ROCCER: an Algorithm for Rule Learning Based on ROC Analysis

Ronaldo C. Prati
Institute of Mathematics and Computer Science
University of São Paulo
São Carlos (SP), Brazil
prati@icmc.usp.br

Peter A. Flach
Department of Computer Science
University of Bristol
Bristol, United Kingdom
Peter.Flach@bristol.ac.uk

Abstract

We introduce a rule selection algorithm called ROCCER, which operates by selecting classification rules from a larger set of rules – for instance found by Apriori – using ROC analysis. Experimental comparison with rule induction algorithms shows that ROCCER tends to produce considerably smaller rule sets with compatible Area Under the ROC Curve (AUC) values. The individual rules that compose the rule set also have higher support and stronger association indexes.

1 Introduction

Classification rule learning can be defined as the process of, given a set of training examples, finding a set of rules that can be used for classification or prediction. Almost all classification rule learning algorithms belong to one of two families, namely separate-and-conquer and divide-and-conquer algorithms. The two families share a number of characteristics, most notably the assumption that the example space contains large continuous regions of constant class membership. The major differences are outlined below.

In the separate-and-conquer family of classification rule learning algorithms [Fürnkranz, 1999], the search procedure is generally an iterative greedy set-covering algorithm that on each iteration finds the best rule (according to a search criterion) and removes the covered examples. The process is repeated on the remaining examples until all examples have been covered or some stopping criterion has been met. In order to build a classifier, the rules found in each iteration are gathered to form either an ordered rule list (a decision list) or an unordered rule set. In the former case the classification is given by the first rule in the list that fires, while in the latter case the predictions of rules that fire are combined to predict a class.

This approach contrasts with the divide-and-conquer family of learning algorithms [Quinlan, 1993], where a global classifier is built following a top-down strategy by consecutive refinements of a partial theory. The result is generally expressed as a decision tree, which completely divides the instance space into non-overlapping axis-parallel hyper-rectangles.

As decision tree induction necessarily builds complete disjoint models, in some complex domains with high-dimensional feature spaces these models can be quite complex. In such cases, individual rule learning algorithms may be preferable since they are capable of inducing overlapping and simpler rule sets [van den Eijkel, 2003]. However, one of the problems of set-covering rule learning is that rules are found in isolation, although they are used in the context of the others inside the classifier. This issue can pose several problems for a rule learning algorithm. First, from the learning perspective, fewer and fewer examples are available as covering progresses. In latter stages of induction, this may lead to fragmented training sets and rules with insufficient statistical support [Domingos, 1996]. Furthermore, each new rule is constructed in complete ignorance of the examples already covered by the previously induced rules. If a bad rule has been introduced in the rule set, there is no chance of finding a better rule for those examples (there is no backtracking).

In this work we present a new rule learning algorithm named ROCCER, aimed to overcome such problems. The main idea of the algorithm is to construct a convex hull in ROC space. We evaluate ROCCER on a broad set of benchmark domains from the UCI repository [Blake and Merz, 1998] and compare it with other rule induction methods. The paper is organized as follows. In Section 2 we discuss the background related to this work. Section 3 presents our proposed algorithm, and Section 4 contains the experimental evaluation. In Section 5 we make some concluding remarks.

2 Related work

Several approaches have been proposed in the literature to overcome the fragmentation problem. [Liu *et al.*, 1998] decouple the rule generation from the covering step. The basic idea is to use an association rule algorithm to gather all rules that predict the class attribute and also pass a minimum quality criterion into the rule set. Although this approach might overcome some of the problems of the separate-and-conquer approach, and its performance was reported to outperform standard rule learning algorithms in some domains, its main drawback is related to the number of generated rules. Often they considerably outnumber the examples, implying serious difficulties from a knowledge discovery point of view, since the understandability and usability of the generated model would decrease and the risk of overfitting would increase. This idea has been extended in the AprioriC and AprioriSD algorithms [Javanoski and Lavrač, 2001; Kavšek *et al.*, 2003] by adding an additional filtering step to remove some of the redundant rules. However, AprioriC still tends to build large rule sets and AprioriSD has been developed mainly for subgroup discovery.

