21st International Conference on Pattern Recognition (ICPR 2012)
November 11-15, 2012. Tsukuba, Japan

Searching for the Optimal Ordering of Classes in Rule Induction

Sezin Ata, Olcay Taner Yıldız
Department of Computer Engineering, Işık University, Şile İstanbul 34980 Turkey

Abstract

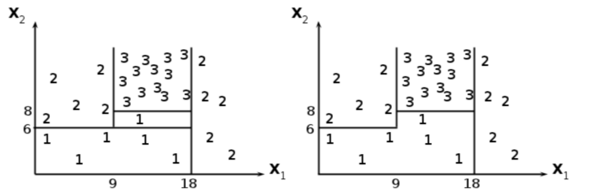
Rule induction algorithms such as Ripper, solve a $K > 2$ class problem by converting it into a sequence of $K - 1$ two-class problems. As a usual heuristic, the classes are fed into the algorithm in the order of increasing prior probabilities. In this paper, we propose two algorithms to improve this heuristic. The first algorithm starts with the ordering the heuristic provides and searches for better orderings by swapping consecutive classes. The second algorithm transforms the ordering search problem into an optimization problem and uses the solution of the optimization problem to extract the optimal ordering. We compared our algorithms with the original Ripper on 8 datasets from UCI repository [2]. Simulation results show that our algorithms produce rulesets that are significantly better than those produced by Ripper proper.

1. Introduction

Rule induction algorithms learn a rule set from a training set. A rule set is typically an ordered list of rules, where a rule contains a conjunction of terms and a class code which is the label assigned to an instance that is covered by the rule [7]. The terms are of the form $x_i = v$, $x_i < \theta$ or $x_i \geq \theta$, depending on respectively whether the input feature x_i is discrete or continuous. There is also a default class assigned to instances not covered by any rule. An example ruleset containing two rules for famous iris problem is:

If $F_3 < 1.9$ **and** $F_4 \geq 5.1$ **Then** iris-setosa
Else
  If $F_3 < 4.7$ **Then** iris-versicolor
  Else iris-virginica

Well known rule induction algorithms are C4.5Rules [11], PART [6], CN2 [3] and Ripper [4]. C4.5Rules generates a decision tree and then transforms it to a set of rules by writing each path from the root to a leaf as a rule; PART grows a partial decision tree and extracts a single rule from the best performing leaf; Ripper and CN2 directly produce a set of rules.



a) Current Approach' s Rule Set
```
If (x1<18)&&(x2<6)
  Then Class = 1
If (x1>9)&&(x2<8)
  Then Class = 1
If (x1<9)
  Then Class = 2
If (x1>18)
  Then Class = 2
Else
  Class = 3
```

b) Another Ordering' s Rule Set
```
If (x1<9)&&(x2>6)
  Then Class = 2
If (x1>18)
  Then Class = 2
If (x2< 8)
  Then Class = 1
Else
  Class = 3
```

Figure 1. For two different class orderings, separation of data and learned rulesets.

More recently, ant colony optimization (ACO) and fuzzy-rough set methodology (FRS) has been carried out for the extraction of rule-based classifiers. The first application of ACO to the rule induction task is the Ant-Miner algorithm [10]. Since then, several extensions and modifications of this sequential covering algorithm have been presented [8], [9]. The fuzzy-rough classification tree classifier [1], use fuzzy-rough hybrids to measure the dependency of decision attributes in the decision tree generation mechanism. Hu et al. [12] discover fuzzy classification rules based on the well-known Apriori algorithm.

Ripper, like CN2, learns rules to separate a positive class from a negative class. In the example above, Ripper first learn rules to separate class *iris-setosa* from both classes *iris-versicolor* and *iris-virginica*, then learn rules to separate class *iris-versicolor* from class *iris-virginica*. The ordering of classes is selected heuristically and may not be optimal in terms of error and/or complexity. In Figure 1 we see an example case, where two different orderings produce two different rulesets with the same error but different complexity, one composed of four rules with six terms, other composed of three rules with four terms. Although we prefer the second ordering, the heuristic may lead us to the first ordering.

In this paper, we propose two different algorithms to find the optimal class ordering. First algorithm, forward ordering search (FOS), does best first search in the ordering space. The algorithm starts from the heuristic's class ordering. At each step, we select the neighbor ordering that has the smallest test error. The algorithm terminates when none of the neighbor orderings has less error than the current best ordering.

