Generating Accurate Rule Sets Without Global Optimization

Eibe Frank
Department of Computer Science
University of Waikato
Hamilton, New Zealand
eibe@cs.waikato.ac.nz

Ian H. Witten
Department of Computer Science
University of Waikato
Hamilton, New Zealand
ihw@cs.waikato.ac.nz

Abstract

The two dominant schemes for rule-learning, C4.5 and RIPPER, both operate in two stages. First they induce an initial rule set and then they refine it using a rather complex optimization stage that discards (C4.5) or adjusts (RIPPER) individual rules to make them work better together. In contrast, this paper shows how good rule sets can be learned one rule at a time, without any need for global optimization. We present an algorithm for inferring rules by repeatedly generating partial decision trees, thus combining the two major paradigms for rule generation—creating rules from decision trees and the separate-and-conquer rule-learning technique. The algorithm is straightforward and elegant: despite this, experiments on standard datasets show that it produces rule sets that are as accurate as and of similar size to those generated by C4.5, and more accurate than RIPPER's. Moreover, it operates efficiently, and because it avoids postprocessing, does not suffer the extremely slow performance on pathological example sets for which the C4.5 method has been criticized.

1 Introduction

If-then rules are the basis for some of the most popular concept description languages used in machine learning. They allow "knowledge" extracted from a dataset to be represented in a form that is easy for people to understand. This gives domain experts the chance to analyze and validate that knowledge, and combine it with previously known facts about the domain.

A variety of approaches to learning rules have been investigated. One is to begin by generating a decision tree, then to transform it into a rule set, and finally to simplify the rules (Quinlan, 1987a); the resulting rule set is often more accurate than the original tree. Another is to use the "separate-and-conquer" strategy (Pagallo & Haussler, 1990) first applied in the AQ family of algorithms (Michalski, 1969) and subsequently used as the basis of many rule learning systems (Fürnkranz, 1996). In essence, this strategy determines the most powerful rule that underlies the dataset, separates out those examples that are covered by it, and repeats the procedure on the remaining examples.

Two dominant practical implementations of rule-learners have emerged from these strands of research: C4.5 (Quinlan, 1993) and RIPPER (Cohen, 1995). Both perform a global optimization process on the set of rules that is induced initially. The motivation for this in C4.5 is that the initial rule set, being generated from a decision tree, is unduly large and redundant: C4.5 drops some individual rules (having previously optimized rules locally by dropping conditions from them). The motivation in RIPPER, on the other hand, is to increase the accuracy of the rule set by replacing or revising individual rules. In either case the two-stage nature of the algorithm remains: as Cohen (1995) puts it, "... both RIPPER*k* and C4.5rules start with an initial model and iteratively improve it using heuristic techniques." Experiments show that both the size and the performance of rule sets are significantly improved by post-induction optimization. On the other hand, the process itself is rather complex and heuristic.

This paper presents a rule-induction procedure that avoids global optimization but nevertheless produces accurate, compact rule sets. The method combines the two rule learning paradigms identified above. Section 2 discusses these two paradigms and their incarnation in C4.5 and RIPPER. Section 3 presents the new algorithm, which we call "PART" because it is based on partial decision trees. Section 4 describes an experimental evaluation on standard datasets comparing PART to C4.5, RIPPER, and C5.0, the commercial successor of C4.5.[1] Section 5 summarizes our findings.

2 Related Work

We review two basic strategies for producing rule sets. The first is to begin by creating a decision tree and then transform it into a rule set by generating one rule for each path from the root to a leaf. Most rule sets derived in this way can be simplified dramatically without losing predictive accuracy. They are unnecessarily complex because the disjunctions that they imply can often not be expressed succinctly in a decision tree. This is sometimes known as the "replicated subtree" problem (Pagallo & Haussler, 1990).

When obtaining a rule set, C4.5 first transforms an unpruned decision tree into a set of rules in the aforementioned way. Then each rule is simplified separately by greedily deleting conditions in order to minimize the rule's estimated error rate. Following that, the rules for each class in turn are considered and a "good" subset is sought, guided by a criterion based on the minimum description length principle (Rissanen, 1978) (this is performed greedily, replacing an earlier method that used simulated annealing). The next step ranks the subsets for the different classes with respect to each other to avoid conflicts, and determines a default class. Finally, rules are greedily deleted from the whole rule set one by one, so long as this decreases the rule set's error on the training data.