A different approach is to use weighted covering, which has been independently proposed by [Cohen and Singer, 1999] and [Weiss and Indurkhya, 2000]. Instead of completely removing the examples covered by the best rule on each iteration, their weights are decreased, and in each iteration the covering algorithm concentrates on highly-weighted (i.e., infrequently covered) examples. [Lavrač *et al.*, 2004] also discuss the use of weighted covering in a subgroup discovery context. Alternative methods to remove redundant rules are based on pruning [Fürnkranz and Widmer, 1994; Cohen, 1995].

Some authors propose the use of confidence thresholds for classification. [Gamberger and Lavrač, 2000] include only rules with high confidence in the rule set. The classifier then refuses to classify a new instance if none of the rules cover it. [Ferri *et al.*, 2004] extends this idea by retraining a new classifier on the unclassified examples.

3 The ROCCER rule selection algorithm.

Our approach relies on using ROC analysis for selecting rules instead of using a classical covering algorithm. Roughly speaking, a ROC graph is a plot of the fraction of positive examples misclassified — false positive rate (*fpr*) — on the x axis against the fraction of positive examples correctly classified — true positive rate (*tpr*) — on the y axis. It is possible to plot in a ROC graph either a single rule, a classifier (formed by a rule set, or not) or even a partial classifier (formed by a subset of a rule set, for instance).

For a threshold-based classifier one can obtain several pairs of points (fpr_i, tpr_i) by varying the threshold. If we trace a line connecting those points we obtain a curve in the ROC space that represents the behaviour of the classifier over all possible choices of the respective threshold. In a rule learning context, [Fürnkranz and Flach, 2005] show that rule learning using a set covering approach can be seen as tracing a curve in ROC space. To see why, assume we have a empty rule list, represented by the point $(0,0)$ in ROC space. Adding a new rule R_j to the rule list implies a shift to the point (fpr_j, tpr_j), where fpr_j and tpr_j is the *tpr* and *fpr* of the partial rule list (interpreted as a decision list) containing all rules already learnt including R_j. A curve can be traced by plotting all partial rule lists (fpr_j, tpr_j), for j varying from 0 to the total number n of rules in the final rule list in the order they are learnt. A final default rule that always predicts the positive class can be added at the end, connecting the point (fpr_n, tpr_n) to the point $(1,1)$.

Our approach is based on this observation, and the fact that the points which represent the optimum thresholds lie on the upper convex hull of the ROC curve [Provost and Fawcett, 2001]. Rules come from a external larger set of rules (in our implementation, we use the Apriori association rule learning algorithm, fixing the head of the rules to each of the possible class values, allowing us to deal with multi-class problems) and we perform a selection step based on the ROC curve. The basic idea is to only insert a rule in the rule list if the insertion leads to a point outside the current ROC convex hull (the current ROC convex hull is the upper convex hull of the rules that are already in the rule list). Otherwise the rule is discarded. For a better understanding of how our algorithm works, we first describe it using an example.

Rules are selected separately for each class, and are kept in an ordered rule list. Let's label the class we are selecting rules for **positive**; the label **negative** represents the (conjunction of) examples in the other class(es). First, we initialize the rule list with a default rule, $R_{default}$, which always predicts positive. The current ROC convex hull is formed then by 2 points, $(0,0)$ and $(1,1)$ which means "ignore the default rule (classify everything as negative) or use the default rule (classify everything as positive)". We use fpr_{Ri}, tpr_{Ri} to refer to the rule R_i's true and false positive rates, and tpr_i, fpr_i to refer to a point i in the ROC curve, representing the corresponding rule list's true and false positive rates. Suppose now we are inserting a new rule R_1. As the actual convex hull is formed only by the line segment $(0,0)-(1,1)$, R_1 will only be inserted if the point formed by the rule's (fpr_{R1}, tpr_{R1}) is above the convex hull. Let's say R_1 is inserted. The current convex hull is then updated, and contains the points $(0,0), (fpr_1, tpr_1), (1,1)$, where $fpr_1 = fpr_{R1}$ and $tpr_1 = tpr_{R1}$. This process is depicted in Figure 1.