Second algorithm, pairwise error approximation (PEA), assumes that the error of an ordering is the sum of $K(K-1)/2$ pairwise errors of classes. We train a random set of orderings and use the test error of them as training data to estimate the pairwise errors. Given the estimated pairwise errors, the algorithm searches for the optimal ordering exhaustively.

This paper is organized as follows: In Section 2, we explain the rule induction algorithm Ripper. In Sections 3 and 4 we explain our novel FOS and PEA algorithms respectively. We give our experimental results in Section 5 and conclude in Section 6.

2. Ripper

Ripper learns rules from scratch starting from an empty rule set. It has two phases: In the first phase, it builds an initial set of rules, one at a time, and in the second phase, it optimizes the rule set m times [4].

The pseudocode for learning ruleset from examples using Ripper is given in Figure 2. We start learning from an empty ruleset (Line 2), and learn ruleset for each class C_i one at a time. To do this, *heuristically*, we sort the classes increasingly according to prior probabilities (Line 3), and we try to separate each class C_p (positives) from the remaining classes $C_{p+1}, \ldots, C_K$ (negatives) (Line 5). Rules are grown (Line 8), pruned (Line 9) and added (Line 14) one by one to the ruleset. We stop adding rules when (i) the training error of the rule is larger than 0.5 (Line 11) or (ii) if there are no remaining positive examples (Line 7). After learning a ruleset, it is optimized twice (Line 17) and simplified (Line 18).

3. Forward Ordering Search

Our first proposed algorithm, namely forward ordering search, views optimizing the ordering of classes in Ripper as a search in the state space of all possible orderings. In our case, the search space contains all possible permutations of classes. Although the search space is finite, it is not possible to train/validate all orderings and select the best one from K! distinct orderings. There

```
1  Ruleset Ripper($D$)
2    $RS$ = {}
3    $C_i$ ordered in increasing prior probability
4    for $p$ = 1 to $K-1$
5      Pos = $C_p$, Neg = $C_{p+1}, \ldots, C_K$
6      $RS_p$ = {}
7      while $D$ contains positive samples
8        Divide $D$ into Grow set $G$ and Prune set $P$
9        $r$ = GrowRule($G$)
10       PruneRule($r$, $P$)
11       if CalculateError($r$)> 0.5
12         break
13       else
14         $RS_p$ = $RS_p$ + $r$
15         Remove examples covered by $r$ from $D$
16     for $i$ = 1 to 2
17       OptimizeRuleset($RS_p$, $D$)
18       SimplifyRuleset($RS_p$, $D$)
19     $RS$ = $RS$ + $RS_p$
20   return $RS$
```

Figure 2. Pseudocode for learning a ruleset using Ripper on dataset D

is hence need for a strategy which finds a good solution visiting only a small part of the search space.

We define an exchange operator that modify the ordering and allow moving from one state to another. Forward search algorithm starts from an initial state and use exchange operators. We select the ordering that the heuristic provides as the initial state. Although not the optimal ordering, the heuristic ordering generally works well.

Let say we have the ordering $C_1C_2C_3\ldots C_{K-1}C_K$. The exchange operator creates the following $K-1$ candidate orderings: $C_2C_1C_3\ldots C_{K-1}C_K$, $C_1C_3C_2\ldots C_{K-1}C_K$, $C_1C_2C_4\ldots C_{K-1}C_K$, $\ldots$, $C_1C_2C_3\ldots C_KC_{K-1}$.

After an operator application, the state evaluation function compares the goodness value of the next state(s) with that of the current state and accepts/rejects the operator depending on whether the goodness value is improved or not. This state evaluation function trains and validates the Ripper algorithm via 10×10-fold cross-validation with the ordering corresponding to the next state(s) and favors ordering that is most accurate. We stop the search when no candidate improves on the current best. Another possibility is to stop when the error falls below a certain level, or when a fixed number of iterations are made.

