Optimized Rule Induction

Sholom M. Weiss and Nitin Indurkhya, Rutgers University

SWAP-1 IS A STATE-OF-THE-ART SYSTEM FOR LEARNING DECISION RULES FROM DATA. FOR MANY APPLICATIONS, SUCH SYSTEMS CAN AUTOMATICALLY CONSTRUCT RELATIVELY COMPACT RULE SETS WITH HIGHLY PREDICTIVE PERFORMANCE.

WITH WORKSTATIONS APproaching the processing speeds of supercomputers, and with databases exploding in size, interest has heightened in computer-based learning from data. Traditional knowledge-based systems have faced significant problems in acquiring and maintaining knowledge. One model of learning, the rule-based approach, has been extensively used in these systems. Techniques for automatically learning decision rules could have a profound effect on future intelligent systems.

Although many different models of induction, such as decision trees, neural nets, and linear discriminants, have been used for classification, they share a common goal: predictive accuracy. A central issue in the design of most classifiers is the tradeoff of goodness of fit versus model complexity. While we can usually improve a complex classifier's coverage of training samples, its accuracy of prediction for new cases might still be inferior to that of a simpler classifier. For example, a fully expanded decision tree might cover training samples completely, but a smaller tree with a larger error on the training cases might produce more accurate predictions for new cases.

Classifier complexity and prediction accuracy are highly related. Learning from sample data can be described as estimating a model's parameters. To find the appropriate complexity fit for a model, we determine the number of parameters that can be accurately estimated from the samples. Given two classifiers that cover the sample data equally well, the simpler one is usually preferred because fewer parameters are estimated and therefore, predictions are probably more accurate.[1-3] Thus, there are strong theoretical reasons for developing learning methods that cover samples in the most efficient and compact manner.

In practice, designers of learning systems have implicitly recognized these principles, and many of their techniques for simplifying models can be characterized as finding relatively compact solutions with an appropriate complexity fit.[4] Examples from decision trees are quite numerous, including heuristic tree-splitting functions,[5] tree pruning,[5,6] and the one-standard-error heuristic for selecting among pruned trees.[5] For parametric statistical linear discriminants, heuristic methods for selecting variables have reduced the number of discriminant features.[7] Rule induction systems have simplified solutions by pruning the implicit rules in decision trees.[6] For single hidden-layer backpropagation neural nets, the number of hidden units can be used as a measure of complexity fit, and the apparent and true error rates follow the classical statistical pattern.[8]

While short expressions in disjunctive normal form (defined below) sometimes offer superior solutions and therefore support the reduced-complexity approach,[9] they apply only to problems with few attributes

0885/9000/93/1200-0061 $3.00 © 1993 IEEE

An earlier version of this article appears under the title "Reduced Complexity Rule Induction" on pages 678-684 in the *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, published by Morgan Kaufmann, San Mateo, Calif., 1991.

```
input: S a set of training cases
initialize R1 := empty set, k := 1, and C1 := S

repeat
   create a rule B with a randomly chosen attribute as its left-hand side
   while (B is not 100-percent predictive) do
      make the single best swap for any component of B,
         including deleting the component, using cases in Ck
      if no swap is found, add the single best component to B
   endwhile
   Pk := rule B that is now 100-percent predictive
   Ek := cases in C that satisfy the single-best-rule Pk
   Rk+1 := Rk ∪ {Pk}
   Ck+1 := Ck – {Ek}
   k := k+1
until (Ck is empty)
find rule r in Rk that can be deleted without affecting performance on cases in S
while (r can be found)
Rk+1 := Rk – {r}
k := k+1
endwhile
output Rk and halt
```

Figure 1. The Swap-1 procedure.

Table 2. Example of swapping rule components.

Step	Predictive value (%)	Rule
1	31	p3
2	36	p6
3	48	p6 & p1
4	49	p4 & p1
5	69	p4 & p1 & p2
6	80	p4 & p1 & p2 & p5
7	100	p3 & p1 & p2 & p5

and classes. Our method generates reduced-complexity solutions by inducing compact solutions in larger dimensions, where many rules might be needed to make accurate predictions.

The rule-based classification model

The chief advantage of disjunctive normal form models is their explanatory capability. They help users answer the question "Why was the decision made?" in a far more comprehensible manner than neural networks or discriminant functions, so they have been a popular approach to building decision support systems and other knowledge-based systems.

Both decision trees and rules can be described as DNF models. They take as input a set of sample cases S, each case comprising observed features and a classification. The problem is to find the best rule set, the one that minimizes the error rate on new cases. Solutions are posed in disjunctive normal form, where each class is specified by a disjunctive set of productions, or rules, which are composed of conjunctive propositions that are either true or false. For example, the rule set

```
CA > 0.5 And CP > 3.5  →  Class = 2
THAL > 6.5             →  Class = 2
[True]                 →  Class = 1
```

is a solution induced from heart disease data (discussed in detail later). The left side of each rule is a conjunction of propositions p_i, each of which evaluates the truth of a binary-valued feature or checks the threshold of a numerical feature's current value. If the left side of the rule is satisfied, the case is assigned to a class according to the right side of the rule. We do not require mutual exclusivity of rules, so we must resolve conflicts when rules for two or more classes are satisfied simultaneously: We assign priorities to classes, with the

Induced rules

The following are the best answers we found. "LD1" is the linear discriminant for Class 1.