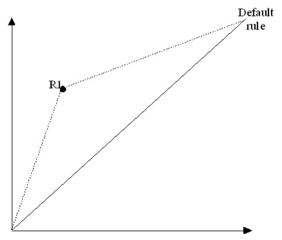
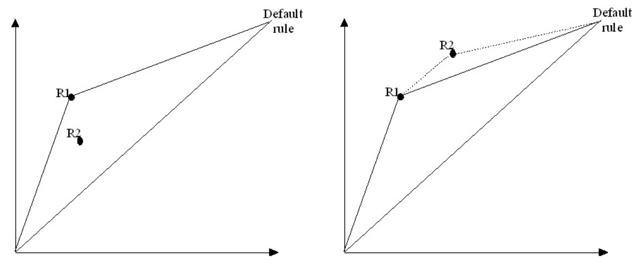


Figure 1: R_1 leads to a point outside the current convex hull (main diagonal) and is therefore inserted in the rule list.

Suppose we are now trying to insert a second rule R_2. As we have said before, in the standard set-covering approach the learning of a new rule does not take into account the rules already learnt. In our approach, we try to overcome this issue using the ROC graph to analyze interactions among rules. We do this by comparing the rule we are trying to insert with the slopes of each of he line segments in the current convex hull in the ROC graph. In our example, if the slope of the point formed by the origin and (fpr_{R2}, tpr_{R2}) is above the line from the origin to the point (fpr_1, tpr_1), we say that R_2 "improves in relation to" R_1 and insert R_2 in the rule list. This is similar to if R_2 had been learned before R_1 in the set-covering approach. By comparing with the rules which are already in the rule list, our algorithm provides a kind of backtracking. This insertion will, of course, produce changes to the other points in the ROC curve. The first non-trivial point in the ROC curve changes to (fpr_{R2}, tpr_{R2}). If R_1 and R_2 do not have any overlap, the second point will be $(fpr_{R2} + fpr_{R1}, tpr_{R2} + tpr_{R2})$. If R_1 and R_2 do have some overlap, however, we should discount the examples covered by both rules to calculate the second point.

If R_2 is not inserted at the first iteration, we proceed by comparing with the remaining line segments in the ROC convex hull. Before we compare with the next line segment, we update R_2's *fpr* and *tpr* by removing the examples covered by R_1 (the examples which evaluate to true for R_1's antecedent). In fact, we are "interpreting" R_2 as $\neg R_1 \wedge R_2$. If the updated position of R_2 is above the line from R_1 to $R_{default}$, R_2 is inserted in the rule list after R_1. This process is depicted in Figure 2. Otherwise (since no further rules remain) R_2 is discarded. The pseudo-code of this algorithm is shown in Algorithm 1.



(a) R_2 does not improve the convex hull when compared to R_1...

(b) ...but it does when compared with $R_{default}$.

Figure 2: Finding the right point to insert R_2.

Algorithm 1: The ROCCER algorithm.

```
Data: $RS_{in}$: A (large) rule set for a given class
Result: $RS_{out}$: A (smaller) rule list containing the
        selected rules
$RS_{out} = \{R_{default}\}$;
foreach $Rule \in RS_{in}$ do
    TryToInsertRule($Rule$)
endfch
return $RS_{out}$

procedure TryToInsertRule($Rule$)
  RuleToCompare = first rule in $RS_{out}$;
  repeat
      if Rule's (fpr,tpr) is outside convex hull then
          Insert Rule into $RS_{out}$ before RuleToCompare
      else
          /* shift to a new origin */
          Remove all the examples from Rule which are
          covered by RuleToCompare;
          Actualize Rule's (fpr,tpr);
          RuleToCompare = next rule in $RS_{out}$
      endif
  until $RuleToCompare \neq R_{default}$;
  if Rule is not inserted then
      Discard Rule
  endif
```

Due to overlapping coverage among rules, this process does not necessarily lead to a convex curve. The insertion of a new rule in the rule list may introduce concavities before or after the point of insertion. If concavities occur after an insertion, the inserted rule covers examples originally covered by subsequent rules, which decreases the latter's precision. In this case, we remove the rules where the concavity occurs. Alternatively, if the concavity occurs before the insertion point, both rules share a region where they misclassify some examples. In this case, it is unreasonable to use the partial rule list including the first rule but excluding the second, because the corresponding ROC point is under the convex hull. Therefore, we construct the disjunction of the two rules and treat it as a single rule.

