Lightweight Rule Induction

Sholom Weiss and Nitin Indurkhya

To appear in
Proceedings of the International Conference on Machine Learning (ICML) 2000

Lightweight Rule Induction

Sholom M. Weiss SHOLOM@US.IBM.COM
Nitin Indurkhya NITIN@DATA-MINER.COM
IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA

Abstract

A lightweight rule induction method is described that generates compact Disjunctive Normal Form (DNF) rules. Each class has an equal number of unweighted rules. A new example is classified by applying all rules and assigning the example to the class with the most satisfied rules. The induction method attempts to minimize the training error with no pruning. An overall design is specified by setting limits on the size and number of rules. During training, cases are adaptively weighted using a simple cumulative error method. The induction method is nearly linear in time relative to an increase in the number of induced rules or the number of cases. Experimental results on large benchmark data sets demonstrate that predictive performance can rival the best reported results in the literature.

1. Introduction

Decision trees and decision rules are well-known and related types of classifiers. The terminal nodes of a tree can be grouped into Disjunctive Normal Form (DNF) rules, only one of which is satisfied for a new case. Decision rules are also DNF rules, but allow rules to overlap, which potentially allows for more compact and interesting rule sets.

Decision tree induction methods are more efficient than those for decision rule induction–some methods for decision rule induction actually start with an induced decision tree. Procedures for pruning and optimization are relatively complex (Weiss & Indurkhya, 1993; Cohen, 1995). In terms of predictive performance, logic-based methods have difficulty with applications having complex solutions. With lesser support for many of the rules, the induced solutions are often bedeviled by high variance, where the training error is far from the test error.

Single decision trees are often dramatically outperformed by voting methods for multiple decision trees. Such methods produce exaggeratedly complex solutions, but they may be the best obtainable with any classifier. In Cohen and Singer (1999), boosting techniques (Schapire, 1999) are used by a system called SLIPPER to generate a weighted set of rules that are shown to generally outperform standard rule induction techniques. While these rules can maintain clarity of explanation, they do not match the predictive performance of the strongest learning methods, such as boosted trees. Of particular interest to our work is Friedman et al. (1998) where very small trees are boosted to high predictive performance by truncated tree induction (TTI). Small trees can be decomposed into a collection of interpretable rules. Some of the boosted collections of tiny trees, even tree stumps, have actually performed best on benchmark applications.

In this paper, a lightweight rule induction method is described that generates compact Disjunctive Normal Form (DNF) classification rules. A simple induction technique is applied to the full set of training cases attempting to minimize error with no pruning. An overall design is specified by setting limits on the size and number of rules. Design goals can be determined based on a tradeoff of simplicity of explanation for predictive performance. Although the algorithms are exceptionally simple for a rule induction system, experimental results on large benchmark data sets demonstrate that predictive performance can rival the best reported results in the literature.

2. Methods and Procedures

The classical approach to rule induction is a two-step process. The first step is to find a single covering solution for all training examples. The covering rule set is found directly by inducing conjunctive rules or indirectly by inducing a decision tree. The direct solution usually involved inducing one rule at a time, removing the cases covered by the rule, and then repeating the process. The second step is to prune the covering rule set or tree into smaller structures, and pick the best one, either by a statistical test or by applying the rule sets to independent test cases.

A pure DNF rule for classification is evaluated as satisfied or not. If satisfied, the rule implies a specific class. The conditions or components of a rule can be

tested by applying $\leq$ or $>$ operators to variables and coding categorical values separately as 1 for true and 0 for false.

We can measure the size of a DNF rule with two measurements: (a) the length of a conjunctive term and the number of terms (disjuncts). For example,

$$\{c_1\ c_2\ c_3\}\ \text{OR}\ \{c_1\ c_3\ c_4\} \Rightarrow \text{Class}$$

is a DNF rule for conditions c_i with maximum length of three and two terms (disjuncts). Complexity of rule sets can be controlled by providing an upper bound on these two measurements.