Heart disease classification:

```
LD1: (THAL*1.73) + (CA*2.027) + (EXANG*3.665) + (TLCH*0.426) + (CP*5.907) + (OLDPK*2.149) + (SEX*2.394) + (–50.848)
LD2: (THAL*1.206) + (CA*0.896) + (EXANG*2.451) + (TLCH*0.448) + (CP*5.23) + (OLDPK*1.674) + (SEX*1.367) + (–46.929)
LD1 → Class 1
[True] → Class 2
```

DNA pattern analysis:

```
LD1: (F58*9.304) + (F68*6.426) + (F62*6.181) + (F156*–3.973) + (F60*7.256) + (–9.963)
LD2: (F58*2.381) + (F68*1.508) + (F62*1.354) + (F156*1.459) + (F60*2.488) + (–1.669)
LD1 → Class 1
[True] → Class 2
```

Rheumatic disease diagnosis:

```
GOTPA → PM
RNP → MCTD
SCLDY → PSS
PSSBX → PSS
~MALAR & ~FEV → RA
[True] → SLE
```

Nettalk: A listing of the 253 rules is available from the authors.

last class considered the default. In our model, all the rules for each class are grouped together. The first rule that fires will determine the class selection.

Decision trees are a restricted version of DNF rules, since they are collections of ORed decision rules. In a decision tree, each path to a terminal node is represented as a rule consisting of a conjunction of tests on the path's internal nodes; the rule's conclusion is the label of the terminal node. One such rule is obtained for each terminal node. By ORing the rules together, we get a DNF rule set that is equivalent to the decision tree. However, since these rules are mutually exclusive, exactly one rule will apply in each case. This restriction can result in solutions that are not as compact as they might be. Thus, as a tree grows in size, its explanatory capability diminishes.

The representation of decision rules resembles that of decision trees, but with some potential advantages. It is a theoretically stronger model and has potentially better explanatory capabilities. Unlike trees, decision rules need not be mutually exclusive; they directly reflect the model's DNF structure. They are also simpler and more compact.

While tree induction remains the most widely applied rule-based learning system, other learning techniques for nonmutually exclusive DNF rule induction have been developed, including C4[10] and the CN2[11] and Greedy3[12] variants of the AQ family of rule induction systems.[13] These methods can be described in terms of their covering schemes and rule refinement techniques. Although they seem quite different, only a few key variations emerge. The covering rule set is induced either by using a decision tree or by finding a single best rule, removing the cases covered by that rule from the training sample, inducing the next rule, and repeating this process until no cases remain. Current tree-covering and single-best-rule covering methods look ahead one attribute and try to specialize the tree or rule, often by using some heuristic mathematical function.[5] For the tree-covering solutions, these heuristics tend to work well on many problems, and the combinatorics of finding an optimal solution make alternative search procedures impractical. Like the tree-covering methods, single-best-rule methods expand only one rule at a time and add propositions one by one until the rule has 100-percent predictive value, that is, until it makes no errors on the training cases. (Some variations such as CN2 sometimes stop earlier, based on a statistical significance test.) Although any single rule is relatively short, these single-best-rule procedures never look back or undo previous decisions, only ahead for a single new test.

However, even though we have obtained a covering rule set, it often "overfits" the training data, performing poorly on new cases. A second refinement step is needed to adjust the rule set to the right complexity fit, either by pruning or by applying some statistical test. Our method preserves this modular two-stage process of rule induction, but we use a new procedure that obtains compact covering rule sets, and we provide a unified approach to finding the appropriate complexity fit among several competing rule sets.

AS OPPOSED TO CURRENT SINGLE-BEST-RULE PROCEDURES THAT ONLY LOOK AHEAD, SWAP-1 CONSTANTLY LOOKS BACK TO SEE WHETHER IT CAN IMPROVE THE RULE BEFORE EXPANDING IT.

Swap-1

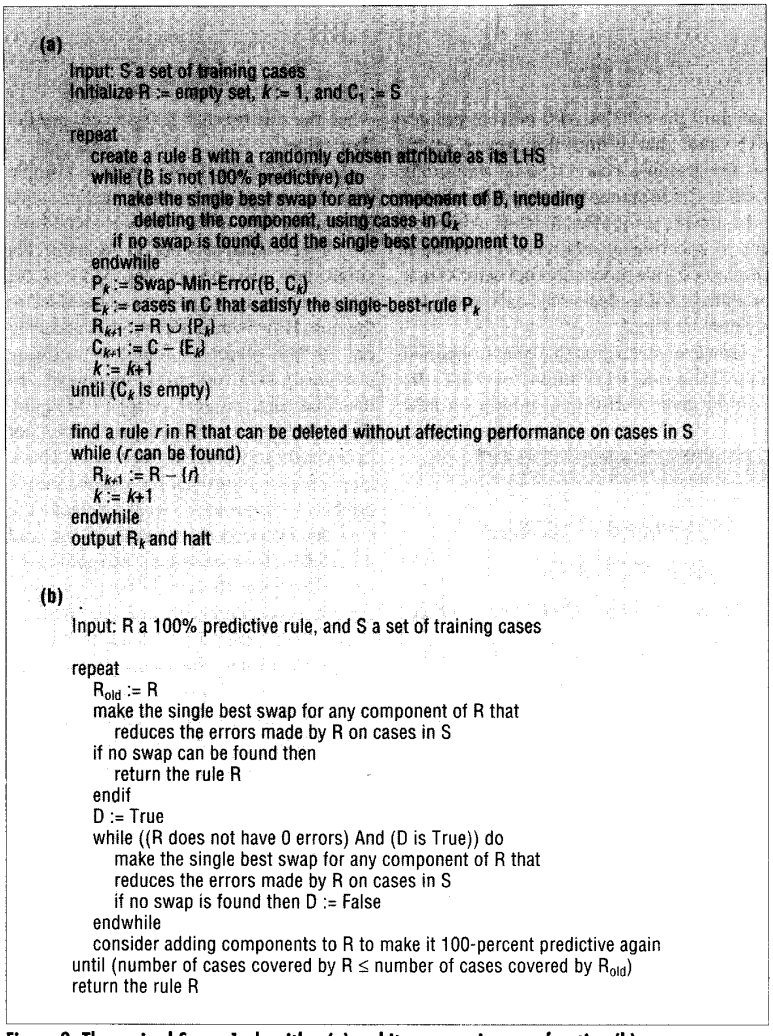
As opposed to current single-best-rule procedures that only look ahead, our Swap-1 procedure constantly looks back to see whether it can improve the rule before expanding it. To form the single best rule, Swap-1 first makes the single best replacement from among all possible rule component swaps, including deleting a component; if no swap is found, it adds the single best component to the rule (see Figure 1). "Best" is evaluated as predictive value, the percentage of correct decisions.[9] When the predictive values are equal, maximum case coverage is a secondary criterion. Swapping and adding components end when the rule reaches 100-percent predictive value.