In our implementation, rules presented to ROCCER are initially ordered, using the Euclidean distance to the point $(0,1)$ in the ROC space. However, due to the possibility to remove a selected rule and the (implicit) backtracking, the order dependence in selecting rules is lower than a decision list in inducing rules. Experiments with another orderings would be an interesting issue for further work, though.

This concludes the description of the training phase. For classification, we also use a ROC-based method. Bayes' theorem states that the odds that a classifier correctly classifies an instance (posterior odds) is given by the likelihood ratio times the odds of the instance being of the predicted class (prior odds). In ROC space, the likelihood ratio can be interpreted as *tpr/fpr*. Recall that we selected rules separately for each class. Thus, we have a ROC convex hull for each class. To classify a new instance we also consider each class separately, and, for each class, we determine the first rule that fires in the respective ROC convex hull. This rule has an associated (*tpr*,*fpr*) in the ROC curve, which yields a likelihood ratio. The posterior odds is then converted back to a probability (for ranking), or we select the class with maximum posterior odds (for classification).

4 Experimental evaluation

In order to empirically evaluate our proposed approach, we performed a experimental evaluation using 16 data sets from UCI [Blake and Merz, 1998]. We used only data sets without missing values, as Apriori (the association rule algorithm we use to generate the rules for subsequent selection by ROCCER) can't handle them. Table 1 summarizes the data sets employed in this study. It shows, for each data set, the number of attributes (#Attrs), the number of examples (#Examples), and percentage of examples in the majority class (%MajClass) – although ROCCER can handle more than two classes, in order to calculate AUC values we restricted our experiments to two-class problems. For data sets having more than two classes, we chose the class with fewer examples as the positive class, and collapsed the remaining classes as the negative.

ROCCER's results were compared with those obtained by the following rule learning systems:

CN2 This algorithm is a classical implementation of the separate-and-conquer rule learning family. In its first version [Clark and Niblett, 1989] CN2 induces a decision list using entropy as a search heuristic. It has later

#	Data set	# Attrs	# Examples	%MajClass
1	Breast	10	683	65.00
2	Bupa	7	345	57.98
3	E.Coli	8	336	89.58
4	Flag	29	194	91.24
5	German	21	1000	70.00
6	Glass	10	214	92.07
7	Haberman	4	306	73.53
8	Heart	14	270	55.55
9	Ionosphere	34	351	64.10
10	Kr-vs-Kp	37	3196	52.22
11	Letter-a	17	20000	96.06
12	New-thyroid	6	215	83.72
13	Nursery	9	12960	97.45
14	Pima	9	768	65.10
15	Satimage	37	6435	90.27
16	Vehicle	19	846	76.48

Table 1: UCI data sets used in our experiments.

been modified to incorporate the induction of unordered rule sets and Laplace error correction as evaluation function [Clark and Boswell, 1991].

Ripper [Cohen, 1995] proposed Ripper in the *Incremental Reduced Error Pruning* (IREP) [Fürnkranz and Widmer, 1994] context. It has features such as error-based pruning and an MDL-based heuristic for determining how many rules should be learned.

Slipper This algorithm is a further improvement of Ripper which uses a weighted set-covering approach [Cohen and Singer, 1999].

C4.5 [Quinlan, 1993]'s C4.5 is almost a standard in empirical comparison of symbolic learning algorithms. It is a member of divide-and-conquer family. C4.5 uses information gain as quality measure to build a decision tree and a post-pruning step based on error reduction. We can consider each branch in a decision tree as a rule.

Ripper and Slipper were used with -a option to generate rules for both classes. CN2 was used in its two versions, ordered (CN2OR) and unordered (CN2). We also evaluated both pruned (C45) and non-pruned (C45NP) trees induced by C4.5. All other parameters were set to default values. In order to calculate the AUC values we estimated probabilities of each rule using Laplace correction. For the unordered version of CN2, probabilities were estimated using all fired rules. AUC values were estimated using the trapezoidal rule. We used [Borgelt and Kruse, 2002]'s implementation of Apriori to generate the large rule sets used by ROCCER. The parameters were set to 50% of confidence and 1/3 of the percentage of minority class as support. For ROCCER, the probabilities were estimated by the posterior odds (described in Section 3). We also compare with a bagging of all rules generated by Apriori.

