Fast Effective Rule Induction Overview

Prepared By:
Jeffrey M. Franczak
CS522 Data Mining
April, 12, 2000

jfranczak@yahoo.com

1 Introduction

“Fast Effective Rule Induction” is a study prepared by William Cohen of AT&T Bell Laboratories that discusses a rule-learning algorithm specifically designed to compete with C4.5rules by offering competitive accuracy performance, while at the same time, running more efficiently. Cohen’s primary goal was to develop an algorithm that would scale well on large noisy datasets. He improves upon the previous work in the areas of Reduced Error Pruning (REP) and Incremental Reduced Error Pruning (IREP) in order to deliver an improved and very accurate and efficient rule learning system.

Through an extensive amount of testing on a large diverse test suite, Cohen shows that his new rule leaning algorithm, Repeated Incremental Pruning to Produce Error Reduction (RIPPER2), is extremely competitive with the results of C4.5rules, while allowing scalability to much larger datasets.

History of REP-based Algorithms

Cohen published his RIPPERk research in 1995, but the family of REP-based algorithms dates back to at least 1987.

Below is a history of the most important contributions to REP-based rule learning and the researcher who published the algorithm.

1987	REP Decision Trees	Quinlan
1990	REP Decision Lists	Pagallo and Haussler
1994	IREP	Fürnkranz and Widmer
1995	RIPPERk	Cohen

Before discussing Cohen’s algorithms, it would be helpful to explain some of the previous work upon which he builds.

2 Reduced Error Pruning, REP

Since the underpinnings of Cohen’s algorithms are a descendant of REP, a brief overview of REP is in order.

REP is a technique used in conjunction with a rule learning system in order to improve the accuracy of a generated rule set.

Seminal implementations of REP were successfully applied to decision trees by [Quinlan 1987], and to decision lists by [Pagallo and Haussler, 1990]. A study of REP including experimental results can be found in [Brunk and Pazzani, 1991].

The REP algorithm is a separate-and-conquer rule learning algorithm, which initially overfits the training data with a set of rules, and then prunes the rule set until an acceptable level of accuracy is achieved. Generally, pruning improves error rates on unseen data, particularly when the data are noisy.

The goal of pruning is to remove aspects of rules that “over-generalize” to the training data. Spurious, incorrect, low-coverage cases in the training data, or incorrect portions of rules caused by algorithm anomalies such as bad splits, tend to create rule sets that do not adapt well to a new untrained case. Removing these types of conditions tends to improve the accuracy of the final rule set when it is applied to unseen (untrained) cases. There are, however, a statistically small number of cases where pruning negatively affects the overall accuracy of the rule set, but overall the benefits of pruning far outweigh these "outlier" scenarios. Furthermore, many of the specific types of problems that perform poorly with REP have been identified. Armed with this knowledge, the implementer can know when not to use REP. These problem areas provide areas for future research and improvement.

In REP, the training data are separated into two disjoint sets, a growing set and a pruning set. In the initial phase, rules are generated using a greedy heuristic methodology that utilizes the data in the growing set. This initial rule set generally overfits the growing set, i.e., the initial rule set is typically much larger and more complex than the final optimized rule set.

After generating the initial rule set, REP uses a post-pruning technique. The rule set is repeatedly simplified by applying pruning operators. At each step of simplification, the pruning operator that produces the best reduction of error on the pruning set is chosen. In other words, each rule is tested by applying it to the pruning set, and then modifying the rule using pruning operators to produce a new rule that best fits the pruning data. A typical pruning operator will delete a condition in a rule or an entire rule. Pruning stops when applying an additional pruning operator decreases the accuracy of a rule with respect to the pruning data.

Experiments by [Brunk and Pazzani, 1991; Cohen, 1993; Pagallo and Haussler, 1990; Fürnkranz and Widmer, 1994] showed that applying the REP algorithm generally does improve the rule-generation accuracy when noisy data are used. However, it was also shown that REP is a very computationally expensive algorithm when it is applied to large noisy data sets. Given sufficiently noisy data, REP required $O(n^4)$ time (where n is the number of training examples.) This type of performance would be prohibitive on large sets of training data, often encountered in real life situations. In 1995, Cohen [Cohen, 95] showed that C4.5rules scale as $O(n^3)$. Thus, the IREP algorithm, discussed next, was designed to address this particular issue.