Table 1 describes the standard analysis of results for binary classification. For evaluation purposes, a rule is applied to each case. Classification error is measured as in equation 1. For case i, FP(i) is 1 for a false positive, FN(i) is 1 for a false negative, and 0 otherwise.

$$Error = FP+FN; FP = \sum_i FP(i); FN = \sum_i FN(i) \quad (1)$$

For almost all applications, more than one rule is needed to achieve good predictive performance. In our lightweight approach, a solution consists of a set of an equal number of unweighted rules for each class. A new example is classified by picking the class having the most votes, the class with the most satisfied rules. We are very democratic; each class has an equal number of rules and votes, and each rule is approximately the same size.

The principal remaining task is to describe a method for inducing rules from data. So far we have given a brief description of binary classification. Yet, this form of binary classification is at the heart of the rule induction algorithm. Let's continue to consider binary classification. The most trivial method for rule induction is to grow a conjunctive term of a rule by the greedy addition of a single condition that minimizes error. To ensure that a term is always added (when error is nonzero) we can define a slightly modified measure, err1 in equation 2. Error is computed over candidate conditions where TP is greater than zero. If no added condition adds a true positive, the cost of a false negative error is doubled and the minimum cost solution is found. The cost of a false positive remains at 1. The minimum err1 is readily computed during sequential search using the bound of the current best err1 value.

$$Err1 = FP + k \cdot FN\ \{where\ k = 1, 2, 4...and TP > 0\} \quad (2)$$

$$Frq(i) = 1 + e(i)^3 \quad (3)$$

$$FP = \sum_i FP(i) \cdot frq(i); FN = \sum_i FN(i) \cdot frq(i) \quad (4)$$

The lightweight method is adaptive, and follows the well-known principle embodied in boosting: Give greater representation to erroneously classified cases. The technique for weighting cases during training is greatly simplified from the usual boosting methods. Analogous to Breiman (1996a), no weights are used in the induced solution. Weighting of cases during sampling follows a simple method: Let $e(i)$ be the cumulative number of errors for case i for all rules. It is computed by applying all prior induced rules and summing the errors for a case. The weighting given to a case during induction is an integer value, representing a relative frequency of that case in the new sample. Equation 3 is the frequency that is used. It has good empirical support, having had the best reported results on an important text-mining benchmark (Weiss et al., 1999), and was first described in Weiss and Indurkhya (1998). Thus if 10 rules have been generated, 4 of them erroneous on case i, then case i is treated as if it appeared in the sample 65 times. Based on prior experience, alternative functions to Equation 3 may also perform well. Unlike the results of Bauer and Kohavi (1999) for the alternative of Breiman (1996a), Equation 3 performs well with or without random resampling, and the LRI algorithm uses no random resampling. The computation of FP and FN during training is modified slightly to follow Equation 4.

Err1 is computed by simple integer addition. In practice, we use only 33 different values of e(i), for i=0 to 32. Whenever, the number of cumulative errors exceeds 32, all cumulative errors are normalized by an integer division of 2.

The training algorithm for inducing a DNF rule R is given in Table 2. The algorithm is repeated sequentially for the desired number of rules. Rules are always induced for binary classification, class versus not-class. A m-class classification problem is handled by mapping it to m binary classification problems – one for each class. Each of the binary classification problems can be computed independently and in parallel. As we shall in Section 4, the equality of voting and rule size, makes the predictive performance of rules induced from multiple binary classification problems quite comparable.

2.1 Data Preparation

A very simple pre-processing step dramatically increases the efficiency of rule induction. Each DNF rule is induced from the complete sample and the expectation is that rule size is relative small. To find the best threshold for splitting a variable, substantial time can be expended in sorting the full data set repeatedly. Prior to any learning, we sort each variable only once

Table 1. Analysis of Error for Binary Classification

	Rule-true	Rule-false
Class-true	True positives (TP)	False negatives (FN)
Class-false	False positives (FP)	True negatives (TN)

Table 2. Lightweight Rule Induction Algorithm

1. Grow conjunctive term T until the maximum length (or until $FN = 0$) by greedily adding conditions that minimize err1.
2. Record T as the next disjunct for rule R. If less than the maximum number of disjuncts (and $FN > 0$), remove cases covered by T, and continue with step 1.
3. Evaluate the induced rule R on all training cases i and update $e(i)$, the cumulative number of errors for case i.

and produce a paired ordered list of the form $\{r_i, j\}$, where r_i is the i-th smallest real value and j is the j-th case. This representation doubles the size of the original data, but transforms the search for the best condition into a sequential loop over the sorted pairs. The expended time for this one-time sort of each attribute is a fixed cost that is almost inconsequential when compared to the overall induction time. Yet, the remaining processing is so simple and attuned to a modern sequential processing computer that the computational complexity becomes approximately linear in the number of rules or cases (See Section 4).