We ran the experiments using 10-fold stratified cross-validation. The experiment is paired, *i.e.*, all inducers were given the same training and test files. The averaged AUC values (and respective standard deviations in brackets) are shown in Table 2. We also perform a two-tailed Dunett multiple comparison with a control procedure[1] using ROCCER as control (the other results against ROCCER). Cells having AUC values statistically better than ROCCER are represented in dark gray while light gray is used to represent cells statistically worse than ROCCER, both with 95% confidence level.

Table 2 shows relatively few statistically significant differences. Comparing against C4.5, ROCCER achieved 4 wins and 1 loss. This is the same score if we compare ROCCER against Ripper. Against Slipper, the results are 5 wins and no losses. A comparison of ROCCER against C4.5 without pruning, CN2 and CN2 ordered yield 2 losses and no wins. We believe these two losses are due to the high degree of class skew in those two datasets (they are the most skewed in our study). In order to allow Apriori to find rules for both classes in these domains, the support parameter used in Apriori is very low. In these cases we have both a small number of rules generated for the minority class and a large number of rules generated for the majority class. Further improvements should be made in ROCCER to cope with such situations (for instance, it would be interesting to introduce different minimum support for each class). Taking into account all the rule learning algorithms, the score is 12 wins and 9 losses (all losses are concentrated in the two skewed domains, though). Comparing with all the generated rules, ROCCER produced 6 wins and no losses. We also compute AUC value on selecting k (the same number as ROCCER)random rules and rules with higher individual AUC values. Due to lack of space results are not shown in this paper, but they are in most of the cases significantly worst and never better than ROCCER. This indicates that ROCCER's selection procedure is responsible for a gain of performance over all the presented rules.

The good results with both versions of CN2, and the relatively poor AUC figures for Ripper and Slipper, are worth noticing, and may be explained by the absence of pruning mechanisms. It has already been reported in the literature that non-pruned trees are better for probability prediction and thus produce higher AUC values [Provost and Domingos, 2003]. It might be expected that a similar phenomenon also would occur with algorithms from the separate-and-conquer family. Ripper and Slipper are – at least conceptually – similar to CN2 but incorporate, respectively, rule pruning and weighted coverage.

Table 3 presents the average size (in number of rules) of the rule sets for each algorithm. Size 0 means that the classifier is formed only by the default rule. The picture here is more clear. Apart from some exceptions, ROCCER produces smaller rule sets than C4.5 without pruning, CN2 (both ordered and unordered), and Slipper. On the other hand, Ripper produced (significantly) smaller rule sets in 7 out of 16 domains, and there were 8 draws and 1 win. A further investigation involving the data sets where most of the algorithms produced smaller rule sets than ROCCER (Breast, Heart, Ionosphere and Kr-vs-kp) might produce some in-

[1]Multiple comparison is used to adjust the observed significance level for the fact that multiple comparisons are made. If we use a t-test, and as each comparison has up to 5% Type I error, then the Type I error rate over the entire group can be much higher than 5%.