The whole process is complex and time-consuming. Five separate stages are required to produce the final rule set. It has been shown that for noisy datasets, runtime is cubic in the number of instances (Cohen, 1995). Moreover, despite the lengthy optimization process, rules are still restricted to conjunctions of those attribute-value tests that occur along a path in the initial decision tree.

Separate-and-conquer algorithms represent a more direct approach to learning decision rules. They generate one rule at a time, remove the instances covered by that rule, and iteratively induce further rules for the remaining instances. In a multi-class setting, this automatically leads to an ordered list of rules, a type of classifier that has been termed a "decision list" (Rivest, 1987). Various different pruning methods for separate-and-conquer algorithms have been investigated by Fürnkranz (1997), who shows that the most effective scheme is to prune each rule back immediately after it is generated, using a separate stopping criterion to determine when to cease adding rules (Fürnkranz & Widmer, 1994). Although originally formulated for two-class problems, this procedure can be applied directly to multi-class settings by building rules separately for each class and ordering them appropriately (Cohen, 1995).

RIPPER implements this strategy using reduced error pruning (Quinlan, 1987b), which sets some training data aside to determine when to drop the tail of a rule, and incorporates a heuristic based on the minimum description length principle as stopping criterion. It follows rule induction with a post-processing step that revises the rule set to more closely approximate what would have been obtained by a more expensive global pruning strategy. To do this, it considers "replacing" or "revising" individual rules, guided by the error of the modified rule set on the pruning data. It then decides whether to leave the original rule alone or substitute its replacement or revision, a decision that is made according to the minimum description length heuristic. It has been claimed (Cohen, 1995) that RIPPER generates rule sets that are as accurate as C4.5's. However, our experiments on a large collection of standard datasets—reported in Section 3—do not confirm this.

As the following example shows, the basic strategy of building a single rule and pruning it back can lead to a particularly problematic form of overpruning, which we call "hasty generalization." This is because the pruning interacts with the covering heuristic. Generalizations are made before their implications are known, and the covering heuristic then prevents the learning algorithm from discovering the implications.

Here is a simple example of hasty generalization. Consider a Boolean dataset with attributes a and b built from the three rules in Figure 1, corrupted by ten percent class noise. Assume that the pruning operator is conservative and can only delete a single final conjunction of a rule at a time (not an entire tail of conjunctions as RIPPER does). Assume further that the first

[1] A test version of C5.0 is available from http://www.rulequest.com.

Rule	Coverage			
	Training Set		Pruning Set	
	$\oplus$	$\ominus$	$\oplus$	$\ominus$
1: a = true $\Rightarrow \oplus$	90	8	30	5
2: a = false $\wedge$ b = true $\Rightarrow \oplus$	200	18	66	6
3: a = false $\wedge$ b = false $\Rightarrow \ominus$	1	10	0	3

Figure 1: A hypothetical target concept for a noisy domain.

rule has been generated and pruned back to

$$a = true \Rightarrow \oplus$$

(The training data in Figure 1 is solely to make this scenario plausible.) Now consider whether the rule should be further pruned. Its error rate on the pruning set is 5/35, and the null rule

$$\Rightarrow \oplus$$

has an error rate of 14/110, which is smaller. Thus the the rule set will be pruned back to this single, trivial, rule, instead of the patently more accurate three-rule set shown in Figure 1.

Hasty generalization is not just an artifact of reduced error pruning: it can happen with pessimistic pruning (Quinlan, 1993) too. Because of variation in the number of noisy instances in the data sample, one can always construct situations in which pruning causes rules with comparatively large coverage to swallow rules with smaller (but still significant) coverage. This can happen whenever the number of errors committed by a rule is large compared with the total number of instances covered by an adjacent rule.

