From "Machine Learning: Proceedings of the Twelfth International Conference" (ML95)

Fast Effective Rule Induction

William W. Cohen
AT&T Bell Laboratories
600 Mountain Avenue Murray Hill, NJ 07974
wcohen@research.att.com

Abstract

Many existing rule learning systems are computationally expensive on large noisy datasets. In this paper we evaluate the recently-proposed rule learning algorithm IREP on a large and diverse collection of benchmark problems. We show that while IREP is extremely efficient, it frequently gives error rates higher than those of C4.5 and C4.5rules. We then propose a number of modifications resulting in an algorithm RIPPER*k* that is very competitive with C4.5rules with respect to error rates, but much more efficient on large samples. RIPPER*k* obtains error rates lower than or equivalent to C4.5rules on 22 of 37 benchmark problems, scales nearly linearly with the number of training examples, and can efficiently process noisy datasets containing hundreds of thousands of examples.

1 INTRODUCTION

Systems that learn sets of rules have a number of desirable properties. Rule sets are relatively easy for people to understand [Catlett, 1991], and rule learning systems outperform decision tree learners on many problems [Pagallo and Haussler, 1990; Quinlan, 1987; Weiss and Indurkhya, 1991]. Rule sets have a natural and familiar first order version, namely Prolog predicates, and techniques for learning propositional rule sets can often be extended to the first-order case [Quinlan, 1990; Quinlan and Cameron-Jones, 1993]. Certain types of prior knowledge can also be easily communicated to rule learning systems [Cohen, 1994; Pazzani and Kibler, 1992].

One weakness with rule learning systems is that they often scale relatively poorly with the sample size, particularly on noisy data [Cohen, 1993]. Given the prevalence of large noisy datasets in real-world applications, this problem is of critical importance. The goal of this paper is to develop propositional rule learning algorithms that perform efficiently on large noisy datasets, that extend naturally to first-order representations, and that are competitive in generalization performance with more mature symbolic learning methods, such as decision trees. The end product of this effort is the algorithm RIPPER*k*, which is competitive with C4.5rules with respect to error rates, scales nearly linearly with the number of training examples, and can efficiently process noisy datasets containing hundreds of thousands of examples.

2 PREVIOUS WORK

2.1 COMPLEXITY OF RULE PRUNING

Many of the techniques used in modern rule learners have been adapted from decision tree learning. Most widely-used decision tree learning systems use an *overfit-and-simplify* learning strategy to handle noisy data: a hypothesis is formed by first growing a complex tree which "overfits" the data, and then simplifying or *pruning* the complex tree [Quinlan, 1987; Mingers, 1989]. Usually (but not always) such pruning strategies improve error rates on unseen data when the training data is noisy [Quinlan, 1987; Mingers, 1989; Schaffer, 1992]. A variety of methods have been proposed to prune trees, but one effective technique is *reduced error pruning (REP)*. REP can be easily adapted to rule learning systems [Pagallo and Haussler, 1990; Brunk and Pazzani, 1991].

In REP for rules, the training data is split into a *growing set* and a *pruning set*. First, an initial rule set is formed that overfits the growing set, using some heuristic method. This overlarge rule set is then repeatedly simplified by applying one of a set of *pruning operators*; typical pruning operators would be to delete any single condition or any single rule. At each stage of simplification, the pruning operator chosen is the one that yields the greatest reduction of error on the pruning set. Simplification ends when applying any pruning operator would increase error on the pruning set.

REP for rules usually does improve generalization performance on noisy data [Pagallo and Haussler, 1990; Brunk and Pazzani, 1991; Weiss and Indurkhya, 1991; Cohen, 1993; Fürnkranz and Widmer, 1994]; however, it is computationally expensive for large datasets. In previous work [Cohen, 1993] we showed that REP requires $O(n^4)$ time, given sufficiently noisy data; in fact, even the initial phase of overfitting the training data requires $O(n^2)$ time. We then proposed an alternative overfit-and-simplify method called Grow that is competitive with REP with respect to error rates, and was an order of magnitude faster on a set of benchmark problems.

We also showed that Grow was asymptotically faster than REP on random data—if one assumes that Grow's hypothesis is approximately the same size as the target concept. However, Cameron-Jones [1994] later showed that Grow systematically overfits the target concept on noisy data. This has an adverse effect on Grow's time complexity and as a result Grow also requires $O(n^4)$ time asymptotically.

In another response to the inefficiency of REP, Fürnkranz and Widmer [1994] proposed a novel learning algorithm called *incremental reduced error pruning (IREP)*. IREP was shown experimentally to be competitive with both REP and Grow with respect to error rates, and much faster than either; in fact, on 18 of 20 benchmark problems, IREP was faster than the initial step of overfitting the data.