2.2 Estimating Future Performance

To select the solution with the best predictive performance, decisions must be made about the two key measures of rule size: (a) conjunctive term length and (b) the number of disjuncts. If the goal is data mining, then the best solution can be found by using an independent test set for estimating true error. For applications with lesser data, two adjunct measures are useful in model selection: (a) training error and (b) the percentage of close votes (margins) in correctly classified cases. Unlike voted combinations of large trees, the small size of the rules will often lead to training errors far larger than zero. As we shall see in Section 4, the trends for training errors, margins and solution complexity, have an important effect on generality.

2.3 Missing Values

A pure DNF rule induction system has strong capabilities for handling missing values. Disjunction can produce overlap and redundancy. If we apply a rule to a case, and a term is not satisfied because one of its conditions has a missing value, the rule may still be satisfied by one of the other disjuncts of the rule. These rules have no special conditions referring to missing values; they look no different than rules induced from data with no missing values. How is this accomplished? For the application of rules, a term is considered not satisfied when a missing value is encountered in a case. During training, the following slight modifications are made to the induction procedures:

- When looping to find the best attribute condition, skip cases with missing values.
- Normalize error to a base relative to the frequency of all cases.

$$Norm_k = \frac{\sum_{n,\,all} frq(n)}{\sum_{i,\,w/o\,missing\,vals} frq(i)} \tag{5}$$

$$FP_k = Norm_k \cdot \sum_i FP(i) \cdot frq(i) \tag{6}$$

$$FN_k = Norm_k \cdot \sum_i FN(i) \cdot frq(i) \tag{7}$$

Each feature may have a variable number of missing values. The normalization factor is computed as in Equation 5 for feature k. The normalization factor is the total number of cases, n, including missing values cases, divided by the frequency of cases without missing values. False positives and negatives are computed as in Equations 6 and 7, a straightforward normalization of Equation 4.

2.4 Feature Selection Speedup

In Friedman et al. (1998), a substantial speedup in training was demonstrated by ignoring cases having low weights. Decision trees were induced without these cases, but tested on all cases. If sufficient errors were encountered on the ignored cases, their weights were increased and they were re-admitted to the training sample.

For our lightweight rule induction method, the following alternative speedup is also effective: during the first k inductions of rules, record the features that are selected. After the first k rules are induced, ignore all features not selected by the first k rules. If the number of features selected is small relative to the total

Table 3. Comparison of High-Performance Rule Induction Methods

	SLIPPER	TTI	LRI
Disjunction	no	yes	yes
Pruning	yes	no	no
Random training	yes	no	no
Default rule	yes	no	no
direct goal	rule	tree	rule
Rule Size limit	no	yes	yes
Minimizing function	Z	gini/entropy	error
Multi-class	no	yes	yes
Weighted solutions	yes	yes	no
Weighted cases	yes	yes	yes
Adaptive function	boosted	boosted	cumulative error
Number of rules per class	unequal	unequal	equal
Voting	unequal	unequal	equal
Missing value rules	special	special	same

number of features, the speedup is high. The ignored features are not re-admitted to the pool.

3. Foundations of Lightweight Rule Induction

Table 3 summarizes the different characteristics of methods that improve on classical rule induction by adaptively inducing solutions. We compare SLIPPER and TTI with our new lightweight rule induction (LRI) method. Disjunction refers to individual DNF rules that are voted as one unit. As will be seen in Section 4, some applications require the more complex basis function implied by disjuncts within an individual rule. Some systems have default rules for the largest class, others generate rules for each class. Most adaptive methods weight each rule or each disjunctive rule set. Although TTI and LRI differ significantly in the use of direct rule induction versus small tree induction, LRI has more similarities to TTI than SLIPPER. Yet, the LRI scheme is highly simplified. It produces unweighted rules; the class with the most votes is selected. All classes have the same number of rules. The minimizing function is simply the error and the adaptive function for weighting the case during training uses the simple cumulative error of all previous rules.