3 IREP Detail

IREP is a learning algorithm developed by [Fürnkranz and Widmer, 1994] that used the basic concept of REP with a modified separate-and-conquer rule learning algorithm and a new pruning technique. Like REP, the training data were divided into a growing set and a pruning set. The Fürnkranz and Widmer's technique placed 2/3 of the training data into the growing set, and the remaining 1/3 into the pruning set.

The most significant change introduced in IREP was the integration of both pre-pruning and post-pruning. This integration completely eliminated the expensive initial phase of overfitting – IREP greedily generated the *final* rule set one rule at a time.

IREP built a rule set in a greedy fashion, a rule at a time. Immediately after a rule is generated, it was pruned. After pruning, the corresponding examples in the training set (growing and pruning sets) were deleted. Unlike REP, the remaining training data were re-partitioned after each rule was learned in order to help stabilize any problems caused by a "bad-split." The process was repeated until there were no positive examples left, or the last rule found had an unacceptably high error rate. More specifically, when the predictive accuracy of the pruned rule was less than that of the empty rule, that rule was not added to the rule set, and IREP returned the learned rule set.

In 1994, Fürnkranz and Widmer experimentally demonstrated IREP to be competitive with REP with respect to error rates, while running faster on 18 of 20 benchmark problems [Fürnkranz and Widmer, 1994]. The run time of IREP was shown to be $O(n \log^2 n)$ [Cohen, 95]. This was a significant improvement over REP's $O(n^4)$ performance.

An abridged intuitive explanation of IREP' s $O(n \log^2 n)$ performance follows:

1) Each rule will on average have log$n$ conditions, because each condition will cover about half of the random instances [Cohen, 93].
2) Each of these conditions has to be tested once against the n instances in the growing set. Thus, for each rule, the cost of adding one condition will be n.
3) With log$n$ conditions per rule, each tested n times, we have a total of n log$n$ for growing each rule.
4) During pruning, each of the log$n$ conditions must be tested against each of the n examples in the pruning set until the final rule is found, i.e., at most log$n$ times. Thus, the costs of pruning are $n \log^2 n$.
5) $O(n \log n) + O(n \log^2 n) = O(n \log^2 n)$
6) Since the size of the final rule set has been shown to be a constant (not proportional to n), the overall cost of IREP is = $O(n \log^2 n)$.

4 Cohen's Test Implementation of IREP

Cohen used IREP as his "point of departure." He retained most of Fürnkranz and Widmer' s original algorithm, with a few small alterations. He then benchmarked his implementation of IREP against C4.5rules as a baseline.

Additionally, Cohen added further value to the original IREP by adding the following features:

1) Support for Multiple Class attributes (IREP only supported binary attributes.)
2) Support for datasets that contain missing attributes. All tests involving attribute *A* are defined to fail on instances where the value of *A* is missing.

Adding support for multiple class attributes and missing attributes allowed Cohen to apply a much wider range of test suite problems, 37 experimental problems in all.

Cohen's GrowRule Implementation

GrowRule begins with an empty rule and considers adding any condition of the form $A_n = v$, $Ac \leq \theta$, or $A_c \geq \theta$, where A_n is a nominal attribute, 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 [Quinlan 1990; Quinlan and Cameron-Jones 1993].

Cohen' s implementation of GrowRule stops adding rules when rule learned has an error rate greater than 50% (IREP2' s stopping condition.) [1]

Cohen's Pruning Implementation

Cohen' s implementation of the Pruning Operator allowed deletions of any final sequence of conditions, whereas IREP only deletes a single final condition.