In this paper, we will take as our point of departure the promising results obtained by Fürnkranz and Widmer with the IREP algorithm. Our initial goal was simply to replicate their results, to evaluate IREP on a broader set of benchmarks, and to compare IREP to more mature tree and rule induction methods. In the course of doing this, we discovered that IREP's generalization performance could be considerably improved, without greatly affecting its computational efficiency. In the remainder of the paper we will describe our implementation of the original IREP algorithm, and give evidence that it affords room for improvement. We will then outline three modifications: a new metric for guiding its pruning phase, a new stopping condition, and a technique for "optimizing" the rules learned by IREP. Taken together these modifications give generalization performance that is comparable to C4.5 and C4.5rules [Quinlan, 1994] on a large set of diverse benchmarks. The modified learning algorithm, however, still scales well with the number of training examples. The current implementation can efficiently handle training sets of several hundred thousand examples.

```
procedure IREP(Pos,Neg)
begin
    Ruleset := ∅
    while Pos≠ ∅ do
        /* grow and prune a new rule */
        split (Pos,Neg) into (GrowPos,GrowNeg)
          and (PrunePos,PruneNeg)
        Rule := GrowRule(GrowPos,GrowNeg)
        Rule := PruneRule(Rule,PrunePos,PruneNeg)
        if the error rate of Rule on
          (PrunePos,PruneNeg) exceeds 50% then
            return Ruleset
        else
            add Rule to Ruleset
            remove examples covered by Rule
              from (Pos,Neg)
        endif
    endwhile
    return Ruleset
end
```

Figure 1: The IREP algorithm

2.2 INCREMENTAL REDUCED ERROR PRUNING

The IREP rule-learning algorithm is described in detail by Fürnkranz and Widmer [1994], but we will summarize it below. IREP tightly integrates reduced error pruning with a separate-and-conquer rule learning algorithm. Figure 1 presents a two-class version of this algorithm. (In the two-class Boolean case a "rule" is simply a conjunction of features, and a "rule set" is a DNF formula.) Like a standard separate-and-conquer algorithm, IREP builds up a rule set in a greedy fashion, one rule at a time. After a rule is found, all examples covered by the rule (both positive and negative) are deleted. This process is repeated until there are no positive examples, or until the rule found by IREP has an unacceptably large error rate.

In order to build a rule, IREP uses the following strategy. First, the uncovered examples are randomly partitioned into two subsets, a *growing set* and a *pruning set*. In our implementation, the growing set contains 2/3 of the examples.

Next, a rule is "grown". Our implementation of GrowRule is a propositional version of FOIL [Quinlan, 1990; Quinlan and Cameron-Jones, 1993]. It begins with an empty conjunction of conditions, and considers adding to this any condition of the form $A_n = v$, $A_c \leq \theta$, or $A_c \geq \theta$, where A_n is a nominal attribute and v is a legal value for A_n, or A_c is a continuous variable and θ is some value for A_c that occurs in the training data. GrowRule repeatedly adds the condition that maximizes FOIL's information gain criterion until the rule covers no negative examples from the growing dataset.

After growing a rule, the rule is immediately pruned. To prune a rule, our implementation considers deleting any final sequence of conditions from the rule, and chooses the deletion that maximizes the function

$$v(Rule, PrunePos, PruneNeg) \equiv \frac{p + (N - n)}{P + N} \quad (1)$$

where P (respectively N) is the total number of examples in *PrunePos* (*PruneNeg*) and p (n) is the number of examples in *PrunePos* (*PruneNeg*) covered by *Rule*. This process is repeated until no deletion improves the value of v.

The IREP algorithm described above is for two-class learning problems. Our implementation handles multiple classes as follows. First, the classes are ordered. In the experiments described below the ordering is always in increasing order of prevalence—*i.e.*, the ordering is $C_1, \ldots, C_k$ where C_1 is the least prevalent class and C_k is the most prevalent. Then, IREP is used to find a rule set that separates C_1 from the remaining classes; this is done with a single call to IREP where *PosData* contains the examples labeled C_1 and *NegData* contains the examples labeled C_2, C_3, ..., or C_k. Next, all instances covered by the learned rule set are removed from the dataset, and IREP is used to separate C_2 from classes $C_3, \ldots, C_k$. This process is repeated until a single class C_k remains; this class will be used as the default class.

We also extended the rule learning algorithm to handle missing attributes as follows: all tests involving the attribute A are defined to fail on instances for which the value of A is missing. This encourages the learner to separate out the positive examples using tests that are known to succeed.

2.3 DIFFERENCES FROM FÜRNKRANZ AND WIDMER'S IREP

This implementation differs from Fürnkranz and Widmer's in several details. In pruning rules, our implementation allows deletions of any final sequence of conditions, whereas Fürnkranz and Widmer's implementation allows only deletions of a single final condition. Our implementation also stops adding rules to a rule set when a rule is learned that has error rate greater than 50%, whereas Fürnkranz and Widmer's implementation stops when the accuracy of the rule is less than the accuracy of the empty rule.[1]

More importantly, our implementation supports missing attributes, numerical variables and multiple classes. This makes it applicable to a wider range of benchmark problems.