#	ROCCER	C45	C45NP	CN2	CN2OR	Ripper	Slipper	All
1	98.63(1.88)	97.76(1.51)	98.39(1.30)	99.26(0.81)	99.13(0.92)	98.72(1.38)	99.24 (0.57)	99.07(0.87)
2	65.30(7.93)	62.14(9.91)	57.44(11.92)	62.74(8.85)	62.21(8.11)	69.10(7.78)	59.84 (6.44)	65.38(10.63)
3	90.31(11.56)	50.00(0.00)	90.06(7.75)	90.17(6.90)	85.15(11.38)	61.86(25.49)	74.78 (15.94)	16.50(10.43)
4	61.83(24.14)	50.00(0.00)	68.68(17.22)	53.22(24.12)	42.78(24.43)	45.28(14.93)	52.35 (7.44)	62.11(23.96)
5	72.08(6.02)	71.43(5.89)	67.71(4.12)	75.25(5.38)	70.90(4.70)	64.02(13.62)	71.32 (6.20)	73.37(4.84)
6	79.45(12.98)	50.00(0.00)	81.50(12.65)	73.74(15.40)	79.64(13.24)	49.75(0.79)	50.00 (2.36)	35.62(18.93)
7	66.41(11.54)	55.84(6.14)	64.33(13.58)	59.83(9.87)	59.28(10.13)	57.45(3.85)	50.40 (11.14)	66.52(5.94)
8	85.78(8.43)	84.81(6.57)	81.11(7.91)	83.61(6.89)	82.25(6.59)	84.89(7.68)	84.03 (6.36)	90.72(6.28)
9	94.18(4.49)	86.09(9.97)	90.91(6.03)	96.23(2.97)	92.18(7.54)	92.06(5.94)	93.95 (6.82)	90.14(5.32)
10	99.35(0.36)	99.85(0.20)	99.86(0.20)	99.85(0.16)	99.91(0.17)	99.85(0.21)	99.91 (0.09)	92.67(1.60)
11	96.08(0.52)	95.49(1.96)	99.33(0.46)	99.34(0.28)	99.44(0.63)	97.27(1.86)	98.82(0.44)	92.45(1.54)
12	98.40(1.70)	87.85(10.43)	97.50(3.39)	99.14(1.19)	98.43(2.58)	94.95(9.94)	99.12 (1.25)	89.97(7.75)
13	97.85(0.44)	99.42(0.14)	99.74(0.13)	100.00(0.00)	99.99(0.01)	99.43(0.26)	94.40(1.59)	97.79(0.65)
14	70.68(5.09)	72.07(4.42)	72.60(6.50)	70.96(4.62)	71.97(5.44)	68.07(9.46)	70.02 (5.97)	70.37(5.01)
15	89.39(2.38)	90.15(1.70)	91.31(1.32)	91.48(1.45)	91.48(0.90)	86.83(3.94)	89.06 (1.98)	79.62(4.95)
16	96.42(1.47)	94.76(3.00)	96.99(1.44)	97.38(2.05)	96.49(2.41)	95.01(2.22)	93.99 (3.13)	93.37(3.05)
Avg	85.13	77.98	84.84	84.51	83.2	79.03	80.08	75.98

Table 2: AUC values estimated with 10-fold cross-validation on the 16 UCI data sets described in Table 1, obtained with ROCCER, C4.5, C4.5 without pruning, CN2 unordered, CN2 ordered (*i.e.*, learning decision lists), Ripper, Slipper, and bagging all rules found by Apriori. Numbers between brackets indicate standard deviations; dark gray indicates significant wins over ROCCER, and light gray indicates significant losses against ROCCER.

sights for improvements to our approach.

We conclude from Tables 2 and 3 that ROCCER combines, in a sense, the best of both worlds: it achieves AUC values that are comparable to those of unpruned decision trees and CN2, but without the large number of rules induced by those systems. Finally, Table 4 presents statistics of the individual rules that comprise the rule sets, which demonstrates another advantage of the ROCCER approach. Support ranges from 0 to 100% and is a measure of the relative coverage of each rule. Weighted relative accuracy (WRAcc) ranges from 0 to 0.25 and assesses the significance of a rule, in terms of difference between the observed and expected numbers of true positives. The odds ratio ranges from 0 to ∞ and is a measure of strength of association. It can clearly be seen that the rules selected by ROCCER have considerably higher values for all measures. This means that the rules are more meaningful in isolation, without reference to the other rules in the rule set. Thus, ROCCER successfully overcomes one of the main drawbacks of the set-covering approach.

	Support (%)	WRAcc	Odds Ratio
ROCCER	13.67 (13.89)	0.0355 (0.018)	154.02 (337.33)
C4.5	3.73 (6.01)	0.0094 (0.013)	44.55 (77.04)
C4.5NP	1.19 (1.06)	0.0030 (0.003)	31.90 (56.68)
CN2	3.90 (2.52)	0.0110 (0.009)	98.73 (138.64)
CN2OR	3.10 (2.18)	0.0085 (0.007)	95.07 (192.94)
Ripper	5.96 (5.34)	0.0184 (0.012)	74.08 (103.88)
Slipper	1.92 (1.58)	0.0060 (0.006)	33.86 (50.41)
All	8.07 (5.06)	0.0114 (0.014)	67.39 (111.72)

Table 4: Support, weighted relative accuracy and odds ratio averaged over all learned rules.