As an example, Table 1 shows a single best rule being generated in seven steps. The initial rule is randomly assigned p3, which gets swapped out in favor of the single best test, p6. In step 3, p1 is the single best component that can be added to the rule. However, in step 4, p6 is swapped out for p4, which is found by refining previously selected rule components. In the final step, p3 gets swapped in again. Thus, if a test is swapped out, it does not necessarily stay out, but can be added back later on if it improves the current rule's predictive accuracy. Swap-1 selects the completed rule as the single best one, and proceeds with removing the covered cases and reapplying the single-best-rule construction procedure to the remaining cases. Classes are ordered in advance; Swap-1 completes all rules for a class before considering the next class.

Swap-1 has been compared to other rule-based methods for covering randomly generated expressions from uniformly distributed attributes.[14] It performs better than other rule induction methods and as well as Fringe, an iterative tree induction technique.[15]

However, when applied to real-world data with numerical attributes, Swap-1 can fragment data by covering with too many short rules. This is because it searches for the most compact 100-percent predictive rule. There might be longer rules that are also 100-percent predictive but cover more cases. Such rules are preferable because they result in more compact covering rule sets. To induce a longer rule R_j after obtaining a 100-percent predictive rule R_i, we swap on R_j for *minimum errors* (not predictive value) to obtain R_k and then reinitialize R_1 with R_k. The process of swapping for minimum errors might result in a rule that is not 100-percent predictive: Unlike evaluation based on predictive value, the cases in class C that are now not covered are considered errors. For example, a rule might be 100-percent predictive of a subset of cases for class C, but cover relatively few cases. Another rule with the same number of components might be less predictive but make fewer overall errors when the cases it does not cover are considered.

Once the procedure obtains the longer rule, it compares R_j with the shorter rule R_i for coverage, and then iterates if R_j covers more cases than R_i (see Figure 2). The



```
(a)
Input: S a set of training cases
Initialize R := empty set, k := 1, and C1 := S

repeat
   create a rule B with a randomly chosen attribute as its LHS
   while (B is not 100% predictive) do
      make the single best swap for any component of B, including
         deleting the component, using cases in Ck
      if no swap is found, add the single best component to B
   endwhile
   Pk := Swap-Min-Error(B, Ck)
   Ek := cases in C that satisfy the single-best-rule Pk
   Rk+1 := R ∪ {Pk}
   Ck+1 := C – {Ek}
   k := k+1
until (Ck is empty)

find a rule r in R that can be deleted without affecting performance on cases in S
while (r can be found)
   Rk+1 := R – {r}
   k := k+1
endwhile
output Rk and halt

(b)
Input: R a 100% predictive rule, and S a set of training cases

repeat
   Rold := R
   make the single best swap for any component of R that
      reduces the errors made by R on cases in S
   if no swap can be found then
      return the rule R
   endif
   D := True
   while ((R does not have 0 errors) And (D is True)) do
      make the single best swap for any component of R that
      reduces the errors made by R on cases in S
      if no swap is found then D := False
   endwhile
   consider adding components to R to make it 100-percent predictive again
until (number of cases covered by R ≤ number of cases covered by Rold)
return the rule R
```

Figure 2. The revised Swap-1 algorithm (a) and its swap-min-error function (b).

immediate objective of the revised Swap-1 algorithm is to find fewer but longer rules, each of which covers many more cases. The overall goal is to find the smallest covering set that separates a class from the others.

Finding the optimal combination of attributes and values for even one fixed-size rule is a complex task. In our approach, the central theme is to hold a model configuration constant except for a single local improvement to that configuration, repeating this process until no further improvements are possible. Making local changes to a configuration is a widely used optimization technique to approximate a global optimum and has been applied successfully, for example to find near-optimal solutions to traveling salesman problems.[16] Another related local optimization technique, called backfitting, has been used in the context of nonlinear statistical regression.[17] Thus, variations on selecting the next improvement could include either the first local improvement encountered (as in backfitting), or the best local improvement (as in Swap-1).

While the first option is more efficient, the second gives us consistently better results. Since the induced (and pruned) rule set environment is mostly stable with relatively few local improvements before convergence, the additional overhead associated with finding the best (as opposed to merely the first) local improvement is not overwhelming. However, in a less stable environment than ours, with large numbers of possible configuration changes, the second alternative might not be feasible.