3 Obtaining Rules From Partial Decision Trees

The new method for rule induction, PART, combines the two approaches discussed in Section 2 in an attempt to avoid their respective problems. Unlike both C4.5 and RIPPER it does not need to perform global optimization to produce accurate rule sets, and this added simplicity is its main advantage. It adopts the separate-and-conquer strategy in that it builds a rule, removes the instances it covers, and continues creating rules recursively for the remaining instances until none are left. It differs from the standard approach in the way that each rule is created. In essence, to make a single rule a pruned decision tree is built for the current set of instances, the leaf with the largest coverage is made into a rule, and the tree is discarded. This avoids hasty generalization by only generalizing once the implications are known (i.e., all the subtrees have been expanded).

The prospect of repeatedly building decision trees only to discard most of them is not as bizarre as it first seems. Using a pruned tree to obtain a rule instead of building it incrementally by adding conjunctions one at a time avoids the over-pruning problem of the basic separate-and-conquer rule learner. Using the separate-and-conquer methodology in conjunction with decision trees adds flexibility and speed. It is indeed wasteful to build a full decision tree just to obtain a single rule, but the process can be accelerated significantly without sacrificing the above advantages.

The key idea is to build a "partial" decision tree instead of a fully explored one. A partial decision tree is an ordinary decision tree that contains branches to undefined subtrees. To generate such a tree, we integrate the construction and pruning operations in order to find a "stable" subtree that can be simplified no further. Once this subtree has been found, tree-building ceases and a single rule is read off.

The tree-building algorithm is summarized in Figure 2: it splits a set of examples recursively into a partial tree. The first step chooses a test and divides the examples into subsets accordingly. Our implementation makes this choice in exactly the same way as C4.5. Then the subsets are expanded in order of their average entropy, starting with the smallest. (The reason for this is that subsequent subsets will most likely not end up being expanded, and the subset with low average entropy is more likely to result in a small subtree and therefore produce a more general rule.) This continues recursively until a subset is expanded into a leaf, and

Procedure Expand Subset

choose split of given set of examples into subsets
while there are subsets that have not been expanded **and**
        all the subsets expanded so far are leaves
    choose next subset to be expanded and expand it
if all the subsets expanded are leaves **and**
    estimated error for subtree $\geq$ estimated error for node
    undo expansion into subsets and make node a leaf

Figure 2: Method that expands a given set of examples into a partial tree

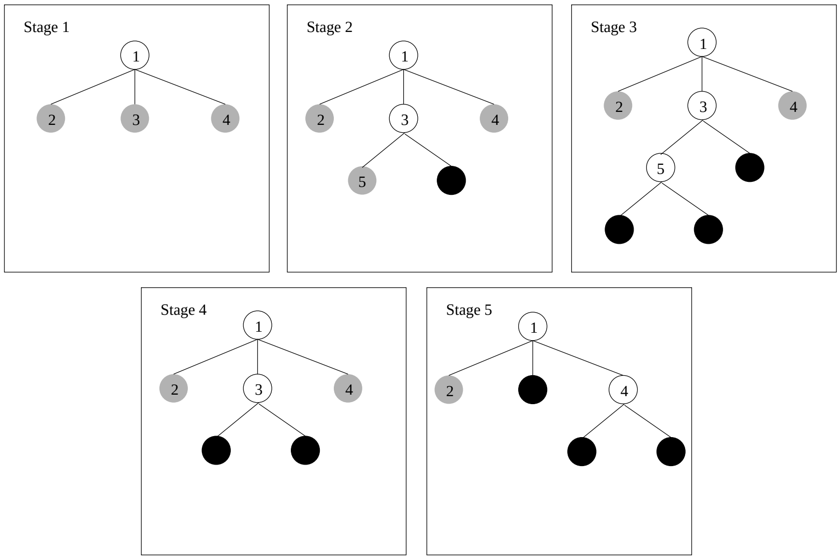

Figure 3: Example of how our algorithm builds a partial tree

then continues further by backtracking. But as soon as an internal node appears which has all its children expanded into leaves, pruning begins: the algorithm checks whether that node is better replaced by a single leaf. This is just the standard "subtree replacement" operation of decision-tree pruning, and our implementation makes the decision in exactly the same way as C4.5. (C4.5's other pruning operation, "subtree raising," plays no part in our algorithm.) If replacement is performed the algorithm backtracks in the standard way, exploring siblings of the newly-replaced node. However, if during backtracking a node is encountered all of whose children are not leaves—and this will happen as soon as a potential subtree replacement is *not* performed—then the remaining subsets are left unexplored and the corresponding subtrees are left undefined. Due to the recursive structure of the algorithm this event automatically terminates tree generation.