A final word should be said regarding computational complexity. ROCCER is, of course, computationally more expensive than the other algorithms. In the worst case, the complexity is $O(n^2)$, where n is the number of rules used as input. However, on average, the number of iterations is $\Omega(mn)$, where m is the number of rules selected by the algorithm. Due to lack of space we will not report runtime statistics for all data sets. For most datasets the runtime ranges from a few seconds to 10 minutes per fold (on a Pentium 4 2.4Ghz machine with 512MB of RAM). For these datasets, the number of rules used as input is up to 1,000. For some domains (Kr-vs-kp and Satimage) the number of rules generated by Apriori is very high (more than 40,000). The runtime in these cases is on average nearly 1.5 hours per fold.

5 Conclusion

We presented ROCCER, a rule selection algorithm based on ROC analysis. ROCCER operates by selecting rules from a larger set of rules by maintaining a ROC convex hull in the ROC space. Featuers of ROCCER's approach include implicit backtracking and discovery of pairs of related rules. Experimental results demonstrate AUC values that are compatible with the best probability predictors such as unpruned decision trees, achieved with considerably smaller rule sets. The rules that compose the rule sets induced by ROCCER have also higher values of support, weighted relative accuracy and odds ratio, and thus are more meaningful as individual rules.

Acknowledgments

This work is partially supported by the Brazilian Research Concil CAPES (Process no. BEX1340/03-8), and was carried out while the first author was visiting the University of Bristol.

References

[Blake and Merz, 1998] C. L. Blake and C. J. Merz. UCI repository of machine learning databases,

#	ROCCER	C4.5	C4.5NP	CN2	CN2OR	Ripper	Slipper	All
1	48.4(2.32)	37.8(12.62)	104.2(9.58)	32.8(2.74)	24.3(2.71)	16.9(1.1)	36.7(10.78)	502.1(8.96)
2	3.9(0.99)	15(10.53)	143.3(13.12)	100.7(3.77)	83.6(5.87)	7.5(1.84)	29.3(8.67)	292.8(21.57)
3	6.7(0.82)	0(0)	62(10.46)	35.3(2.83)	33.6(3.13)	4.9(4.33)	15.2(6.73)	27.5(6.05)
4	1.7(2.26)	0(0)	35.7(7.41)	20.2(1.62)	13.4(2.12)	1.3(0.48)	1.6(3.53)	827.4(254.92)
5	23.7(6.75)	78.2(18.5)	388.6(19.07)	143.9(6.98)	106.8(4.37)	8.9(3.25)	37.4(12.42)	2886.1(577.3)
6	2.4(0.52)	0(0)	32.1(5.99)	21.6(1.26)	21.9(3.18)	0.2(0.42)	2(3.43)	26.7(12.1)
7	0.8(0.42)	5.6(5.72)	71.2(9.39)	75.9(3.9)	57(5.98)	3.5(0.97)	11.8(13.21)	21.2(1.4)
8	68.2(4.42)	13.2(4.49)	97.4(15.04)	42.8(2.49)	36.1(1.79)	7.7(1.25)	27.2(8.73)	1875.6(91.9)
9	67.1(5.38)	23.4(3.98)	128.4(14.1)	36.9(2.28)	22.9(1.29)	19.6(4.43)	38.8(10.43)	20855.9(2690.9)
10	43.6(7.97)	29.2(2.15)	37.4(3.5)	30.1(1.85)	28.1(1.73)	27.3(1.7)	61.2(9.87)	41790.8(565.72)
11	78.4(3.06)	183.8(15.22)	700.4(31.76)	126.3(3.56)	120.2(2.7)	71(6.58)	115.2(6.16)	649.1(20.55)
12	8.5(0.53)	4(0)	35.4(4.2)	18.7(0.67)	15.3(0.82)	10.9(3.21)	20.7(5.48)	39.1(4.72)
13	18.9(1.66)	114.6(3.1)	227.1(3.57)	112.4(2.27)	17.6(0.52)	44.1(7.32)	57.6(2.37)	100.0(4.29)
14	4(0.82)	49.4(20.27)	347.4(10)	169.7(10.08)	168.4(7.9)	8.7(2.21)	30.5(10.82)	10.7(3.62)
15	143.9(51.28)	531.1(84.63)	1767.2(111.93)	158.8(6.09)	199.8(6.99)	32(4.37)	52.3(13.9)	4451.2(712.9)
16	48.4(4.48)	89.5(12.12)	188.9(9.42)	49.8(3.29)	41.2(2.82)	23.7(4.06)	75.7(12.11)	431.4(53.85)
Avg	35.54	73.43	272.92	73.49	61.89	18.01	38.33	4674.23