Experienced users of conventional rule induction methods may be very surprised by LRI's egalitarian representation of decision rules. These rules appear the same as rules induced by classical methods, yet many rules are approximately the same size, and each class has an equal number of rules. Surely something is amiss when we find applications where a class is readily discriminated by a single short rule, and another class requires many long rules.

The classical view of a rule or tree induction is to find a strong set of rules or concepts that explain the cases. A good solution consists of a many strong rules each of which covers a subset of cases. A decision tree for example will have only one rule invoked for a single case, and to be successful, it must be a strong rule that effectively captures the concept that it represents,

But the rules of LRI are not induced in the conventional manner. To vote rules, each rule must be considered a complete "solution" with training and testing on all cases in the sample. Voting methods can combine many weak solutions and produce a strong overall solution. For each DNF rule, the LRI algorithm in Table 2 clearly tries to induce a complete solution taken over "all cases" in the (weighted) sample. It does not vote the number of disjuncts fired in a rule; it gives one vote to each DNF rule, and that rule is a complete, often weak solution for all cases. Because the solutions are induced by binary classification, it is natural to find an equal number of complete solutions for each class, and then select the class with the most votes, Although we use unweighted votes, the identical solution may be replicated many times, and its relative importance increased by tabulating the votes.

To understand why most LRI rules are approximately the same size, i.e. bounded by a maximum number of disjuncts and maximum length, we look to statistical learning theory. In Friedman et al. (1998), we see boosting explained as a special form of additive logistic regression, a voting system that can use a basis function of fixed complexity. In classical statistical fashion, they show that the basis function need not be complex like a full tree. Instead, they demonstrate that truncated trees, i.e. trees having a maximum depth, have the potential for very strong solutions when the right-size basis function is used. Their trees are grown only to a fixed depth, and the size of the implied rules are fixed, exactly as we do in LRI. From a classical statis-

tical viewpoint, choosing a basis function of fixed complexity is natural for many statistical learning methods such as regression with splines (Friedman, 1991) or neural nets with a fixed number of hidden units (Weiss & Kapouleas, 1989), or cost complexity pruning with resampling for decision trees (Breiman et al., 1984). Individual LRI rules of a fixed, short size may be weaker and make more errors than those found by classical methods, but in the context of voting they may be considered basis functions whose fixed complexity allows for estimation of just the right fit for the best solution.

Ultimately the validity of this approach can be tested on data. In the next section, we present our experimental results.

4. Results

To test the efficacy of lightweight rule induction, data sets from the UCI repository (Blake et al., 1999) were processed. Table 4 summarizes the characteristics of these data. The number of features describes numerical features and categorical variables decomposed into binary features. Our principle objective is data mining, so we selected data sets having relatively large numbers of training cases and designated test sets. We also included several non-proprietary data sets described in Cohen and Singer (1999). Both real-world and synthetic data sets (wave, led, noise) were used. It is well-known that decision trees do poorly on many of the data sets that we used, for example letter, digit and wave (as will be shown in Table 5). Two noisy data sets are also included (a) noise - a set of random numbers with a .3 prior for the smaller class and (b) digit25 - the digit data set with 25% of the feature values randomly set to missing.

Table 4. Data Characteristics

Name	Train	Test	Features	Classes
adult	30162	15060	105	2
blackj	5000	10000	6	2
coding	5000	15000	60	2
digit	7291	2007	256	10
digit25	7291	2007	256	10
dna	2000	1186	180	3
isolet	6238	1559	617	26
led	5000	5000	24	10
letter	16000	4000	16	26
move	1483	1546	76	2
noise	5000	5000	20	2
satellite	4435	2000	36	6
splice	2175	1000	240	2
wave	5000	5000	40	3

LRI has 4 design parameters that affect results: (a) the number of rules per class (b) the number of rules after which feature selection is frozen (c) the maximum length of a rule and (d) the maximum number of disjunctions. For all of our experiments, we set the length of rules to 5 conditions and froze all feature selection after 50 rules for each class. We varied the number of disjuncts in each rule from 1, 2, 4, 8, 16, where 1 is a rule with a single conjunctive term.