[1] The original IREP GrowRule stops adding rules when accuracy is less than the accuracy of the empty rule.

```
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: Cohen’ s Incremental Reduced Error Pruning

5 IREP Experimental Results

Some important facts are discovered when Cohen’ s IREP experimental results are examined. Figure 2 is a copy of Cohen’ s experimental plots representing the results of three of the experiments.

Discussion of Figure 2

Notice that the axes are “ CPU time” vs. “ Number of Training Examples” . This will directly show how well the algorithms scale to large amounts of training data, and how the algorithms stack up against each other.

It is important to note that a log-log scale is used. This allows the plotting of polynomial functions to appear as lines, with the slope indicating the degree of the function. Thus, it is easy to compare the relative complexity of different algorithms by taking the difference of their slopes.

Figure 2 Points of Interest:

1) The C4.5rules scales roughly as n^3 by comparing its plots to the kx^3 function.
2) The IREP algorithm scales almost linearly. Its plot is roughly parallel to the $m \log^2 m$ function shown.
3) Cohen provided the following results from the "Artificial Concept" dataset of 500,000 Examples:
    a) IREP takes 7 CPU Minutes
    b) C4.5rules takes 79 CPU Years (this is approximately five orders of magnitude slower!)
4) Additional examples with 100 -10,000 dataset sizes are also shown in other two graphs.

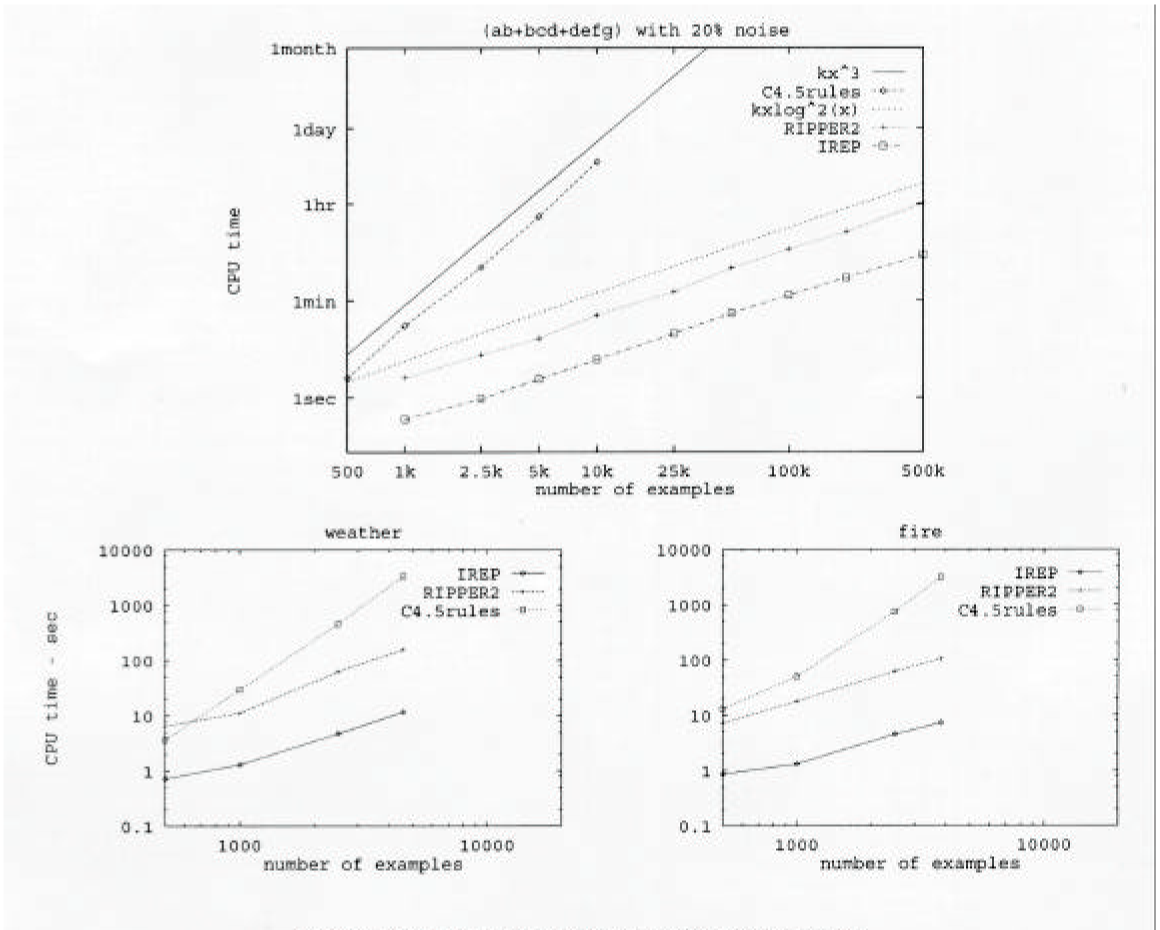

Figure 2: CPU times for C4.5rules, IREP, and RIPPER2

Figure 2: Selected Experimental CPU Times for C4.5rules, IREP, and RIPPER

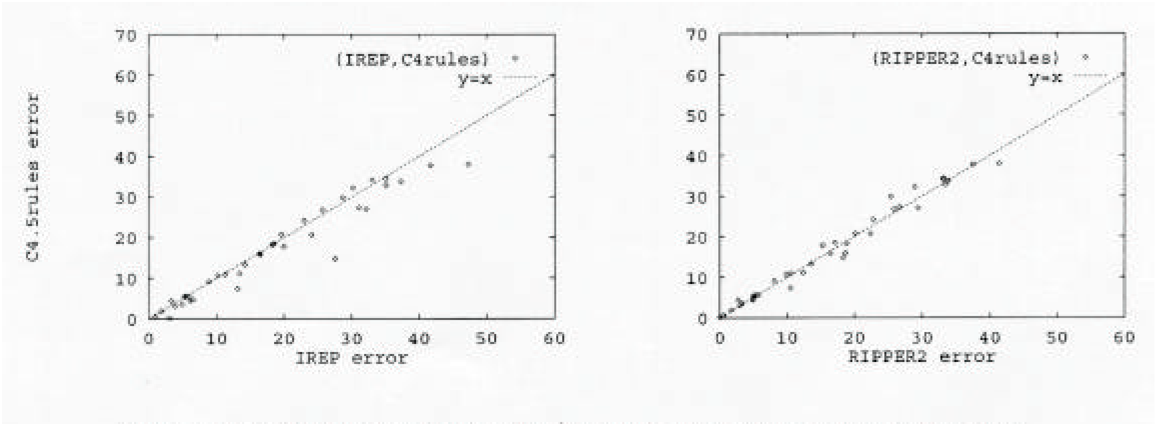

Figure 3: Comparison of generalization performances: C4.5rules *vs.* IREP and RIPPER2.

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

In reviewing Figure 3 above, which is a comparison of the generalization performance, or accuracy, of C4.5rules versus IREP, we gain some *disconcerting* information.

Discussion of Figure 3

Points below the line y =x indicate inferior IREP error performance, and points above the line y =x indicate superior IREP error performance.

IREP does worse than C4.5rules much more often than it does better, i.e., there are significantly more data points below than above the y = x line. The error rate of IREP is higher 23 times, lower 11 times, and the same 3 times.

Additional Test Results

One possible way of comparing the accuracy of two different algorithms is to compare their error rates. Cohen evaluated the error ratios of IREP compared to C4.5rules in order to gain some perspective on the “gap” between the accuracy of C4.5rules and IREP. The ratio is computed by averaging the errors from each test problem (for each algorithm) and then computing the ratio between the two averages. One particular dataset, the mushroom dataset, was determined to be an extreme statistical outlier, therefore it was excluded from the error comparison analysis.

IREP / C4.5Rules	1.13
IREP2 / C4.5Rules	1.15
FOIL with No Pruning / C4.5rules	1.17

Table 1: IREP vs. C4.5rules Error Comparison Ratios

Based on Table1, it can be seen that even the best variant of IREP is 13% less accurate than C4.5rules. Surprisingly, IREP didn' t do much better than Propositional FOIL (GrowRule) with no pruning.

Cohen' s experimentation determined that IREP failed to converge some datasets – " this is worrisome behavior for an algorithm whose main strength is that it efficiently handles very large numbers of examples." [Cohen, 95] He specifically points to two statistically significant test cases.