Figure 3 shows a step-by-step example. During stages 1–3, tree-building continues recursively in the normal way—except that at each point the lowest-entropy sibling is chosen for expansion: node 3 between stages 1 and 2. Gray nodes are as yet unexpanded; black ones are leaves. Between Stages 2 and 3, the black node will have lower entropy than its sibling, node 5; but cannot be expanded further since it is a leaf. Backtracking occurs and node 5 is chosen for expansion. Once stage 3 is reached, there is a node—node 5—which has all of its children expanded into leaves, and this triggers pruning. Subtree replacement for node 5 is considered, and accepted, leading to stage 4. Now node 3 is considered for subtree replacement, and this operation is again accepted. Backtracking continues, and node 4, having lower entropy than 2, is expanded—into two leaves. Now subtree replacement is considered for node 4: let us suppose that node 4 is not replaced. At this point, the process effectively terminates with the 3-leaf

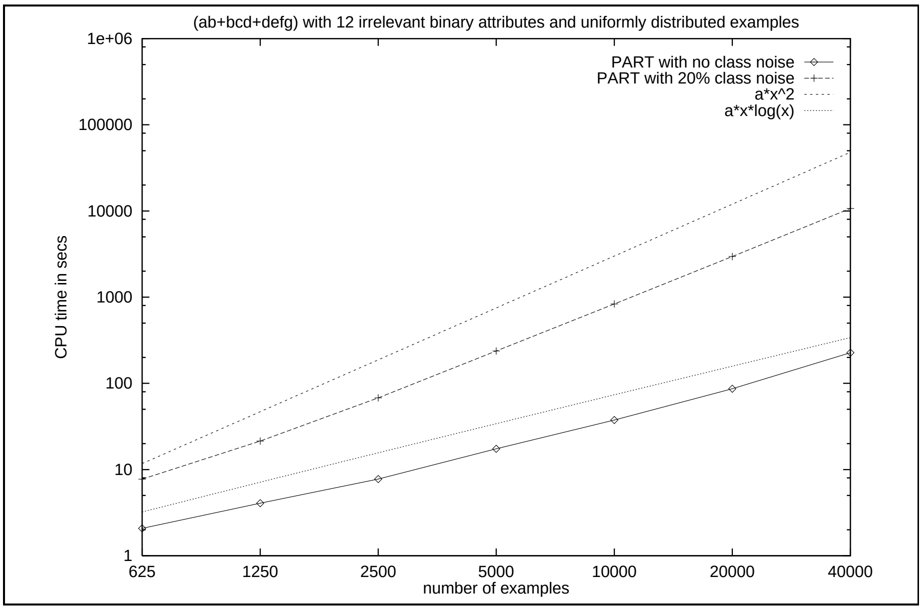

Figure 4: CPU times for PART on artificial dataset

partial tree of stage 5.

This procedure ensures that the over-pruning effect discussed in Section 2 cannot occur. A node can only be pruned if all its successors are leaves. This can only happen if all its subtrees have been explored and either found to be leaves, or are pruned back to leaves. Situations like that shown in Figure 1 are therefore handled correctly.

If a dataset is noise-free and contains enough instances to prevent the algorithm from doing any pruning, just one path of the full decision tree has to be explored. This achieves the greatest possible performance gain over the naive method that builds a full decision tree each time. The gain decreases as more pruning takes place. For datasets with numeric attributes, the asymptotic time complexity of the algorithm is the same as for building the full decision tree[2] because in this case the complexity is dominated by the time needed to sort the attribute values in the first place.

Once a partial tree has been built, a single rule is extracted from it. Each leaf corresponds to a possible rule, and we seek the "best" leaf of those subtrees (typically a small minority) that have been expanded into leaves. Our implementation aims at the most general rule by choosing the leaf that covers the greatest number of instances. (We have experimented with choosing the most accurate rule, that is, the leaf with the lowest error rate, error being estimated according to C4.5's Bernoulli heuristic, but this does not improve the rule set's accuracy.)