Finding the rule set with the right complexity

The objective of the algorithms just described is to cover the data with a concise rule set. However, the goal of machine-learning methods is not just to discriminate among known samples, but to predict and generalize to future unseen samples. The statistics literature describes good techniques for estimating future performance that are based on principles of training on certain data and testing on independent data.

Some rules in a rule set can perform far better than others. Researchers have found that some rules do not generalize well because they are too specific to the characteristics of the original samples. Often these are rules that cover a relatively small number of cases in the training samples. One successful approach in many machine-learning methods is to throw out part of the decision model (in our case, remove some of the rules) and see whether performance improves. With highly predictive rules, the covering set might be best. In the worst situation, with poorly predictive features, we might do better to remove all rules and simply decide by always picking the largest class.

Thus we must accomplish two important tasks: decide which rules should be removed, and estimate the performance of what remains. The techniques for estimating performance are straightforward when independent test cases are available; the question of how to eliminate rules from a rule set is less obvious. It is impractical to test every possible combination of rules. Our induction model relies on *weakest-link pruning* to select the next rule to eliminate: The initial rule set is the covering rule set, from which we remove the weakest rule or component. Then we evaluate the resulting smaller rule set. Intuitively, the weakest rule has the least effect on overall performance. But how do we measure this? Later we will define this precisely, but for now, an effective measure is to take into account the number of eliminated components and the number of cases involved in errors that are created by eliminating that rule. We eliminate the rule (or component) that

Table 2. Example summary table.

RS_i, i=	Rules	Cx	Err_{app}	Err_{test}	Test SE	Mean(Cx)	WL_i
1	11	18	.0000	.1074	.0282	17.9	.0
2	10	15	.0083	.0909	.0261	14.9	.3
3	9	13	.0165	.0909	.0261	13.0	.5
4*	6	7	.0661	.0744	.0239	7.0	1.0
5	6	6	.0826	.1322	.0308	6.0	2.0
6	4	4	.1322	.1322	.0308	4.0	3.0
7	3	3	.2975	.2975	.0416	3.0	20.0
8	2	2	.5372	.5620	.0451	2.0	29.0
9	1	1	.6529	.6529	.0433	1.0	14.0

produces the fewest number of errors per eliminated component. For example, consider two candidate rules for pruning: one with three components, whose elimination would yield three new errors; and one with a single component, whose elimination would yield two errors. The first rule would be eliminated. Clearly, we could develop alternative measures that are based solely on the number of errors per eliminated rule, but combining errors and rule complexity in a single measure has strong empirical support in the literature and is consistent with the concept of reduced-complexity solutions.

In practice, we need not evaluate every pruned rule set that results from removing a single rule or component. Instead, we examine a limited number of pruned rule sets corresponding to rule sets with distinct degrees of performance on the training cases. The formal objective is to find a rule set that minimizes the true error rate on new cases, and this can be measured more directly by estimating the true error rates of varying-complexity rule sets using independent test cases. In practice, the optimal solution usually cannot be found because of incomplete samples and limitations on search time. Typically there are not enough cases to both train and accurately estimate a rule set's error rate. It is also impossible to search all possible rule sets of a particular complexity (such as the number of components in the rule set).

Several thousand independent test cases are sufficient to give highly accurate estimates of a classifier's error rate.[18] When fewer cases are available, resampling gives the most accurate estimates. Cross-validation[19] is generally the procedure of choice,[5] and tenfold cross-validation (the average results of 10 runs using 90-percent training and 10-percent testing cases, with 10 mutually exclusive test partitions) is usually quite accurate when the cases number in the hundreds. Because cross-validation techniques average the estimates for classifiers that are trained on about the same number of cases as the full sample, learning techniques have been developed that can train on all sample cases.

If the set $\{RS_1,\ldots RS_i,\ldots RS_n\}$ is ordered by some complexity measure $Cx(RS_i)$, then the best one is selected by $\min[Err(RS_i)]$. A practical alternative is the minimum-complexity rule set that is near the minimum error rate solution (within one standard error).[5] A method must induce and order $\{RS_i\}$ by $Cx(RS_i)$ and estimate each $Err(RS_i)$; this has been developed for decision trees. Minimizing a single complexity parameter in addition to the error rate estimator adds little bias to the estimates when used with resampling.[5]

We can use a variation of weakest-link pruning, also known as cost-complexity pruning, to prune a rule set and form $\{RS_i\}$.[5] Let the rule set RS_1 be the covering rule set. Find each subsequent RS_{i+1} by pruning RS_i at its weakest link. A rule set's weakest link can be defined as

$$WL_{i+1} = \min\left[\frac{Err(RS_k) - Err(RS_i)}{Size(RS_i) - Size(RS_k)}\right]$$

for all k prunes, where $Err(RS_i)$ is the number of errors that RS_i makes on the training cases, and $Size(RS_i)$ is the number of components in RS_i. The weakest link is the point at which the fewest new errors per deleted component are introduced. A rule set can be pruned by deleting single rules or components.[6] Repeated pruning of the least significant component or rule to RS_i forms $\{RS_i^k\}$, with a global minimum of $WL(i)$. The rule set at that point becomes the next RS_{i+1}. The process is repeated until the final RS_n is generated, where RS_n is the single-component rule that selects the largest class.

Thus, the application of weakest-link pruning results in an ordered series of decreasing-complexity rule sets $\{RS_i\}$, as shown in Table 2. The complexity of RS_i can be measured in terms of WL_i or $Size(RS_i)$. For a large rule set that is pruned, little is usually lost when the weakest component or rule is deleted: WL_i is relatively small. But when a small rule set is pruned, the effect of deleting a component is much greater, and this is reflected in a large WL_i.