1) KRK-Illegal Problem (King vs. Rook and King, Chess Problem)
   a) IREP error rate using 100 –100,000 examples, 0.6%
   b) C4.5rules provided a perfect model from 5000 examples
2) Artificial Concept dataset, ab $\vee$ ac $\vee$ ade
   a) IREP Error rate using 100 – 100,000 *noise-free* examples, 9.5%

Based on the above results, the generalization performance of IREP conclusively offers *substantial* room for improvement. In Sections 6 and 7, we discuss Cohen' s improvements to IREP.

6 The IREP* Rule-Value Metric Improvement

In order to improve the accuracy of IREP, Cohen made two major changes to the algorithm. First, he proposes a new pruning metric, and second, a new stopping condition. He refers to the original IREP with these modifications as IREP*.

Cohen's IREP* Pruning Implementation Steps

1) Consider deleting any final sequence of conditions from the rule, and choose the deletion that maximizes the IREP* Rule-Value Metric (see Figure 4)
2) Repeat step 1 until no deletion improves the result of the IREP* Rule-Value Metric

$v^*(\text{Rule},\text{PrunePos},\text{PruneNeg}) \equiv [p - n] / [p + n]$ (IREP* Rule-Value Metric)

$v(\text{Rule},\text{PrunePos},\text{PruneNeg}) \equiv [p + (N - n)] / [P + N]$ (IREP Rule -Value Metric)

$v'(\text{Rule},\text{PrunePos},\text{PruneNeg}) \equiv p / [p + n]$ (IREP2 Rule-Value Metric)

Where,
P is the total number of examples in *PrunePos*
N is the total number of examples in *PruneNeg*
n is the number of examples in *PruneNeg* covered by *Rule*
p is the number of examples in *PrunePos* covered by *Rule*

Figure 4: Various Flavors of the IREP Rule-Value Metric

The occasional failure of IREP to converge (KRK Illegal and Artificial Concept Experiments) as the number of examples increases can be readily traced to the metric used to guide pruning.

Let' s look at an example where the IREP Rule-Value Metric produces some undesirable results:

Rule 1: $p = 2000$, $n = 1000$
Rule 2: $p = 1000$, $n = 1$

Reviewing the above example, we can see Rule 2 is very predictive, whereas Rule 1 is not. But, the IREP Rule-Value Metric prefers Rule 1 to Rule 2. However, the IREP* Rule-Value Metric clearly gives preference to Rule 1 in this example.[2]

A summary of results of the IREP* Rule-Value Metric as compared to the other IREP variants can be seen below in Table 2.

	won-loss-tied vs. C4.5 rules	error ratio to C4.5rules [a]
IREP[O]	9-28-0	1.71
IREP2	11-25-1	1.15
FOIL no pruning	17-20-0	1.17
IREP[C]	11-23-3	1.13
IREP*	16-21-0	1.06
RIPPER	20-15-2	1.01
RIPPER2	21-15-1	0.99

[a] Averaging all datasets except *mushroom*
[B] Propositional FOIL with no pruning as discussed in Section 5
[O] Using Fürnkranz and Widmer' s original stopping criterion
[C] Cohen' s implementation: Stops adding rules to a rule set when a rule learned has an error rate greater than 50%

Table 2: Summary of Generalization Performance

7 THE IREP* GROW RULE STOPPING CONDITION IMPROVEMENT

Cohen' s *initial* IREP implementation stopped greedily adding rules to a rule set when the last rule constructed has an error rate exceeding 50% on the pruning data. (Equation 7.2) This is the same stopping condition used in Fürnkranz and Widmer' s version of IREP2.

IREP: $(p / (p + n)) < (N / (P + N))$ *(Equation 7.1)*
IREP2: $(p / (p + n)) < 0.5$ *(Equation 7.2)*

Examining the error statistics in Table 2 we can see that Cohen' s IREP implementation performed substantially better than the original IREP, and somewhat better than IREP2. However, on average, IREP is still 13% less accurate than C4.5rules.