Datasets often contain missing attribute values, and practical learning schemes must deal with them efficiently. When constructing a partial tree we treat missing values in exactly the same way as C4.5: if an instance cannot be assigned deterministically to a branch because of a missing attribute value, it is assigned to each of the branches with a weight proportional to the number of training instances going down that branch, normalized by the total number of training instances with known values at the node. During testing we apply the same procedure separately to each rule, thus associating a weight with the application of each rule to the test instance. That weight is deducted from the instance's total weight before it is passed to the next rule in the list. Once the weight has reduced to zero, the predicted class probabilities are combined into a final classification according to the weights.

The algorithm's runtime depends on the number of rules it generates. Because a decision tree can be built in time $O(an \log n)$ for a dataset with n examples and a attributes, the time taken to generate a rule set of size k is $O(kan \log n)$. Assuming (as the analyses of (Cohen, 1995) and (Fürnkranz, 1997) do) that the size of the final theory is constant, the overall time complexity is $O(an \log n)$, as compared to

[2] Assuming no subtree raising.

Table 1: Datasets used for the experiments

Dataset	Instances	Missing values (%)	Numeric attributes	Nominal attributes	Classes
anneal	898	0.0	6	32	5
audiology	226	2.0	0	69	24
australian	690	0.6	6	9	2
autos	205	1.1	15	10	6
balance-scale	625	0.0	4	0	3
breast-cancer	286	0.3	0	9	2
breast-w	699	0.3	9	0	2
german	1000	0.0	7	13	2
glass (G2)	163	0.0	9	0	2
glass	214	0.0	9	0	6
heart-c	303	0.2	6	7	2
heart-h	294	20.4	6	7	2
heart-statlog	270	0.0	13	0	2
hepatitis	155	5.6	6	13	2
horse-colic	368	23.8	7	15	2
hypothyroid	3772	5.5	7	22	4
ionosphere	351	0.0	34	0	2
iris	150	0.0	4	0	3
kr-vs-kp	3196	0.0	0	36	2
labor	57	3.9	8	8	2
lymphography	148	0.0	3	15	4
mushroom	8124	1.4	0	22	2
pima-indians	768	0.0	8	0	2
primary-tumor	339	3.9	0	17	21
segment	2310	0.0	19	0	7
sick	3772	5.5	7	22	2
sonar	208	0.0	60	0	2
soybean	683	9.8	0	25	19
splice	3190	0.0	0	61	3
vehicle	846	0.0	18	0	4
vote	435	5.6	0	16	2
vowel	990	0.0	10	3	11
waveform-noise	5000	0.0	40	0	3
zoo	101	0.0	1	15	7

$O(an \log^2 n)$ for RIPPER. In practice, the number of rules grows with the size of the training data because of the greedy rule learning strategy and pessimistic pruning. However, even in the worst case when the number of rules increases linearly with training examples, the overall complexity is bounded by $O(an^2 \log n)$. In our experiments we only ever observed subquadratic run times—even for the artificial dataset that Cohen (1995) used to show that C4.5's performance can be cubic in the number of examples. The results of timing our method, PART, on this dataset are depicted on a log-log scale in Figure 4, for no class noise and for 20 percent class noise. In the latter case C4.5 scales as the cube of the number of examples.

4 Experimental Results

In order to evaluate the performance of PART on a diverse set of practical learning problems, we performed experiments on thirty-four standard datasets from the UCI collection (Merz & Murphy, 1996).[3] The datasets and their characteristics are listed in Table 1.

As well as the learning algorithm PART described above, we also ran C4.5,[4] C5.0 and RIPPER on all the datasets. The results are listed in Table 2. They give the percentage of correct classifications, averaged over ten ten-fold cross-validation runs, and standard

[3]Following Holte (Holte, 1993), the G2 variant of the *glass* dataset has classes 1 and 3 combined and classes 4 to 7 deleted, and the *horse-colic* dataset has attributes 3, 25, 26, 27, 28 deleted with attribute 24 being used as the class. We also deleted all identifier attributes from the datasets.

[4]We used Revision 8 of C4.5.