Table 3: Average numbers of rules obtained with the experiments reported in Table 2.

1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[Borgelt and Kruse, 2002] C. Borgelt and R. Kruse. Induction of association rules: A priori implementation. In *Proc. 15th Conf. on Computational Statistics*, 2002.

[Clark and Boswell, 1991] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Proc. 5th European Conf. on Machine Learning*, volume 482 of *LNAI*, pages 151–163. Springer-Verlag, 1991.

[Clark and Niblett, 1989] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[Cohen and Singer, 1999] W. Cohen and Y. Singer. A simple, fast and effective rule learner. In *Proc. 16th Nat. Conf. on Artificial Intelligence*, pages 335–342. AAAI/MIT Press, 1999.

[Cohen, 1995] W. Cohen. Fast effective rule induction. In *Proc. 12th Int. Conf. on Machine Learning*, pages 115–123, 1995.

[Domingos, 1996] P. Domingos. Unifying instance-based and rule-based induction. *Machine Learning*, 24(2):141–168, 1996.

[Ferri *et al.*, 2004] C. Ferri, P. Flach, and J. Hernández. Delegating classifiers. In *Proc. 21st Int. Conf. on Machine Learning*, pages 289–296, 2004.

[Fürnkranz and Flach, 2005] J. Fürnkranz and P. Flach. ROC'n'rule learning – toward a better understanding of rule covering algorithms. *Machine Learning*, 58(1):39–77, 2005.

[Fürnkranz and Widmer, 1994] J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In *Proc. 11th Int. Conf. on Machine Learning*, pages 70–77. Morgan Kaufmann, 1994.

[Fürnkranz, 1999] J. Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, February 1999.

[Gamberger and Lavrač, 2000] D. Gamberger and N. Lavrač. Confirmation rule sets. In *Proc. 4th European Conf. on Principles of Data Mining and Knowledge Discovery*, volume 1910 of *LNAI*, pages 34–43. Springer-Verlag, 2000.

[Javanoski and Lavrač, 2001] V. Javanoski and N. Lavrač. Classification rule learning with Apriori-C. In *Proc. 10th Portuguese Conf. on Artificial Intelligence*, volume 2258 of *LNAI*, pages 44–52, Porto, Portugal, 2001. Springer-Verlag.

[Kavšek *et al.*, 2003] B. Kavšek, N. Lavrač, and V. Javanoski. Apriori-SD: Adapting association rule learning to subgroup discovery. In *Proc. 5th Intelligent Data Analysis*, volume 2810 of *LNCS*, pages 230–241. Springer Verlag, 2003.

[Lavrač *et al.*, 2004] N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski. Subgroup discovery with CN2-SD. *Journal of Machine Learning Research*, 5:153–188, 2004.

[Liu *et al.*, 1998] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining*, pages 80–86, New York, USA, 1998.

[Provost and Domingos, 2003] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[Provost and Fawcett, 2001] F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231, 2001.

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

[van den Eijkel, 2003] G. van den Eijkel. Information-theoretic tree and rule induction. In *Intelligent Data Analysis*, pages 465–475. Springer-Verlag, 2003.

[Weiss and Indurkhya, 2000] S. M. Weiss and N. Indurkhya. Lightweight rule induction. In *Proc. 17th Int. Conf. on Machine Learning*, pages 1135–1142, 2000.