[2] The IREP2 Rule-Value Metric "correctly" prefers Rule 2, however [Fürnkranz and Widmer, 1994] state that in general, IREP2 Rule-Value Metric offers inferior performance when compared to the IREP version. Cohen, offers further proof with his own experiments shown in Table 1.

Cohen' s experimentation led him to the conclusion that the stopping condition, i.e,.Equation 7.2, was suspect in causing some problems to stop learning prematurely, thus giving rise to inaccuracies and higher than desired error rates. The 50% stopping condition was particularly sensitive to small and moderate-sized training sets, especially when learning a concept containing many low coverage rules. In other words, IREP seemed to be unduly sensitive to the "small disjunct problem" [Holte *et al.*, 1989]. Or, practically speaking, when learning low-coverage rules, the estimate heuristic when applied to the pruning set will have high variance. When learning a series of low-coverage rules, there is a reasonable chance (statistically), that at least one of the rules will have its error rate computed at greater than 50% (due to an uneven split between the growing and training sets.)

Cohen' s solution was to replace Equation 7.2 with a different measuring algorithm called the Minimum Description Length Principle or MDL. Cohen customized a version of MDL as implemented in C4.5rules in order to "measure" the complexity of the IREP rules being generated.

Cohen's MDL Algorithm

Although the specifics and the math that underlie MDL is fairly complex, the basic principle is quite easy to understand.

After each rule is added, the total description length, an integer value, of the rule set and the examples it covers in the pruning set (both positive and negative) are computed. When the description length is more than "d" bits larger than the smallest description length obtained so far, or when there are no more positive examples, IREP* stops generating new rules. Thus, the description length gives a *measure* of the complexity of a specific rule. By setting a value for "d" [3], we are in essence defining the maximum complexity of a rule in relationship to the least complex rule in the entire rule set.

Using MDL as a replacement stopping condition in conjunction with the new IREP* rule-value metric proved very successful. The error rate dropped to within 6% of C4.5rules and the won-lost-tied score improved to 16-21-0. Additionally, IREP* now successfully converges the KRK-illegal and artificial concept problems that were mentioned in Section 5.

8 RIPPER Post-Process Rule Optimization

The repeated grow-and-simplify approach used in IREP can produce significantly different results than conventional (non-incremental) REP. A possible way to further improve the IREP* incremental method is to post-process the rules produced by IREP*. This should more closely approximate the conventional method of REP since REP' s pruning algorithm is a post-process algorithm.

Cohen' s experimentation led to the following set of steps (method) of "optimizing" a rule set $R_1 \ldots R_{k.}$

1) Consider each rule in the order it was learned, one at a time, i.e., R_1, R_2, etc.
2) For each rule R_i, construct two alternatives, a "Replacement" R_i' , and a "Revision" R_i''
   a) R_i' is formed by growing (starting with an empty rule set), then pruning a rule, where the pruning is guided so as to minimize the error rate of the entire rule set: set $R_{1,\ldots,}$ R_i' $_{,}\ldots R_k$, on the pruning data.