3 EXPERIMENTS WITH IREP

Experiments with IREP showed that it is indeed fast. Results for one representative artificial problem[2] are summarized in the first graph in Figure 2; the CPU time needed by C4.5rules is also shown.[3] The results are shown on a log-log scale; recall that polynomials appear as lines on such a plot, with the slope of the line indicating its degree. C4.5rules scales roughly as the cube of the number of examples, whereas IREP scales almost linearly. Extrapolating the curves suggests that it would require about 79 CPU years for C4.5rules to process the 500,000 example dataset, which IREP handles in around seven CPU minutes.

Although we have used an artificial concept with an extremely large number of training examples to demonstrate these issues, similar performance issues also arise on natural datasets, as the two smaller graphs of Figure 2 demonstrate.

For reference, the first graph in Figure 2 also shows the curves kx^3 and $y = kx \log^2 x$. Fürnkranz and Widmer's formal analysis of IREP predicts a running time of $O(m \log^2 m)$, where m is the number of examples, on

[1] Actually, Fürnkranz and Widmer described two pruning algorithms. The first, which they called IREP, prunes according to Equation 1, and stops when $p/(p+n) < N/(P+N)$. The second, which they called IREP2, prunes according to the metric $v(Rule, PrunePos, PruneNeg) \equiv \frac{p}{p+n}$ and stops when $p/(p+n) < 1/2$. Our experiments confirmed the conclusion of Fürnkranz and Widmer that IREP generally outperforms IREP2; however, we also discovered that IREP's performance was noticibly improved by adopting IREP2's stopping condition.

[2] The concept $ab \vee bcd \vee defg$ with 12 irrelevant binary attributes, 20% classification noise, and uniformly distributed examples. CPU time was measured on a MIPS Irix 5, configured with 8 150 MHz R4400 processors and 1Gb of memory. Since IREP is a randomized algorithm (because of its random partitioning of the examples) the curve for IREP is the average of 10 trials.

[3] The time for C4.5rules ignores the time needed to run C4.5. However, C4.5 is generally much faster than C4.5rules; on this problem, C4.5 requires less than 400 CPU seconds to handle the 500,000 example dataset. The run-time of C4.5 is generally comparable to that of IREP.