With a large set of independent test cases and weakest-link pruning, we can estimate the true error rate of each rule set by its error rate on test cases. With smaller samples, where thousands of test cases are not available, resampling is preferable and more accurate: First we determine $\{RS_1,\ldots RS_i,\ldots RS_n\}$ by weakest-link pruning on the complete training set and then perform an n-fold (typically tenfold) cross-validation. We induce an auxiliary rule set in each iterative pass using the training set of that pass, and find by weakest-link pruning a new RS_i^k of about equal complexity. We then obtain test error rates using the test cases corresponding to that pass. The average of the error rates over all the passes for each rule set of $Size(RS_i)$, $Err_{cv}(RS_i)$, is the cross-validation estimate of the true error rate of RS_i.

Consistent with the minimum length description approach, each rule covers the original cases with only the weakest rules and components removed. Pruning a rule set is less stable and accurate than tree pruning because coverage of the pruned set is highly variable. While pruning a subtree retains full coverage of the data set, pruning rules can leave gaps. Moreover, for RS_i of a certain complexity, there might be a better RS_i' of the same size. Unlike decision tree induction where the nodes are fixed, RS_i can be refined by swapping single components to minimize the apparent training error $Err_{app}(RS_i)$. The process of refining any rule set RS into RS_i' can be described as modifying RS_i such that $Err_{app}(RS_i') \leq Err_{app}(RS_i)$ and $Cx(RS_i') \leq Cx(RS_i)$. The rules are iteratively checked for the best component deletion, rule deletion, or component swap. Here, *best* means minimum errors. Thus a rule set can be refined so that it is smaller than or equal in size to the original rule set and makes fewer or an equal number of errors.

The net result of this process is an error rate estimate for rule sets of varying complexity. Table 2 shows a typical result from resampling for rheumatic disease. For each rule set, the figure lists the number of rules

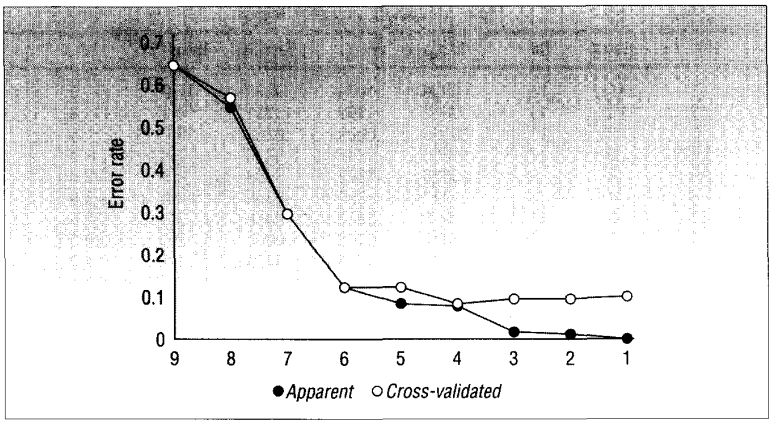


Figure 3. Error rates versus model complexity, with rule sets ordered by increasing complexity.

and rule components, the apparent error rate on the training cases, the test error rate (by cross-validation), the standard error of the test error, the average number of components over all cross-validated test sets, and the complexity measured by weakest-link pruning. Plotting the error rates for increasingly complex RS_i, as in Figure 3, illustrates the classical pattern of behavior for the apparent error rate on training cases versus the estimated true error on test cases. As model complexity increases, the apparent error rate decreases, but the true error rate flattens out and eventually increases.

Mixed models for reduced complexity

Our approach has been to minimize the complexity of rule sets. Although difficult to express in terms of a single unit, the true complexity of a solution can be further reduced by allowing for mixed models. This has led to a number of hybrid methods that incrementally embed alternative models within decision trees, such as piecewise linear discriminants[5] and perceptrons.[20]

In contrast to these nonparametric, incremental methods, we used the standard parametric linear discriminant found in all general statistical texts and software packages, and for many years the most widely used classification scheme.[21] Assuming normality and equal-covariance matrices, we derive the discriminant by solving for the linear function $f_i(e)$ of the set of attributes e for each class i not equal to j:

$$f_i(e) + \ln(P(C_i)) > f_j(e) + \ln(P(C_j))$$

where $P(C_i)$ is the prior probability of class i. Each function is characterized by a set of weights, one for each attribute. Our method classifies an unknown pattern by applying the functions (that is, multiplying each feature by a weight and adding up the score for each class), and choosing the class with the greatest magnitude. In addition, we can use heuristic stepwise feature selection to find a reduced-complexity linear solution by reducing the original set of features to a subset, thereby reducing the number of weights that are estimated. By adding relatively few weights, we expect to achieve greater accuracy, but the tradeoff is less intelligibility of the resulting solution.

The parametric stepwise linear discriminant has a strong theoretical foundation and has been rigorously tested over many years.[7,21] It tends to produce comparable results on training and test cases. In our design, we derive the discriminant function completely prior to rule induction, and use the training-case results to create artificial features. We create one binary higher order feature per class, where each feature is simply whether or not the linear discriminant selects that class during training. To the rule induction system, each of these higher order features is no different than any of the original features. Unlike previous approaches that use the numerical result of applying a linear function, here the encoding is simplified to a binary feature that preserves the classification result of the discriminant. This results in an interesting interplay between the linear discriminant and the original features in the induced rule set. For example, consider these mixed-model rules from a heart disease data set:

```
LD1 & RBPS > 109        → Class1
RBPS > 151 & Age < 62   → Class1
[True]                  → Class2
```

Class1 is chosen when both LD1 and RBPS > 109 are true, where LD1 is the selection of Class1 by the linear discriminant, and RBPS is a test result.