   b) Ri' ' is constructed analogously to Ri' , except that the revision is grown by greedily adding conditions to Ri, rather than the empty rule.
3) Consider Ri, Ri' , and Ri' ' , and choose the best rule using the MDL metric discussed in Section 7.

[3] Experiments cited used a "d" value of 64.

Each 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 the variants of R_i.

Cohen called the new algorithm of applying the results of IREP* to the optimizer, to produce a new optimized rule set, Repeated Incremental Pruning to Produce Error Reduction, or RIPPER. Re-optimizing the resultant rule set of RIPPER is called RIPPER2, and the general case of re-optimizing "k" times is called RIPPERk.

RIPPERk Algorithm

1) Use IREP* to determine an initial rule set
2) Optimize rule set using rule optimization algorithm
3) Add rules to cover any remaining positive examples using IREP*
4) Take results of step 3, and re-optimize again by repeating steps 2 and 3
5) Stop after "k" optimizations

9 RIPPERk Performance Summary

RIPPER significantly improved generalization performance over IREP*. RIPPER's average error rate on the test suite was now within 1% of C4.5rules. And, RIPPER's won-lost-tie score was 20-15-2. RIPPER was now winning more often then losing when put head-to-head against C4.5rules on this test suite.

Adding one more stage of optimization, RIPPER2, produced even better results. RIPPER2 was slightly *more* accurate than C4.5rules, as measured against the test suite, and its won-lost-tie score improved slightly to 21-15-1.

Table 3 summarizes the most interesting RIPPERk's performance results as reported by Cohen.

Won/Lost/Tied, RIPPER vs. IREP*	28-7-2
Won/Lost/Tied, RIPPER vs. C4.5rules	20-15-2
Won/Lost/Tied, RIPPER2 vs. C4.5rules	**21-15-1**
Error Ratio, RIPPER vs. IREP*	1.06
Error Ratio, RIPPER vs. C4.5rules	1.01
Error Ratio, RIPPER2 vs. C4.5rules	**0.995**

Table 3: Summary of RIPPER Performance

10 RIPPERk Efficiency Analysis

Cohen has clearly shown RIPPER2 to be statistically compatible (with respect to accuracy) to the results produced by C4.5rules. In fact, on the test suite used, RIPPER2 slightly outperformed C4.5rules.

Also, none of the proposed modifications to IREP (IREP* and RIPPERk) have a major effect on computational efficiency. Looking at the graphed functions for RIPPER2 and IREP shown in *Figure 2*, we notice that they are parallel. This means that the modifications introduced by Cohen only affected the constant factors, and not the asymptotic complexity of the algorithm.

Even the increased constant factors of RIPPER2 are reasonably low; RIPPER2 was able to process 500K examples of the Artificial Concept test suite in 61 CPU minutes. The same problem would have taken C4.5rules about 79 CPU *years*. In addition, RIPPERk was also quite space efficient. It does not require any data structures larger than the test dataset.

The following intuitive explanation should clarify RIPPERk's ability to achieve such favorable results, while at the same time reducing run-time by several orders of magnitude on many problems of modest size.

RIPPERk's optimization process is efficient because building the initial model with IREP* is efficient. There are two basic reasons IREP* is efficient. First, the initial model does not tend to be large compared to the optimized model (mostly due to incremental pruning), and, second, the optimization steps only require linear time with respect to the number of examples and the size of the initial model.