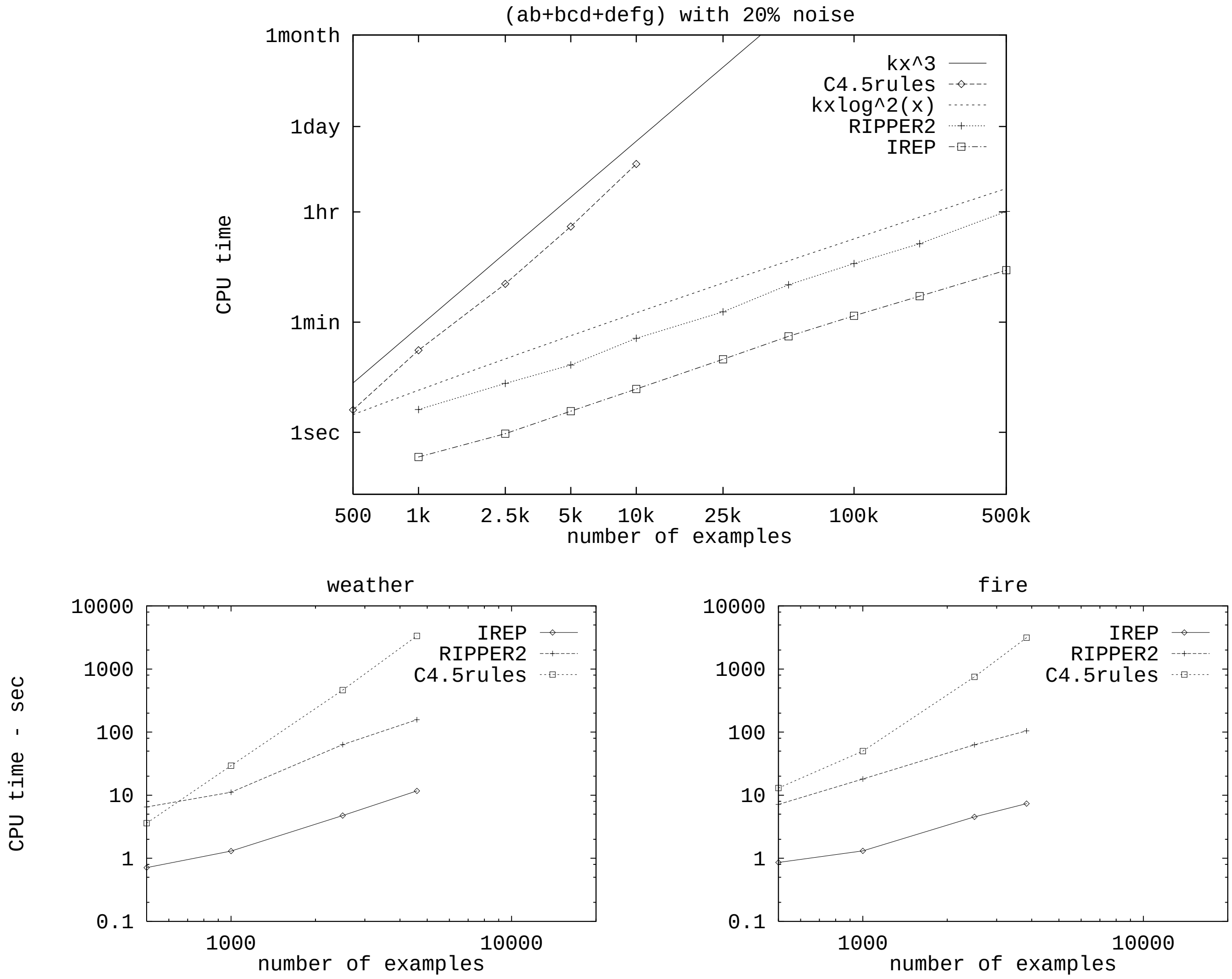

Figure 2: CPU times for C4.5rules, IREP, and RIPPER2

any dataset that contains a fixed percentage of classification noise. Our results are consistent with this prediction. Analysis similar to Fürnkranz and Widmer's also predicts the cubic behavior shown by C4.5rules.

Although IREP is efficient, experiments on real-world datasets showed that the generalization performance of IREP offered substantial room for improvement. We compared IREP to C4.5 and C4.5rules on a diverse set of benchmark problems, summarized in Table 1. Where a test set associated with the benchmark is indicated, we ran C4.5 and C4.5rules once, and ran IREP 10 times and averaged. Where no test set is indicated, we ran 10 different 10-fold cross-validations for all the algorithms and averaged the results. Due to space considerations we will focus on comparisons to C4.5rules, since it also learns rule sets; however, the performance of C4.5 and C4.5rules on these datasets was similar. We used C4.5 Release 6 [Quinlan, 1994], and the most recent version of C4.5rules [Quinlan, 1995].

The left-hand graph of Figure 3 contains one point for each benchmark problem, positioned so that IREP's error rate is the x-axis position and C4.5rules' error rate is the y-axis position. Thus for points below the line $y = x$ IREP's performance is inferior to C4.5rules, and for points above the line IREP's performance is better. From the graph one can readily see that IREP does worse than C4.5rules more often than it does better; specifically, IREP's error rate is higher 23 times, lower 11 times, and the same 3 times.

Of course, it may be that IREP is in fact as likely to outperform C4.5rules as the converse on problems from this test suite, and that the won-lost-tie ratio of 11-23-3 is due to random variation in the error esti-

Table 1: The 37 benchmark problems used in the experiments, with size of training and testing sets; number of classes; number of nominal (n) and continuous (c) attributes; and a brief description. Starred problems are from the UC/Irvine Repository.

Name	Train	Test	Classes	Attributes		Description
AP1-10	999	–	2	85-130n		text categorization (10 problems)
audiology*	226	–	24	60n		medical diagnosis
bridges1-5*	106	–	2-6	6n	1c	mech. engineering (5 problems)
iris*	150	–	3		4c	flower classification
labor*	57	–	2	8n	8c	labor negotiations
promoters*	106	–	2	57n		DNA promoter sequences
sonar*	208	–	2		60c	sonar signal classification
ticket1-3	556	–	2	78n		text categorization (3 problems)
ui	373	–	18	10n		text-to-speech subproblem
coding1*	5000	15000	2	15n		DNA coding sequences
fire	3225	608	8		11c	risk of forest fires
market	3181	1616	2	3n	7c	market analysis
mushroom*	3988	4136	2	22n		random split of mushroom data
netwk1	2500	1077	2		30c	predict equipment failure
netwk2	2600	1226	2		35c	predict equipment failure
ocr	1318	1370	2	576n		image classification
segment*	1133	1177	7	19n		image analysis
splice*	1614	1561	3	60n		split of DNA splice-junction data
thyroid*	2514	1258	5	22n	7c	medical diagnosis
vidgame	1484	1546	2	10n		decide if game moves are random
voting*	300	135	2	16n		congressional voting records
weather	1000	4597	2		35c	weather prediction

mates. Using a nonparametric sign test [Mendenhall *et al.*, 1981, page 578], one can determine that the probability of observing a ratio this one-sided would be just under 0.05 if IREP had a 50/50 chance of bettering C4.5rules on problems in this test suite. We can thus conclude with 95% confidence that C4.5rules outperforms IREP on this test suite.[4]

It is also evident from the graph that IREP seldom does much better than C4.5rules, and not infrequently does much worse. It is not obvious how to best aggregate measurements across learning problems, but one method is to consider the average value of the ratio

$$\frac{\textit{error rate of IREP}}{\textit{error rate of C4.5rules}}$$

For this set of problems the average of this ratio is 1.13, if one discounts a single extreme outlier; thus on average IREP's error rates are about 13% higher than those of C4.5rules. (This average is 1.52 if one includes the *mushroom* dataset—on this benchmark C4.5rules obtains an error of 0.2% to IREP's 3.1%.)