We also varied the number of rules to measure the gain in predictive performance as the number of rules increases versus the performance of simpler, but fewer rules. Table 5 summarizes the results for a maximum number of terms from 1 to 16. Also included in the table are the published results of SLIPPER, and the results of selecting the minimum test-error tree from a collection of pruned trees trained by a variation of CART applied solely to the training data. Because the test error was used for finding the minimum error tree, the tree results are somewhat optimistic. Still, for data mining applications, this procedure might be quite reasonable (Breiman et al., 1984). The standard error is computed for the error of the minimum tree. A asterisk next to an LRI entry indicates that it is the minimum training error solution. Although complexity can be increased, LRI does not always decrease its training error. Occasionally, the training error can actually degenerate markedly, e.g. the blackjack data. A dagger indicates that the entry is a much simpler solution and is within 1% of the minimum training error.

Table 5 compares the results for just 10 LRI rules with other DNF methods. For greater numbers of rules, it is possible to far exceed those levels of predictive performance. Table 6 summarizes the results for varying the number of rules for each class. The error listed is for the solution with the minimum training error. Also listed is the global minimum test error for any of the trained solutions. The approximate minimum error found in the literature is also given with its standard error. For noise, led, and wave, the Bayes error is known. For satellite, this result is cited in Dietterich (in press). For blackj, coding, move and splice, the results cited in Cohen and Singer (1999) are used since they were based on similar train-test splits as ours. For adult, the result in Cohen and Singer (1999) is used even though it is based on a smaller training set (5000 cases only) than ours. For letter, the result is cited in Friedman et al. (1998). For digit and dna, the results are taken from Breiman (1996b). For isolet, the results are cited in Dietterich and Bakri (1991).

To obtain approximate timings, LRI was trained on the digit data set with two and four terms. The number of rules was increased from 25 to 50 to 100. To observe the effect of doubling the number of rules, a baseline time for 25 rules was compared to 50 rules, and a baseline time for 50 rules was compared to 100 rules. Figure 1 summarizes the results. The time al-

Table 5. Comparative Error for Rule Sets with Ten Rules

Name	Number of disjuncts per rule 1	2	4	8	16	SLIPPER	min-tree	SE
adult	.158	.153	.150	.152	.143*	.147	.145	.002
blackj	.308	.283*	.330	.281	.612	.279	.278	.004
coding	.331	.332	.294	.292	.295*	.302	.337	.004
digit	.110	.099	.091	.089†	.088*	-	.154	.008
digit25	.158	.142	.130	.129†	.114*	-	-	-
dna	.161	.076	.063	.051†	.064*	-	.075	.008
isolet	.078	.072†	.081	.083	.113*	-	.173	.010
led	.275	.266	.284	.282†	.291*	-	.264	.006
letter	.215	.155	.114	.089	.071*	-	.134	.005
move	.282	.278	.255	.241	.202*	.239	.255	.011
noise	.324	.317	.336	.332	.339*	-	.298	.006
satellite	.141	.128	.117	.113	.116*	-	.146	.008
splice	.227	.085	.050	.044†	.045*	.059	.043	.006
wave	.167	.158	.160	.155†	.162*	-	.231	.006