In contrast, the optimized C4.5rules model is a subset of rules extracted from an unpruned decision tree. Generally, a lot of post-pruning is required, and the pruning process for C4.5rules tends to be quite inefficient. The brief outline of the C4.5rules pruning algorithm below, followed by a brief analysis, should clearly show why the C4.5rules process of optimization is inefficient.

C4.5rules Optimization Algorithm

1) The improvement process greedily deletes or adds *single* rules to reduce the description length of the rule set.
2) C4.5rules repeats this process for several different-sized subsets of the total pool of extracted rules, and then uses the best rule set it finds. The subsets used are:
    a) The empty rule set
    b) The complete rule set
    c) Randomly chosen subsets of 10%, 20%, …, 90% of the rules.

Analyzing the initial (unpruned) C4.5rules theory, we find that for noisy datasets, the number of rules from the unpruned decision tree grows as "n", the size of the training set. Because the unpruned result is so large, the following conditions arise:

A) Each initial model considered above, in Step 1, will be of size proportional to "n".
B) If "n" is sufficiently large, all the example models will be much larger than the best pruned model.
C) To build a final model requires many (order of "n") changes to the initial model, and at each step in the optimization, many (on the order of "n") changes are possible.
D) All these changes result in a very expensive optimization step.

The conclusion is that the IREP* pruning and RIPPERk optimization steps are much more efficient than the C4.5rules optimization (post-pruning) process. This is the chief reason RIPPERk significantly outperforms C4.5rules with respect to speed of rule generation.

11 Conclusions

Cohen has done an exceptional job of building upon the framework laid by Fürnkranz and Widmer' s IREP algorithm. He significantly improves its accuracy while maintaining the run-time performance that is the foundational impetus of IREP.

The RIPPERk algorithm has been demonstrated to be significantly faster than C4.5rules on a wide array of well-established test problems. At the same time, RIPPERk is essentially statistically equivalent to C4.5rules with respect to the accuracy of the rules produced.

Of particular importance, RIPPERk performs exceptionally well on large noisy datasets. This speedup allows RIPPERk to process several hundreds of thousands of noisy examples. This is precisely the type of data most often encountered in real-world situations.

12 ACKNOWLEDGEMENTS

Special thanks to William W. Cohen for his assistance in answering questions via e-mail.

I would also like to thank J. Ross Quinlan for his assistance in providing his previous work on FOIL and MDL.

Glossary of Terms

IREP Incremental Reduced Error Pruning

A learning algorithm developed by Fürnkranz and Widmer that tightly integrates reduced error pruning with a separate-and-conquer rule learning algorithm. Like REP, the training data are divided into a growing set and a pruning set. IREP was designed to approach the rule generation performance of C4.5rules, with a significantly shorter run-time.

IREP builds a rule set in a greedy fashion, a rule at a time. Immediately after a rule is generated it is pruned. After pruning, the corresponding examples in the training set (growing and pruning sets) are deleted. Unlike REP, the remaining training data are re-partitioned after each rule is learned in order to help stabilize any problems caused by a "bad-split." The process is repeated until there are no positive examples left, or the last rule found has an unacceptably high error rate [Fürnkranz and Widmer 1994].