As an additional point of reference, we also ran propositional FOIL without any pruning mechanism. The ratio of the error rate of the hypothesis obtained by "overfitting" the data with propositional FOIL to the error rate of C4.5rules is 1.17 excluding the *mushroom* dataset, and 1.14 overall. Finally, we ran IREP2 (also described by Fürnkranz and Widmer [1994]) and IREP with Fürnkranz and Widmer's stopping condition. The average ratio for IREP2 was 1.15 without the *mushroom* dataset, and 1.14 overall. For IREP with the more restrictive Fürnkranz and Widmer stopping condition, the average ratio was 1.71 without *mushroom* and 2.08 overall. The best won-loss-tied record of any of these three systems relative to C4.5rules was 17-20-0, achieved by propositional FOIL without pruning. To summarize, on average, all of the IREP variants performed substantially worse than C4.5rules, and none of the IREP variants performed substantially better than simply overfitting the data.

There is also evidence that IREP fails to converge on some natural datasets. One example is the well-known KRK-illegal problem [Muggleton *et al.*, 1989;

[4] More precisely, we can conclude that C4.5rules outperforms IREP in this sense: if a problem is drawn at random from this test suite and its error rate is measured as described above, then with probability greater than 0.5, the measured error rate of C4.5rules will be lower than that of IREP.

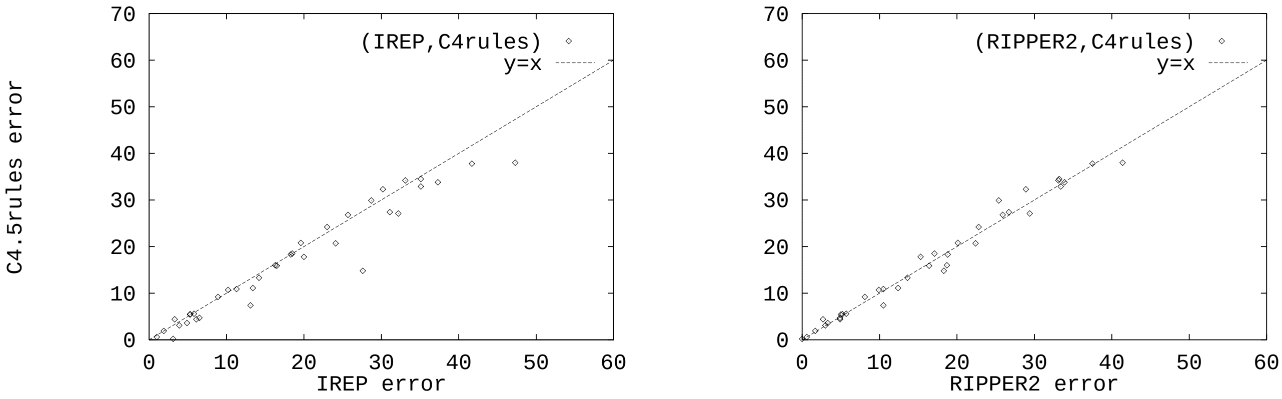

Figure 3: Comparison of generalization performances: C4.5rules *vs.* IREP and RIPPER2.

Quinlan, 1990]. We encoded a propositional version of this problem, and implemented a data generator.[5] Without noise, IREP reliably learns an approximate theory with an error rate of 0.6% from as few as 100 examples; however, IREP does not improve this error rate even if as many as 100,000 examples are given. In contrast C4.5rules reliably produces a perfect theory from only 5000 examples. Artificial examples can also be constructed which show non-convergence to a greater extent; for example, IREP obtains an error of 9.5% given anywhere between 100 and 100,000 noise-free examples of the concept $ab \vee ac \vee ade$. This is worrisome behavior for an algorithm whose main strength is that it efficiently handles very large numbers of examples.

4 IMPROVEMENTS TO IREP

Based on our experiments with IREP, we implemented three modifications to the algorithm: an alternative metric for assessing the value of rules in the pruning phase of IREP; a new heuristic for determining when to stop adding rules to a rule set; and a postpass that "optimizes" a rule set in an attempt to more closely approximate conventional (*i.e.*, non-incremental) reduced error pruning.

4.1 THE RULE-VALUE METRIC

The occasional failure of IREP to converge as the number of examples increases can be readily traced to the metric used to guide pruning (given above in Equation 1). The preferences encoded in this metric are sometimes highly unintuitive; for instance (assuming that P and N are fixed) the metric prefers a rule R_1 that covers $p_1 = 2000$ positive examples and $n_1 = 1000$ negative examples to a rule R_2 that covers $p_1 = 1000$ examples and $n_1 = 1$ negative example; note, however, that R_2 is highly predictive and R_1 is not. We thus replaced IREP's metric with

$$v^*(Rule, PrunePos, PruneNeg) \equiv \frac{p-n}{p+n}$$

which seems to have more intuitively satisfying behavior.

4.2 THE STOPPING CONDITION

Our implementation of IREP stops greedily adding rules to a rule set when the last rule constructed has an error exceeding 50% on the pruning data. This heuristic often stops too soon given moderate-sized samples; this is especially true when learning a concept containing many low-coverage rules. Our assessment of the problem is that for low-coverage rules, the estimate of error afforded by the pruning data has high variance; thus in learning a series of small rules, there is a good chance that one of the rules in the series will have its error rate incorrectly assessed at more than 50%, causing IREP to stop prematurely. Put another way, IREP seemed to be unduly sensitive to the "small disjunct problem" [Holte *et al.*, 1989].