4. Pairwise Error Approximation

Our second proposed algorithm, pairwise error approximation, assumes that the expected error of an ordering, that is, the expected error of the Ripper algorithm trained with that ordering, is the sum of $K(K-1)/2$ pairwise expected errors of classes. The algorithm then tries to estimate the expected error contribution of each pair of classes. More formally, the expected error of the Ripper algorithm with ordering π_i is defined as

$$\hat{E}_{\pi_i} = \sum_{j=1}^{K} \sum_{k=1,[j]<_{\pi_i}[k]}^{K} e_{jk} \tag{1}$$

where $[j]$ represents the index of class j with respect to the ordering π_i and e_{jk} represents the error contribution of separation of class j from class k. For example, the expected error of the ordering 123 (three class problem) is defined as

$$\hat{E}_{123} = e_{12} + e_{13} + e_{23} \tag{2}$$

e_{jk} contains two types of instances (See Figure 4):

- False positives, instances of class k covered by the rules of class j.
- False negatives, instances of class j covered by the rules of class k.

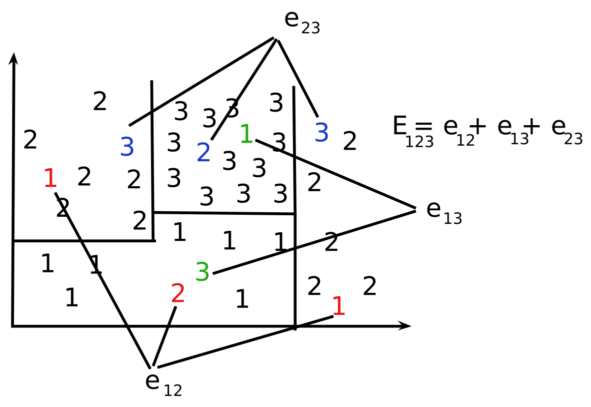


Figure 3. The expected error of the ordering 123 and its components (e_{ij}'s) for a dataset with $K = 3$.

Since we can not estimate e_{jk} from a single ordering, we run Ripper algorithm N times with N random orderings π_i and get the test errors E_{π_i}. Total estimation error over N runs is defined as

$$E_t = \sum_{i=1}^{N} (E_{\pi_i} - \hat{E}_{\pi_i})^2 \tag{3}$$

Table 1. Details of the datasets. d: Number of attributes, K: Number of classes, n: Sample size

Dataset	d	K	n
led7	7	7	3200
ocr	256	10	600
*opt*digits	64	10	3823
*pen*digits	16	10	7494
*seg*ment	19	7	2310
*shu*ttle	9	7	58000
*wine*quality	11	7	6497
*yea*st	8	10	1484

An ordering with K classes has $K(K-1)/2$ different e_{jk} pairs. Since the number of all possible pairs is $K(K$ - 1), each e_{jk} will appear N / 2 times in N random orderings approximately.

In order to minimize the total estimation error, we take the partial derivatives of E_t with respect to e_{jk}'s and solve the following system of linear equations

$$\forall_{j,k} \frac{\partial E_t}{\partial e_{jk}} = 0 \tag{4}$$

with e_{jk}'s as unknown variables.

Now we have the error contributions of all class pairs e_{jk}, we can estimate the error of any ordering π_i using Equation 1 without actually running Ripper with that ordering π_i. Not only that, we can also search all possible class orderings to get the optimal ordering exhaustively:

$$\pi_{optimal} = \arg\min_{\pi_i} \hat{E}_{\pi_i} \tag{5}$$

5. Experiments

5.1. Setup

In this section, we compare the performance of our proposed algorithms FOS and PEA with Ripper proper in terms of generalization error. We did our experiments on 8 data sets from UCI repository [2] (See Table 1). We use 10×10-fold cross-validation to generate training and test sets and paired t test [5] to compare algorithms with a confidence level of $\alpha = 0.05$. For PEA algorithm, we take N = 100, that is, we run Ripper algorithm with 100 random orderings π_i.

5.2. Results

Table 2 shows the average and standard deviations of error rates of rulesets generated using Ripper and our proposed algorithms FOS and PEA. We see from the results that, FOS is significantly better than Ripper in

Table 2. The average and standard deviations of error rates of rulesets generated using Ripper, FOS, and PEA. Statistically significant results are shown in boldface.