Results

The Swap-1 approach to automated rule learning has been explored in a wide variety of applications. In two small nondeterministic applications, the well-known iris data and an appendicitis data set, Swap-1 obtained the optimum rule sets.[8] In a study using artificial data with simulated noisy attributes, Swap-1 readily matched or exceeded the performance of other rule induction techniques.[14] Even more interesting is its performance on large-scale industrial applications with hundreds of features and thousands of samples, including predicting disk drive failure prior to final (expensive) testing,[22] automatically classifying electronic documents based on word patterns,[23] and diagnosing chronic problems in a long-distance telephone network.[24] The scope of looking for patterns in these large data sets exceeds human capacities. Swap-1 found valuable new decision rules, in effect automatically inducing knowledge bases from data.

Some of the applications we tested are proprietary, so we have taken a basic research approach here and compare our results with published results on nonproprietary data. The four real-world applications contained generally noisy data, and the best solutions all had substantial error rates. We wanted to see whether Swap-1 could find less complex but still clear and insightful solutions to these problems.

The applications compared different learning models, usually backpropagation neural nets and decision trees. We were particularly interested in these applications because decision trees performed relatively poorly compared to other learning models. Table 3 summarizes the characteristics of the four data sets and the training and test variations used to estimate the true error rates. Table 4 summarizes the results of previous studies for standard learning models and our results for a reduced-complexity approach. For neural nets and linear discriminants, the unit used to measure

Table 3. Data set characteristics for four applications. Lv-1 stands for leaving one out.

Data set	Training cases	Test cases	Feature type	Features	Classes in training set
Nettalk	7,229	7,242	Boolean	203	99
Heart	297	(10) 70%/30%	Numerical	13	2
DNA	106	Lv-1	Boolean	228	2
Rheum	121	Lv-1	Boolean and numerical	140	5

complexity was the number of weights; for rules, it was the number of propositions; and for trees, it was the number of nodes. While the complexity of different models is not strictly comparable, these are the most natural measures for each model.

Text-to-speech recognition. The Nettalk text-to-speech application addresses the problem of recognizing "letters," or classes of phoneme-stress pairs.[25] Although previous studies used a relatively complex nonmutually exclusive encoding for classes (based on domain knowledge and suitability for a neural-net representation),[26] we used the traditional classification mold of mutually exclusive classes. The training set contains 47 phoneme classes and four stress classes but only 99 phoneme-stress pairs (letters) with at least two cases. Even though the set comprises more than 7,000 cases, many of these 99 classes have relatively few cases in them. The rule sets $\{RS_i\}$ are induced from the training set. Given the large number of test cases, the appropriate complexity fit $Cx(RS_i)$ can be determined. The classes were ordered by the decreasing predictive value of their rule sets.

The previously reported best result was for a neural net with an error rate of .291 and more than 27,000 weights. Swap-1 found a reduced-complexity rule-based solution with an error rate of .260 and 663 rule components. (With 99 classes, we were unable to get a useful linear discriminant.) The first reported error rate for a tree-based solution was a relatively weak .344, later improved to .292 when a single relatively unpruned tree was induced for all 126 classes in the training set.[27] However, with the 99-class representation and some pruning, the error rate of a Cart-induced tree was the same as the Swap-1 rule-based solution, although with greater complexity. A slightly better result of .256 was reported for trees using a 157-bit error-correcting code that induced a separate tree for each bit, yielding a total complexity of 207,804 nodes.[27]

Classifying heart disease. A heart disease application[28] compared ID3 decision trees with backpropagation neural nets, using the average test results for 10 experiments. In each experiment, 70 percent of the cases were randomly selected for training and the rest were used for testing.[29] The best result was for a neural net with an error rate of .194 and 86 weights. The result of a tenfold (and a threefold) cross-validation with Swap-1 yielded a simple four-rule, six-component solution with an estimated error rate of .215, as compared with .288 for the decision tree. We obtained better results by pruning the rule sets and not using binary encodings of numerical variables. When we added the linear discriminant to the original features, it was the best rule set, with 16 weights and an estimated error rate of .176 (the average of three threefold cross-validations). The estimate for a tenfold cross-validation was .168.

DNA pattern analysis. Researchers also compared several learning methods on a DNA pattern recognition task.[30] In this study, a human expert's theory (described in terms of a grammar) was reformulated as a neural network and refined, yielding somewhat better results than pure empirical learning systems. (We used the leaving-one-out method, in which a single case is used as a test case, and the remaining cases are used for training. This is repeated for all cases in the sample. Using this method, we estimated the human-assisted solution at an error rate of 0.038.) The best learning result was for a neural net with an error rate of .075 and more than 3,600 weights. The tree solution did relatively poorly, with an error rate of .179; a reduced-complexity three-rule solution with five components performed better (with an error rate of .132). Still, the rule-based solution was not fully competitive. By including the results of the linear discriminant in the feature set, the researchers obtained an induced rule set that was pruned back to the linear discriminant alone. This function had only 12 weights and an error rate estimate of .047; its reduced complexity was due to the success of stepwise feature selection. With 228 features and only 106 cases, there were far too many weights to be estimated if all the features were used, so a reduced-complexity discriminant was desirable.