Our solution to this problem is the following. After each rule is added, the total *description length* of the rule set and the examples is computed. The new version of IREP stops adding rules when this description length is more than d bits larger than the smallest de-

[5] Our propositional encoding is the one that would be constructed by LINUS [Džeroski and Lavrac, 1991], and we used a uniform distribution to generate KRK positions.

scription length obtained so far, or when there are no more positive examples. In the experiments of this paper we used $d = 64$. The rule set is then simplified by examining each rule in turn (starting with the last rule added) and deleting rules so as to reduce total description length.[6]

Together, the revised rule-value metric and stopping heuristic substantially improve IREP's generalization performance. Unlike the original IREP, the modified version of IREP (henceforth IREP$*$) converges *KRK-illegal* and the artificial concept $ab \vee ac \vee ade$. IREP$*$'s won-lost-tied record against IREP is 28-8-1; thus with high confidence ($p > 0.992$) one can state that IREP$*$ outperforms IREP on problems from this test suite. The error ratio to C4.5rules is also reduced from 1.13 (or 1.52, including *mushroom*) to 1.06 (or 1.04, including *mushroom*.) IREP$*$'s won-lost-tied record against C4.5rules is 16-21-0.

4.3 RULE OPTIMIZATION

The repeated grow-and-simplify approach used in IREP can produce results quite different from conventional (non-incremental) reduced error pruning. One way to possibly improve IREP$*$'s incremental approach is to postprocess the rules produced by IREP$*$ so as to more closely approximate the effect of conventional reduced error pruning. For instance, one could re-prune each rule so as to minimize the error of the complete rule set.

After some experimentation we developed the following method for "optimizing" a rule set $R_1, \ldots, R_k$. Each rule is considered in turn: first R_1, then R_2, etc, in the order in which they were learned. For each rule R_i, two alternative rules are constructed. The *replacement for* R_i is formed by growing and then pruning a rule R'_i, where pruning is guided so as to minimize error of the entire rule set $R_1, \ldots, R'_i, \ldots, R_k$ on the pruning data. The *revision of* R_i is formed analogously, except that the revision is grown by greedily adding conditions to R_i, rather than the empty rule. Finally a decision is made as to whether the final theory should include the revised rule, the replacement rule, or the original rule. This decision is made using the MDL heuristic.[7] Optimization is integrated with IREP$*$ as follows. First, IREP$*$ is used to obtain an initial rule set. This rule set is next optimized as described above. Finally rules are added to cover any remaining positive examples using IREP$*$. Below, we will call this algorithm RIPPER (for Repeated Incremental Pruning to Produce Error Reduction.).

Optimization can also be iterated by optimizing the rule set output by RIPPER and then adding additional rules using IREP$*$; we will call this algorithm RIPPER2, and in general use RIPPERk for the algorithm that repeatedly optimizes k times.

Table 2: Summary of generalization results

	won-loss-tied *vs* C4.5rules	error ratio to C4.5rules[a]
IREP[b]	9-28-0	2.08 1.71 1.93
IREP2	11-25-1	1.15 1.15 1.22
IREP[c]	11-23-3	1.51 1.13 1.20
IREP$*$	16-21-0	1.04 1.06 1.09
RIPPER	20-15-2	0.98 1.01 1.03
RIPPER2	21-15-1	0.97 0.99 1.01

[a]Format: all datasets; all datasets except *mushroom*; all datasesets except *mushroom* and weighting similar datasets together.
[b]Using Fürnkranz and Widmer's stopping criterion.
[c]As described in Section 2.3.

4.4 GENERALIZATION PERFORMANCE

RIPPER noticibly improves generalization performance over IREP$*$. Its won-lost-tied record against IREP$*$ is 28-7-2, a significant improvement ($p > 0.9986$). The error ratio to C4.5rules is also reduced: excluding *mushroom*, the error ratio is 1.06 for IREP$*$ and 1.01 for RIPPER, and including *mushroom*, the error ratio is 1.04 for IREP$*$ and 0.982 for RIPPER. RIPPER's won-lost-tied record against C4.5rules is 20-15-2.

One additional stage of optimization gives some fur-

[6]To briefly summarize our MDL encoding scheme: the method used for encoding a set of examples given a theory is the same as that used in the latest version of C4.5rules [Quinlan, 1995]. One part of this encoding scheme allows one to identify a subset of k elements of a known set of n elements using

$$S(n,k,p) \equiv k \log_2 \frac{1}{p} + (n-k) \log_2 \frac{1}{1-p}$$

bits, where p is known by the recipient of the message. Thus we allow $||k|| + S(n,k,k/n)$ bits to send a rule with k conditions, where n is the number of possible conditions that could appear in a rule and $||k||$ is the number of bits needed to send the integer k. As in C4.5rules [Quinlan, 1994, page 53] the estimated number of bits required to send the theory is multiplied by 0.5 to adjust for possible redundancy in the attributes.