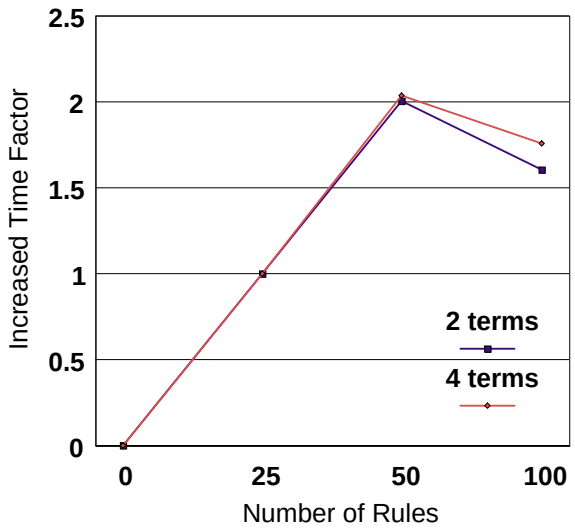

Figure 1. Timing Ratio on Doubling the Number of Rules

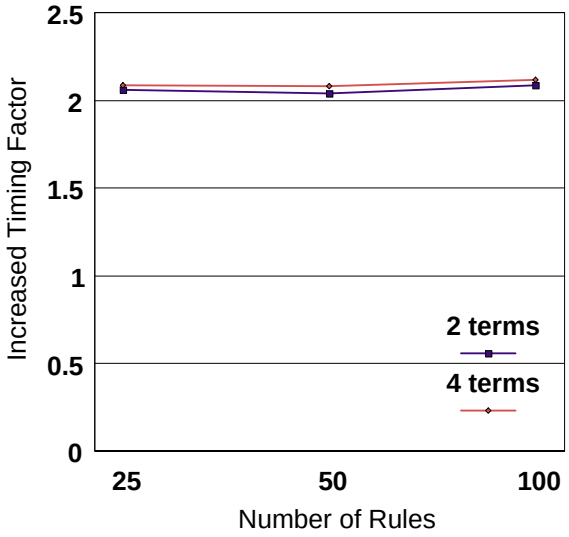

Figure 2. Timing Ratio on Doubling the Number of Cases

most exactly doubles for an increase from 25 to 50 rules. For 100 rules, the time factor is less than 2 because feature selection is frozen at 50 rules.

A random sample of 50% was taken from the digit training data, and timings were made for training both the full sample and the half-sample. Figure 2 summarizes the results. In all variations, a doubling of the number of cases takes approximately twice the time for the same number of rules.

When working with very large databases, the following effects can be noted: (a) Memory is not a major limiter. The data are processed in linear order, and all data could reside completely on disk with minimal use of real memory. (b) Computational time for increased number of rules or cases is nearly linear as demonstrated by Figures 2 and 3. (c) One need not repeat experiments for different number of rules. For a fixed size rule, the first k rules are the same no matter how many additional rules are induced. in general results improve or stabilize with more rules and smaller numbers of rules can be extracted from a single run of a larger number of rules. Therefore the number of rules is not a factor requiring repeated experiments. (d) Separate training for different rules sizes increases computational costs. It's balanced by the

Table 6. Comparative Error for Different Number of Rules

Name	Number of rules in LRI-solution 25	50	100	250	500	LRI-min	Best report	SE
adult	.135	.133	.133	.131	.132	.131	.147	.003
blackj	.273	.274	.275	.276	.276	.273	.278	.004
coding	.276	.256	.251	.249	.246	.246	.302	.004
digit	.073	.067	.062	.060	.059	.059	.062	.005
digit25	.096	.088	.087	.082	.082	.082	-	-
dna	.051	.046	.042	.046	.045	.042	.042	.006
isolet	.056	.048	.049	.050	.052	.048	.033	.005
led	.284	.265	.263	.265	.266	.263	.260	.006
letter	.048	.043	.039	.039	.040	.039	.029	.003
move	.228	.205	.195	.200	.195	.195	.239	.011
noise	.318	.311	.309	.305	.305	.299	.300	.006
satellite	.094	.093	.093	.091	.092	.092	.085	.006
splice	.042	.040	.038	.039	.039	.035	.043	.006
wave	.151	.147	.149	.150	.148	.142	.140	.005

fast training. (e) By far the biggest expense is multi-class application, which can be solved in parallel as independent problems. For sequential processing, the computational costs are proportional to the number of classes.

5. Discussion

Lightweight Rule Induction has a very simple representation: pure DNF rules for each class. It is egalitarian, each class has the same number of rules of approximately the same-size rules. Scoring is trivial to understand: the class with the most satisfied rules wins.

The method is about as simple as any rule induction method can be. The algorithm is rudimentary, and our C code implementation is less than 300 lines. It produces designer rules, where the size of the rules are specified by the application designer.

The central question in Section 4 is: How well does LRI do on practical applications? For best predictive performance, a number of parameters must be selected prior to running. We have concentrated on data mining applications where it can be expected that sufficient tests are available for easy estimation. Thus, we have included results that describe the minimum test error. With big data, its easy to obtain more than one test sample, and for estimating a single variable, a large single test set is adequate in practice (Breiman et al., 1984). For purposes of experimentation, we fixed almost all parameters, except for maximum number of disjuncts and the number of rules. The number of disjuncts is clearly on the critical path to higher performance. As already shown for boosting and all forms of adaptive resampling, most of the gains in performance are achieved with the initial smaller set of classifiers.