Diagnosing rheumatic disease. More than a decade ago, researchers described and evaluated a rule-based expert system for diagnosing rheumatic diseases.[31] The knowledge base and data were later used in heuristic refinement systems that could modify a theory (that is, an expert-specified rule base) to improve its performance.[32] These systems were restricted to minor refinements that would assure the preservation of the expert's knowledge base close to its original form. These same data were used to evaluate an alternative theory revision approach, called RTLS, which used the expert knowledge base as a starting

Table 4. Comparison of best reported, decision tree, and Swap-1 results. NN stands for neural net. Cx-LD stands for complexity of linear discriminant.

Data set	Previously reported best results			Tree results		Swap-1 results						
						Rules only			Rules with discriminants			
	Err	Type	Cx	Err	Cx	Err	Rules	Cx	Err	Rules	Cx of rules	Cx-LD
Nettalk	0.291	NN	27,626	0.292	5,303	0.260	253	663				
Heart	0.194	NN	86	0.288	109	0.215	4	6	0.1766	2	2	16
DNA	0.075	NN	3,698	0.179	25	0.132	3	5	0.0472	2	2	12
Rheum	> 0.074	Rule	3,000	0.372	63	0.074	6	7	Not applicable			

Table 5. CPU times for swapping rule components on a Sparc-10.

Data set	Find a covering set on all cases	Perform a full analyses
Nettalk	1 hour	3 hours
Heart	2 seconds	21 seconds
DNA	1 second	5 seconds
Rheum	1 second	5 seconds

point and allowed unrestricted changes to the rule set.[33] The error rate for the five-class problem was estimated at zero. Because this new rule set was radically different from the expert's knowledge base, it was worth considering how well a purely empirical learning system might do.

The RTLS solution was far too complex (with hundreds of rules and thousands of components to provide a good complexity fit to the data. The true error rate of RTLS was also higher than cited, because error rates were measured in a nontraditional way. Some ties were scored as correct, and answers were marked correct when they agreed with the output of a second refinement program, Seek2. However, the output of Seek2 had an error rate estimate of .074, so the true error rate of the refined rule set must be greater than that.

Most empirical learning systems could not handle this data set because almost half the feature values were missing. Inducing a decision tree using Cart (with its surrogate strategy for handling missing values) yielded a high error rate of .405. Swap-1 found a six-rule, seven-component solution with an estimate of .074. Even with missing values, the covering of the 121 cases was quite compact. With so many missing values, the advantage of the rule-based solution over the decision tree can be traced to the rules' nonmutual exclusivity. (Although the results were quite good, caution is still warranted. With missing values, there is always the question of whether a prediction is based on a feature value or on the event that a value is missing.)

Computational timings. Swapping components in a large rule set might appear to be computationally prohibitive. This article has presented the algorithms in a way that clarifies their functionality, but in practice they can be encoded far more efficiently (though less intuitively) to take advantage of the strong computational bounds. Thus, these computations are usually quite tractable. Table 5 shows the CPU times for finding a covering set on all cases and for doing a complete analysis, including full pruning and swapping, on a Sparc-10 workstation. For all data sets except Nettalk, the times are for a tenfold cross-validation; for Nettalk, the time is for a single training and test cycle.

FOR SOME APPLICATIONS, SUCH as parity checking, a disjunctive normal form model might be weaker than a neural net, since the latter can theoretically approximate any function. In practice though, a DNF model induced from data can often outperform other types of models, including neural nets, by more efficiently searching a restricted solution space. Our reduced-complexity rule induction approach performed as well as or better than previous solutions in terms of estimated error rates, but equally important, they were far less complex than previously reported solutions.

However, in two instances, the pure rule-based solution was weaker than the neural nets. By combining the results of the stepwise parametric linear discriminant with the original feature set, we were able to exceed previous results at the expense of a mixed-model solution. Solutions based on incorporating a traditional classifier model into a rule-based approach can offer new insight into an application and are fully compatible with knowledge-based systems.

While our approach is directed toward a rule-based solution, the notion of minimizing complexity is not restricted to any particular model. Neural-net solutions that are compact and minimize the number of weights are also likely to increase predictive accuracy. For the applications cited here, similar simplifications to other models might yield improved results. For example, the Nettalk model might benefit from a reduced number of weights. The limiting factor is not only the specific model, but the effectiveness of the learning technique and computational time.

With increasingly available computational power, we can look forward to more computationally intensive attempts to extract the maximum amount of information from sample data, since computation is clearly far less expensive than human effort. For applications with very large databases, these intensive computational efforts might supplement or even replace human knowledge-engineering efforts.