[7]More precisely, a variant of R_i is evaluated by inserting it into the rule set and then deleting rules that increase the total description length of the rules and examples. The total description length of the examples and the simplified rule set is then used to compare variants of R_i.

ther benefit. RIPPER2 reduces the error ratio to C4.5rules to 0.995 excluding *mushroom*, or 0.968 including *mushroom*, and RIPPER2's won-lost-tied against C4.5rules is improved to 21-15-1. RIPPER2 is not statistically significantly better than C4.5rules; however, RIPPER2 is certainly quite competitive on the problems in this test suite. To make this concrete, let q be the probability that RIPPER2's measured error rate will be less than or equal to that of C4.5rules on a problem taken at random from the test suite. The won-lost-tied record of 21-15-2 means we can be 93% confident that q is at least 0.5, 95% confident that q is at least 0.488, and 99% confident that q is at least 0.431.

The right-hand graph in Figure 3 gives a more detailed comparison of the error rates of RIPPER2 and C4.5rules, and Table 2 summarizes some of the generalization results given in this section.

One problem with averaging error ratios is that when the actual error rates are very small, ratios tend to have extreme values. (This is the reason why we have reported all averages with and without the *mushroom* dataset: for this dataset the actual error rates range from 0.0% to 3.1% and the ratios range from 0.0 to 17.5.) The following remarks may help reassure readers of the stability of our comparison:

- If groups of similar datasets are weighted together,[8] then the average ratio of RIPPER2 to C4.5rules is 0.957. If *mushroom* is excluded, then the weighted average ratio is 1.005.
- If the two largest and the two smallest ratios are excluded, then the average ratio of RIPPER2 to C4.5rules is 0.986. (The ratio for *mushroom* is one of the four extreme values.)
- The average difference between RIPPER2's error rate and C4.5rules' error rate is -0.1%.
- The won-loss-tied record of RIPPER2 to the C4.5 decision tree learner (with pruning) is 23-12-2. The average ratio of RIPPER2 to C4.5 with pruning is 0.964 with *mushroom*, and 0.991 without.

4.5 EFFICIENCY OF RIPPERk

Importantly, none of the modifications we have described have a major effect on computational efficiency. Figure 2 also shows how RIPPER2 scales with the number of examples on three concepts: one artificial concept, and two of the larger and noisier natural datasets in our test suite. The fact that the lines for RIPPER2 and IREP are parallel shows that the modifications we have introduced affect only the constant factors, and not the asymptotic complexity of the algorithm. The constant factors for RIPPER2 are also still reasonably low: RIPPER2 requires only 61 CPU minutes to process 500,000 examples of the artificial concept of Figure 2. RIPPERk is also quite space efficient, as it requires no data structures larger than the dataset.

In previous work [Cohen, 1993] we sought formal explanations for the efficiency or inefficiencies of REP and other rule-pruning algorithms. While space does not permit such an analysis here, we would like to present some of the intuitions as to why RIPPERk is so much faster on large noisy datasets.

The basic strategy used by RIPPERk to find a ruleset that models the data is to first use IREP$*$ to find an initial model, and then to iteratively improve that model, using the "optimization" procedure described in 4.3. This process is efficient because building the initial model is efficient, because the initial model does not tend to be large relative to the target concept, and because the optimization steps only require time linear in the number of examples and the size of the initial model.

C4.5rules also constructs an initial model and then iteratively improves it. However, for C4.5rules, the initial model is a subset of rules extracted from a unpruned decision tree, and the improvement process greedily deletes or adds single rules in an effort to reduce description length. C4.5rules repeats this process for several different-sized subsets of the total pool of extracted rules and uses the best ruleset found as its hypothesis; the subsets it uses are the empty ruleset, the complete ruleset, and randomly-chosen subsets of 10%, 20%, ..., and 90% of the rules.

Unfortunately, for noisy datasets, the number of rules extracted from the unpruned decision tree grows as m, the number of examples. This means that each initial model (save the empty model) will also be of size proportional to m, and hence if m is sufficiently large, *all* of the initial models will be much larger than the target hypothesis. This means that to build a theory about the same size as the target concept always requires many (on the order of m) changes to the initial model, and at each step in the optimization, many (on the order of m) changes are possible. The improvement process is thus expensive; since it is a greedy search, it is also potentially quite likely to miss finding

[8] "Weighting similar datasets together" means that the ratios for the ten *AP* datasets, the five *bridges* datasets, the three *ticket* datasets and the two *network* datasets are each averaged together before being averaged with the ratios for the remaining seventeed datasets.

the best ruleset.[9]

In summary, both RIPPERk and C4.5rules start with an initial model and iteratively improve it using heuristic techniques. However, for large noisy datasets, RIPPERk generally seems to start with an initial model that is about the right size, while C4.5rules starts with an over-large initial model. This means that RIPPERk's search is more efficient. We conjecture also that RIPPERk's search is also more effective on large noisy datasets. (RIPPER2 generally seems to do better compared to C4.5rules on larger datasets; in particular for datasets with no more than 150 examples, the average ratio of RIPPER2 to C4.5rules is 1.051, and for datasets with more than 150 examples, the average ratio of RIPPER2 to C4.5rules is 0.944.)