Dataset	Ripper	FOS	PEA
led7	31.83 ± 0.23	$\mathbf{29.67 \pm 0.23}$	$\mathbf{28.93 \pm 0.21}$
ocr	26.61 ± 0.58	$\mathbf{24.73 \pm 0.59}$	$\mathbf{22.08 \pm 0.54}$
opt	10.96 ± 0.14	$\mathbf{10.55 \pm 0.15}$	$\mathbf{8.57 \pm 0.12}$
pen	5.32 ± 0.07	$\mathbf{4.87 \pm 0.08}$	$\mathbf{4.49 \pm 0.08}$
seg	6.54 ± 0.17	$\mathbf{4.38 \pm 0.12}$	$\mathbf{5.03 \pm 0.14}$
shu	0.04 ± 0.00	$\mathbf{0.03 \pm 0.00}$	$\mathbf{0.01 \pm 0.00}$
wine	46.32 ± 0.16	46.23 ± 0.13	56.64 ± 0.23
yea	43.09 ± 0.35	$\mathbf{42.39 \pm 0.40}$	43.38 ± 0.36

Table 3. The average number of distinct orderings visited by FOS.

led7	ocr	opt	pen	seg	shu	wine	yea
40	18	25	41	22	12	17	25

Table 4. The average estimation error $\overline{E_t}$ calculated over N = 100 random orderings.

led7	ocr	opt	pen	seg	shu	wine	yea
0.31	0.63	0.20	0.08	0.24	1.86	3.41	2.67

seven datasets out of eight datasets. The results show that (i) although heuristic ordering works well in general, it is not optimal, (ii) best first search in the ordering space helps us to find significantly better orderings than the heuristic ordering in terms of error rate.

Table 3 shows the average number of states visited by FOS while searching the ordering space. The number of states is not so large and if compared by the number of classes in each dataset (K), we see that FOS stops early, it makes on average 2 to 5 iterations.

Similarly, PEA is significantly better than Ripper in six datasets out of eight datasets. For the other two datasets, namely *winequality* and *yeast*, the e_{jk}'s have a large variance across the possible class orderings it belong to, and the PEA estimate of the e_{jk}'s lead to noisy approximation of the overall error rate of a given ordering. Table 4, which shows the average estimation error $\overline{E_t}$ calculated over N = 100 random orderings, supports this view. These results (i) support our assumption that the error of an ordering is the sum of pairwise errors of classes, and (ii) show that our estimation of those pairwise errors works well since PEA finds mostly significantly better orderings than the heuristic ordering in terms of error rate.

6. Conclusion

Current heuristic approach used in Ripper that orders the classes in a dataset according to their sample sizes, usually does not give the most accurate classification. In this paper, we propose two algorithms to improve this heuristic.

The first algorithm, FOS, is based on best first search in the ordering space and guaranteed to find a better ordering if there exist one. The second algorithm, PEA, although not guarantees to find a better ordering, is usually better than FOS (and therefore Ripper proper) and can be improved by including more random orderings in the optimization.

References

[1] R. Bhatt and M. Gopal. Frct: Fuzzy-rough classification trees. *Pattern Analysis and Applications*, 11:73–88, 2008.

[2] C. Blake and C. Merz. UCI repository of machine learning databases, 2000.

[3] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Lecture Notes in Artificial Intelligence*, volume 482, pages 151–163, 1990.

[4] W. W. Cohen. Fast effective rule induction. In *The Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[5] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning classifiers. *Neural Computation*, 10:1895–1923, 1998.

[6] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the 15th International Conference on Machine Learning*, pages 144–151, 1998.

[7] J. Fürnkranz. Separate-and-conquer learning. *Artificial Intelligence Review*, 13:3–54, 1999.

[8] D. Martens, M. D. Backer, J. Vanthienen, M. Snoeck, and B. Baesens. Classification with ant colony optimization. *IEEE Transions on Evolutionary Computation*, 11:651–665, 2007.

[9] J. L. Olmo, J. R. Romero, and S. Ventura. Using ant programming guided by grammar for building rule-based classifiers. *IEEE Transactions On Systems, Man, And CyberneticsPart B: Cybernetics*, 41:1585–1599, 2011.

[10] R. Parpinelli, A. A. Freitas, and H. S. Lopes. Data mining with an ant colony optimization algorithm. *IEEE Transions on Evolutionary Computation*, 6:321–332, 2002.

[11] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Meteo, CA, 1993.

[12] G.-H. T. Yi-Chung Hu, Ruey-Shun Chen. Finding fuzzy classification rules using data mining techniques. *Pattern Recognition Letters*, 24:509–519, 2003.