How good is predictive performance? For even small numbers of rules, performance is generally superior to its rule induction competitors, some of which only operate on binary classification. When the number of rules is increased, LRI can compete with the best classifiers.

Once the features are pre-sorted, timings suggest that training is nearly linear in most directions. Intuitively this makes sense. The program merely loops on linear lists for each condition that is added to a rule. Unless the complexity of the solutions changes drastically, adding cases or specifying more rules, just lengthens the sorted list or the number of rules. Speedup can also improve on linear performance when only a subset of features are detected as useful during initial training.

Of special interest is the natural capability of pure DNF rules to train and process examples with missing values. The inherent redundancy of disjunction was demonstrated in the digit25 example, where 25% of the feature values were destroyed, yet predictive performance was almost maintained. And most importantly, the solution's rules had no special mention of missing values. The rules look like rules generated without missing values.

The binary classification model requires a separate induction of rules for each class. Although theoretically compensated by parallel and independent processing, separate induction is a drawback for multi-class induction of mutually exclusive classes. One might consider the alternative of inducing small truncated trees using our adaptive scheme. We did try this approach; it was not successful. Ignoring the issues of equal-

ity of rule numbers and redundancy, we found that for unweighted rules, very large trees were needed to match the performance of LRI. Otherwise for smaller trees, coverage of classes was insufficient and could not compete with the faster converging weighted boosting methods.

We examined a good portion of the large data sets that are freely available for analysis. We added some noisy data sets, including one with 100% noise. Although there is still some tendency to overfit, the effect seems less than found for boosting (Dietterich, in press), and the results suggest that overfitting is avoidable by testing with varying complexity.

Increasing the margins of votes for correct and incorrect answers has been described as the key mechanism of boosting (Schapire et al., 1998). LRI directly improves margins of voting in binary classification. In addition to recording training error, we also recorded the percentage of correct cases with a margin of less than 10% for the second highest vote getter. These results were not used directly in our tables of results, but the margin can be very helpful in estimating generalization. For example in the noise application, the training error decreases for increase complexity, yet the margins of correct cases show a marked decrease.

Overall, results for lightweight rule induction are very promising, and as with any new method, additional real-world experience is needed to determine its weaknesses and its ultimate potential.

References

Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, *36*, 105–139.

Blake, C., Keogh, E., & Merz, C. (1999). *Uci repository of machine learning databases* (Technical Report). University of California Irvine. www.ics.uci.edu/∼mlearn/MLRepository.html.

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, *24*, 123–140.

Breiman, L. (1996b). *Bias, variance, amd arcing classifiers* (Technical Report 460). University of California, Berkeley.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees.* Monterrey, CA.: Wadsworth.

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

Cohen, W., & Singer, Y. (1999). A simple, fast, and effective rule learner. *Proceedings of Annual Conference of American Association for Artificial Intelligence* (pp. 335–342).

Dietterich, T. (in press). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning.*

Dietterich, T., & Bakri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. *Proceedings of American Association on Artificial Intelligence* (pp. 572–577).

Friedman, J. (1991). Multivariate adaptive regression splines. *Annals of Statistics*, *19*, 1–141.

Friedman, J., Hastie, T., & Tibshirani, R. (1998). *Additive logistic regression: A statistical view of boosting* (Technical Report). Stanford University Statistics Department. www.stat-stanford.edu/∼tibs.

Schapire, R. (1999). A brief introduction to boosting. *Proceedings of International Joint Conference on Artificial Intelligence* (pp. 1401–1405).

Schapire, R., Freund, Y., Bartlett, P., & Lee, W. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, *26*, 1651–1686.

Weiss, S., Apté, C., Damerau, F., & et al. (1999). Maximizing text-mining performance. *IEEE Intelligent Systems*, *14*, 63–69.

Weiss, S., & Indurkhya, N. (1993). Optimized rule induction. *IEEE EXPERT*, *8*, 61–69.

Weiss, S., & Indurkhya, N. (1998). *Predictive data mining: A practical guide.* Morgan Kaufmann. DMSK Software: www.data-miner.com.

Weiss, S., & Kapouleas, I. (1989). An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. *Proceedings of International Joint Conference on Artificial Intelligence* (pp. 781–787). Detroit, Michigan.