5 CONCLUSIONS

Incremental reduced error pruning (IREP) is a recent rule learning algorithm that can efficiently handle large noisy datasets. In this paper we have presented some experiments on a large collection of benchmark problems with an extended implementation of IREP which allows continuous variables and multiple classes. We showed that IREP does not perform as well as the more mature (but also more expensive) rule learning algorithm C4.5rules.

We also proposed a series of improvements to IREP that make it extremely competitive with C4.5rules, without seriously affecting its efficiency. IREP$*$ incorporates a new metric to guide rule pruning and an MDL-based heuristic for determining how many rules should be learned. RIPPERk adds to this k iterations of an optimization step that more closely mimics the effect of non-incremental reduced error pruning.

IREP$*$ and RIPPERk were shown statistically to be clear improvements over IREP on problems from our test suite. RIPPER2 is also extremely competitive with C4.5rules; in fact on 22 of 37 problems in the test suite RIPPER2 achieves error rates lower than or equivalent to those of C4.5rules.

However, on noisy datasets, RIPPERk is much more efficient than C4.5rules. It scales nearly linearly with the number of examples in a dataset; in contrast C4.5rules scales as the cube of the number of examples. This asymptotic improvement translates to speedups of several orders of magnitude on problems of modest size (up to a few thousand examples), and the ability to effectively process datasets containing several hundreds of thousands of noisy examples.

[9] This situation should be contrasted to decision tree pruning, in which even a large tree can be pruned efficiently and, in certain senses, optimally; for instance, the pruned tree with the lowest error on a pruning set can be found in linear time.

Acknowledgements

Much of this research was conducted during a visit to the University of Sydney, which was funded by a grant to Ross Quinlan from the Australian Research Council. This research and this paper have benefitted substantially from numerous helpful discussions with Ross Quinlan, Nitin Indurkhya, and other members of University of Sydney machine learning community. The author is also grateful for valuable suggestions from Jason Catlett, for comments from Jason Catlett and Haym Hirsh on a draft of the paper, and for the suggestions made by two anonymous reviewers.

References

(Brunk and Pazzani, 1991) Clifford Brunk and Michael Pazzani. Noise-tolerant relational concept learning algorithms. In *Proceedings of the Eighth International Workshop on Machine Learning*, Ithaca, New York, 1991. Morgan Kaufmann.

(Cameron-Jones, 1994) Michael Cameron-Jones. The complexity of Cohen's Grow method. Unpublished manuscript, 1994.

(Catlett, 1991) Jason Catlett. Megainduction: a test flight. In *Proceedings of the Eighth International Workshop on Machine Learning*, Ithaca, New York, 1991. Morgan Kaufmann.

(Cohen, 1993) William W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993.

(Cohen, 1994) William W. Cohen. Grammatically biased learning: learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68:303–366, 1994.

(Džeroski and Lavrac, 1991) Sašo Džeroski and Nada Lavrac. Learning relations from noisy examples. In *Proceedings of the Eighth International Workshop on Machine Learning*, Ithaca, New York, 1991. Morgan Kaufmann.

(Fürnkranz and Widmer, 1994) Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In *Machine Learning: Proceedings of the*

Eleventh Annual Conference, New Brunswick, New Jersey, 1994. Morgan Kaufmann.

(Holte *et al.*, 1989) Robert Holte, Liane Acker, and Bruce Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. Morgan Kaufmann.

(Mendenhall *et al.*, 1981) William Mendenhall, Richard Scheaffer, and Dennis Wackerly, editors. *Mathematical Statistics with Applications*. Duxbury Press, second edition, 1981.

(Mingers, 1989) John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2), 1989.

(Muggleton *et al.*, 1989) Stephen Muggleton, Machael Bain, Jean Hayes-Michie, and Donald Michie. An experimental comparison of human and machine learning formalisms. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, New York, 1989. Morgan Kaufmann.

(Pagallo and Haussler, 1990) Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5(1), 1990.

(Pazzani and Kibler, 1992) Michael Pazzani and Dennis Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9(1), 1992.

(Quinlan and Cameron-Jones, 1993) J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In Pavel B. Brazdil, editor, *Machine Learning: ECML-93*, Vienna, Austria, 1993. Springer-Verlag. Lecture notes in Computer Science # 667.

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

(Quinlan, 1990) J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3), 1990.

(Quinlan, 1994) J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1994.

(Quinlan, 1995) J. Ross Quinlan. MDL and categorical theories (continued). In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

(Schaffer, 1992) Cullen Schaffer. Sparse data and the effect of overfitting avoidance in decision tree induction. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, California, 1992. MIT Press.

(Weiss and Indurkhya, 1991) Sholom Weiss and Nitin Indurkhya. Reduced complexity rule induction. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991. Morgan Kaufmann.