IREP2

IREP that utilizes a pruning criterion that is "purity" based as opposed to the "accuracy" based criterion used in IREP. As cited in [Fürnkranz and Widmer 1994], in most cases, IREP outperformed IREP2 (with respect to error rate), however, certain learning problems were better suited to IREP2.

IREP*

A learning algorithm developed by Cohen based on the IREP algorithm. Cohen made several modifications to IREP, which improve its error performance and ability to generalize noisy data sets, while maintaining the efficiency gains of IREP [Cohen 1995].

MDL Maximum Description Length

An encoding scheme described in [Quinlan 1995] that is used to encode a theory and the set of examples given by the theory (rule set). The MDL equation produces a measurement metric that represents the number of bits required to encode the input theory. MDL is used by C4.5rules and IREP*. IREP* uses the MDL metric to represent the complexity of the input theory, which in turn guides the rule growing portion the IREP* algorithm. When the MDL metric exceeds the smallest description length, IREP* stops growing new rules.

Pre-Pruning

During rule generation some training examples are deliberately ignored so that the final rule set does not classify all training examples correctly. The driving force behind this technique is to ignore noisy training examples or those that will negatively affect the ability of the rule set to generalize.

Post-Pruning

After the rule set is generated that perfectly explains all training instances, the theory is generalized by altering and/or deleting some rules.

Glossary of Terms

REP Reduced Error Pruning

A rule learning algorithm reviewed by Brunk and Pazzani. This algorithm is a separate-and-conquer rule learning algorithm based on splitting the training data into two disjoint sets, a growing set and a pruning set. The initial phase forms an initial rule set that overfits the growing set using a greedy heuristic method (this initial rule set is typically much larger than the final/pruned rule set.) In the second phase, the overfitted rule set is then repeatedly simplified by applying pruning operators. At each step of simplification, the pruning operator that produces the best reduction of error on the pruning set is chosen [Brunk and Pazzani 1991].

RIPPER Repeated Incremental Pruning to Produce Error Reduction

An optimization postprocess developed by Cohen produces a rule set that more closely approximates the effect of conventional reduced error pruning. RIPPER is run against the resultant rule set generated by IREP* [Cohen 1995].

RIPPERk

Iterating the optimization step of the RIPPER algorithm "k" times in order to improve the generalization performance of the final rule set.

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.

(Cohen, 1993) William W. Cohen. Efficient Pruning Methods for Separate-and-Conquer Rule Leaning Systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 988-994, Chambery, France, 1993.

(Cohen, 1995) William W. Cohen. Fast Effective Rule Induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. 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, Liana Acker, and Bruce Porter. Concept Learning and the problem of small disjuncts. In *Machine Learning: Proceedings of the Eleventh Annual Conference*, New Brunswick, New Jersey, 1994. Morgan Kaufmann.

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

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