in prediction when there are already enough rules to make prediction.

5 Experimental Results

We have conducted an extensive performance study to evaluate accuracy and efficiency of CPAR and compare it with that of C4.5 [8], RIPPER [3], CBA [7] and CMAR [6].

As in [7] and [6], 26 datasets from UCI Machine Learning Repository are used. All the experiments are performed on a 1.7GHz Pentium-4 PC with 1GB main memory. All the approaches are implemented by their authors. The parameters of CPAR are set as the following. In the rule generation algorithm, δ is set to 0.05, *min_gain* to 0.7, and α to 2/3. The best 5 rules are used in prediction.

Table 1 shows the accuracy of the five approaches on 26 datasets from UCI ML Repository. 10-fold cross validation is used for every dataset.

Table 2 compares the running (training) time of RIPPER, CMAR (which is claimed to be more efficient than CBA) and CPAR on the 26 datasets. Notice that Table 2 uses both arithmetic and geometric average. This is because the running times of different datasets

Dataset	c4.5	ripper	cba	cmar	cpar
anneal	94.8	95.8	97.9	97.3	98.4
austral	84.7	87.3	84.9	86.1	86.2
auto	80.1	72.8	78.3	78.1	82.0
breast	95.0	95.1	96.3	96.4	96.0
cleve	78.2	82.2	82.8	82.2	81.5
crx	84.9	84.9	84.7	84.9	85.7
diabetes	74.2	74.7	74.5	75.8	75.1
german	72.3	69.8	73.4	74.9	73.4
glass	68.7	69.1	73.9	70.1	74.4
heart	80.8	80.7	81.9	82.2	82.6
hepatic	80.6	76.7	81.8	80.5	79.4
horse	82.6	84.8	82.1	82.6	84.2
hypo	99.2	98.9	98.9	98.4	98.1
iono	90.0	91.2	92.3	91.5	92.6
iris	95.3	94.0	94.7	94.0	94.7
labor	79.3	84.0	86.3	89.7	84.7
led7	73.5	69.7	71.9	72.5	73.6
lymph	73.5	79.0	77.8	83.1	82.3
pima	75.5	73.1	72.9	75.1	73.8
sick	98.5	97.7	97.0	97.5	96.8
sonar	70.2	78.4	77.5	79.4	79.3
tic-tac	99.4	98.0	99.6	99.2	98.6
vehicle	72.6	62.7	68.7	68.8	69.5
waveform	78.1	76.0	80.0	83.2	80.9
wine	92.7	91.6	95.0	95.0	95.5
zoo	92.2	88.1	96.8	97.1	95.1
Average	83.34	82.93	84.69	85.22	85.17

Table 1: Accuracy: C4.5, RIPPER, CBA, CMAR and CPAR

differ a lot, and the arithmetic average is dominated by the most time-consuming datasets. Using geometric average, equal weight is put on every dataset. Thus we consider geometric average as a more reasonable measure. Table 3 shows the average number of rules used in RIPPER, CMAR and CPAR.

	RIPPER	CMAR	CPAR
Arithmetic average	0.218	30.24	0.555
Geometric average	0.036	2.877	0.105

Table 2: Running time (in sec.): RIPPER, CMAR and CPAR

6 Conclusions

A new classification approach, called CPAR, is developed to integrate classification and association rule mining. Based on our performance study, CPAR achieves high accuracy and efficiency, which can be credited to the following distinguished features: (1) it uses greedy approach in rule generation, which is much more efficient than generating all candidate rules, (2) it uses a dynamic programming approach to avoid repeated calculation in rule generation, (3) it selects multiple literals and builds multiple rules simultaneously, and (4) it uses expected accuracy to evaluate rules, and uses the best k rules in prediction.

	RIPPER	CMAR	CPAR
Arithmetic average	8.20	305	244
Geometric average	5.74	185	106

Table 3: Number of rules: RIPPER, CMAR and CPAR

CPAR represents a new approach towards efficient and high quality classification. It is interesting to further enhance the efficiency and scalability of this approach and compare it with other well-established classification schemes. Moreover, the strength of the derived predictive rules also motivates us to perform an in-depth study on alternative approaches towards effective association rule mining.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB'94*, pp. 487–499, Santiago, Chile, Sept. 1994.

[2] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Proc. 1991 European Working Session on Learning (EWSL'91)*, pp. 151–163, Porto, Portugal, Mar. 1991.

[3] W. Cohen. Fast effective rule induction. In *ICML'95*, pp. 115–123, Tahoe City, CA, July 1995.

[4] J. Gehrke, R. Ramakrishnan, and V. Ganti. RainForest: A Framework for Fast Decision Tree Construction of Large Datasets. In *VLDB'98*, pp. 416-427, New York, NY, Aug. 1998.

[5] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD'00*, pp. 1–12, Dallas, TX, May 2000.

[6] W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *ICDM'01*, pp. 369–376, San Jose, CA, Nov. 2001.

[7] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD'98*, pp. 80–86, New York, NY, Aug. 1998.

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

[9] J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *Proc. 1993 European Conf. Machine Learning*, pp. 3–20, Vienna, Austria, 1993.