Table 2: Experimental results: percentage of correct classifications, and standard deviation

Dataset	PART	C4.5		C5.0		RIPPER	
anneal	98.4±0.3	98.6±0.2	†	98.7±0.3	∘	98.3±0.1	
audiology	78.7±1.1	76.3±1.2	•†	77.3±1.2	†	72.3±2.2	•
australian	84.3±1.2	84.8±1.1	•	85.4±0.7		85.3±0.7	
autos	74.5±1.4	76.5±2.9	†	79.1±2.1	∘†	72.0±2.0	•
balance-scale	82.3±1.2	78.0±0.7	•	79.0±1.0	•	81.0±1.1	
breast-cancer	69.6±1.6	70.3±1.6		73.6±1.6	∘	71.8±1.6	∘
breast-w	94.9±0.4	95.5±0.6	†	95.5±0.3	∘	95.6±0.7	
horse-colic	84.4±0.8	83.0±0.6	•	85.0±0.5		85.0±0.8	
german	70.0±1.4	71.9±1.4	∘	72.3±0.5	∘	71.4±0.7	∘
glass (G2)	80.0±3.6	79.4±2.3	†	80.2±1.8	†	80.9±1.4	
glass	70.0±1.6	67.3±2.4		68.4±2.8	†	66.7±2.1	•
heart-c	78.5±1.7	79.7±1.5		79.1±0.9		78.5±1.9	
heart-h	80.5±1.5	79.7±1.7		80.7±1.1		78.7±1.3	•
heart-statlog	78.9±1.3	81.2±1.3	∘	81.9±1.4	∘	79.0±1.4	
hepatitis	80.2±1.9	79.7±1.0	†	81.1±0.7		77.2±2.0	•
hypothyroid	99.5±0.1	99.5±0.1	†	99.5±0.0	•†	99.4±0.1	•
ionosphere	90.6±1.3	89.9±1.5	†	89.3±1.4	•†	89.2±0.8	•
iris	93.7±1.6	95.1±1.0	∘†	94.4±0.7	†	94.4±1.7	
kr-vs-kp	99.3±0.1	99.4±0.1	∘†	99.3±0.1		99.1±0.1	•
labor	77.3±3.9	81.4±2.6	∘†	77.1±3.7	†	83.5±3.9	∘
lymphography	76.5±2.7	78.0±2.2		76.8±2.7	†	76.1±2.4	
mushroom	100.0±0.0	100.0±0.0	•†	99.9±0.0	•†	100.0±0.0	
pima-indians	74.0±0.5	74.2±1.2	†	75.5±0.9	∘†	75.2±1.1	∘
primary-tumor	41.7±1.3	40.1±1.7	•	28.7±2.5	•	38.5±0.8	•
segment	96.6±0.4	96.1±0.3	•†	96.3±0.4	†	95.2±0.5	•
sick	98.6±0.1	98.4±0.2	•	98.4±0.1	•	98.3±0.2	•
sonar	76.5±2.3	74.4±2.9	†	75.3±2.2	†	75.7±1.9	
soybean	91.4±0.5	91.9±0.7		92.2±0.6	†	92.0±0.4	
splice	92.5±0.4	93.4±0.3	∘	94.3±0.3	∘	93.4±0.2	∘
vehicle	72.4±0.8	72.9±0.9		72.4±0.8	†	69.0±0.6	•
vote	95.9±0.6	95.9±0.6	†	96.0±0.6	†	95.6±0.3	
vowel	78.1±1.1	77.9±1.3	†	79.9±1.2	∘†	69.6±1.9	•
waveform-noise	78.0±0.5	76.3±0.4	•	79.4±0.5	∘	79.1±0.6	∘
zoo	92.2±1.2	90.9±1.2	•†	91.5±1.2	†	87.8±2.4	•

deviations of the ten are also shown. The same folds were used for each scheme.[5] Results for C4.5, C5.0 and RIPPER are marked with ∘ if they show significant improvement over the corresponding results for PART, and with • if they show significant degradation. (The † marks are discussed below.) Throughout, we speak of results being "significantly different" if the difference is statistically significant at the 1% level according to a paired two-sided *t*-test, each pair of data points consisting of the estimates obtained in one ten-fold cross-validation run for the two learning schemes being compared. Table 3 shows how the different methods compare with each other. Each entry indicates the number of datasets for which the method associated with its column is significantly more accurate than the method associated with its row.

We observe from Table 3 that PART outperforms C4.5 on nine datasets, whereas C4.5 outperforms PART on six. The chance probability of this distribution is 0.3 according to a sign test: thus there is only very weak evidence that PART outperforms C4.5 on a collection of datasets similar to the one we used. According to Table 3, PART is significantly less accurate than C5.0 on ten datasets and significantly more accurate on six. The corresponding probability for this distribution is 0.23, providing only weak evidence that C5.0 performs better than PART. For RIPPER the situation is different: PART outperforms it on fourteen datasets and performs worse on six. The probability for this distribution is 0.06, a value that provides fairly strong

[5]The results of PART and C5.0 on the hypothyroid data, and of PART and C4.5 on the mushroom data, are not in fact the same—they differ in the second decimal place.

evidence that PART outperforms RIPPER on a collection of datasets of this type.

Table 3: Results of paired t-tests (p=0.01): number indicates how often method in column significantly outperforms method in row

	PART	C4.5	C5.0	RIPPER
PART	–	**6**	**10**	**6**
C4.5	**9**	–	9	4
C5.0	**6**	5	–	4
RIPPER	**14**	10	12	–

As well as accuracy, the size of a rule set is important because it has a strong influence on comprehensibility. The † marks in Table 2 give information about the relative size of the rule sets produced: they mark learning schemes and datasets for which—on average—PART generates fewer rules (this never occurs for RIPPER). Compared to C4.5 and C5.0, the average number of rules generated by PART is smaller for eighteen datasets and larger for sixteen.

5 Conclusions

This paper has presented a simple, yet surprisingly effective, method for learning decision lists based on the repeated generation of partial decision trees in a separate-and-conquer manner. The main advantage of PART over the other schemes discussed is not performance but simplicity: by combining two paradigms of rule learning it produces good rule sets without any need for global optimization. Despite this simplicity, the method produces rule sets that compare favorably with those generated by C4.5 and C5.0, and are more accurate (though larger) than those produced by RIPPER.

An interesting question for future research is whether the size of the rule sets obtained by our method can be decreased by employing a stopping criterion based on the minimum description length principle, as is done in RIPPER, or by using reduced error pruning instead of pessimistic pruning.

Acknowledgements

We would like to thank the anonymous reviewers for their comments, which significantly improved the exposition. We would also like to thank all the people who have donated datasets to the UCI repository.

References

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning* (pp. 115–123). Morgan Kaufmann.

Fürnkranz, J. (1996). Separate-and-conquer rule learning. Technical Report TR-96-25, Austrian Research Institute for Artificial Intelligence, Vienna. [ftp://ftp.ai.univie.ac.at/papers/oefai-tr-96-25.ps.Z].

Fürnkranz, J. (1997). Pruning algorithms for rule learning. *Machine Learning*, *27(2)*, 139–171.

Fürnkranz, J. & Widmer, G. (1994). Incremental reduced error pruning. In *Proceedings of the 11th International Conference on Machine Learning* (pp. 70–77). Morgan Kaufmann.

Holte, R. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, *11*, 63–91.

Merz, C. J. & Murphy, P. M. (1996). *UCI Repository of Machine Learning Data-Bases.* Irvine, CA: University of California, Department of Information and Computer Science. [http://www.ics.uci.edu/∼mlearn/MLRepository.html].

Michalski, R. S. (1969). On the quasi-minimal solution of the covering problem. In *Proceedings of the 5th International Symposium on Information Processing (FCIP-69), Vol. A3 (Switching Circuits)* (pp. 125–128). Bled, Yugoslavia.

Pagallo, G. & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, *5(1)*, 71–99.

Quinlan, J. R. (1987a). Generating production rules from decision trees. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence* (pp. 304–307). Morgan Kaufmann.

Quinlan, J. R. (1987b). Simplifying decision trees. *International Journal of Man-Machine Studies*, *27*, 221–234.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Rissanen, J. (1978). Modelling by shortest data description. *Automatica*, *14*, 465–471.

Rivest, R. L. (1987). Learning decision lists. *Machine Learning*, *2*, 229–246.