References

1. C. Wallace and P. Freeman, "Estimation and Inference by Compact Encoding," *J. Royal Statistical Soc. Series B.*, Vol. 49B, No. 3, 1987, pp. 240-265.
2. J. Rissanen, "Modeling by Shortest Data Description," *Automatica*, Vol. 14, 1978, pp. 465-471.
3. J. Rissanen, "Stochastic Complexity." *J. Royal Statistical Soc. Series B.*, Vol. 49, No. 3, 1987, pp. 223-239.
4. R. Holte, "Very Simple Classification Rules Perform Well on Most Data Sets," Tech. Report TR-91-16, Computer Science Dept., U. of Ottawa, Canada, 1991. To be published in *Machine Learning*.
5. L. Breiman et al., *Classification and Regression Trees*, Wadsworth, Monterrey, Calif., 1984.
6. J. Quinlan, "Simplifying Decision Trees," *Int'l J. Man-Machine Studies*, Vol. 27, 1987, pp. 221-234.
7. M. James, *Classification Algorithms*, John Wiley & Sons, New York, 1985.
8. S. Weiss and I. Kapouleas, "An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods," *Int'l Joint Conf. Artificial Intelligence (IJCAI-89)*, Morgan Kaufmann, San Mateo, Calif., 1989, 781-787.
9. S. Weiss, R. Galen, and P. Tadepalli, "Maximizing the Predictive Value of Production Rules," *Artificial Intelligence*, Vol. 45, 1990, pp. 47-71.
10. J. Quinlan, "Generating Production Rules from Decision Trees," *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI-87)*, Morgan Kaufmann, San Mateo, Calif., 1987, pp. 304-307.
11. P. Clark and T. Niblett, "The CN2 Induction Algorithm," *Machine Learning*, Vol. 3, 1989, pp. 261-283.
12. G. Pagallo and D. Haussler, "Boolean Feature Discovery in Empirical Learning," *Machine Learning*, Vol. 5, No. 1, 1990, pp. 71-99.
13. R. Michalski et al., "The Multipurpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains," *Nat'l Conf. Artificial Intelligence (AAAI-86)*, MIT Press, Cambridge, Mass., 1986, pp. 1,041-1,045.
14. N. Indurkhya and S. Weiss, "Iterative Rule Induction Methods," *Applied Intelligence*, Vol. 1, 1991, pp. 43-54.
15. G. Pagallo, "Learning DNF by Decision Trees," *Proc. Int'l Joint Conf. Artificial*

Intelligence '89, Morgan Kaufmann, San Mateo, Calif., 1989, 639-644.

16. S. Lin and B. Kernighan, "An Efficient Heuristic for the Traveling Salesman Problem," *Operations Research*, Vol. 21, No. 2, 1973, pp. 498-516.
17. T. Hastie and R. Tibshirani, *Generalized Additive Models*, Chapman and Hall, London, 1990.
18. W. Highleyman, "The Design and Analysis of Pattern Recognition Experiments," *Bell System Tech. J.*, Vol. 41, 1962, pp. 723-744.
19. M. Stone, "Cross-Validatory Choice and Assessment of Statistical Predictions," *J. the Royal Statistical Society*, Vol. 36, 1974, pp. 111-147.
20. P. Utgoff, "Perceptron Trees: A Case Study in Hybrid Concept Representation," *Proc. Nat'l Conf. Artificial Intelligence (AAAI-88)*, Morgan Kaufmann, San Mateo, Calif., 1988, pp. 601-606.
21. R. Fisher, "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, Vol. 7, 1936, pp. 179-188.
22. C. Apte, S. Weiss, and G. Grout, "Predicting Defects in Disk Drive Manufacturing: A Case Study in High-Dimensional Classification," *Proc. IEEE Computer Soc. Conf. on Artificial Intelligence for Applications (CAIA-93)*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1993, pp. 212-218.
23. C. Apte, F. Damerau, and S. Weiss, "Automated Learning of Decision Rules for Text Categorization," to be published in *ACM Trans. on Office Information Systems*, 1994.
24. R. Sasisekharan, V. Seshadri, and S.M. Weiss, "Proactive Network Maintenance Using Machine Learning," to appear in *Proc. IEEE Globecom '93*, IEEE Service Center, Piscataway, N.J., 1993.
25. T. Sejnowski and R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, Vol. 1, 1987, pp. 145-168.
26. T. Dietterich, H. Hild, and G. Bakiri, "A Comparative Study of ID3 and Backpropagation for English Text-to-Speech Mapping," *Proc. Seventh Int'l Conf. Machine Learning*, Morgan Kaufmann, San Mateo, Calif., 1990, pp. 24-31.
27. T. Dietterich and G. Bakiri, "Error-Correcting Output Codes: A General Method for Improving Multiclass Inductive Learning Programs," *Proc. Nat'l Conf. Artificial Intelligence '91*, Morgan Kaufmann, San Mateo, Calif., 1991, pp. 572-577.
28. R. Detrano et al., "Int'l Application of a New Probability Algorithm for the Diagnosis of Coronary Artery Disease," *Amer. J. Cardiology*, Vol. 64, 1989, pp. 304-310.
29. J. Shavlik, R. Mooney, and G. Towell, "Symbolic and Neural Learning Algorithms: An Experimental Comparison," *Machine Learning*, Vol. 6, 1991.
30. G. Towell, J. Shavlik, and M. Noordeweir, "Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks," *Proc. Nat'l Conf. Artificial Intelligence '90*, Morgan Kaufmann, San Mateo, Calif., 1990, pp. 861-866.
31. D. Lindberg et al., "Computer-Based Rheumatology Consultant," Medinfo-80: *Proc. Third World Conf. Medical Informatics 1980*, North Holland, Amsterdam, 1980, pp. 1,311-1,315.
32. A. Ginsberg, S. Weiss, and P. Politakis, "Automatic Knowledge Base Refinement for Classification Systems," *Artificial Intelligence*, Vol. 35, 1988, pp. 197-226.
33. A. Ginsberg, "Theory Reduction, Theory Revision, and Retranslation," *Proc. Nat'l Conf. Artificial Intelligence '90*, Morgan Kaufmann, San Mateo, Calif., 1990, pp. 777-782.

Sholom M. Weiss is a research professor of computer science at Rutgers University and the author of *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems* (Morgan Kaufmann, 1991). His current research interests emphasize machine learning from data. He is a fellow of the American Association for Artificial Intelligence, serves on numerous editorial boards, including that of IEEE Expert, and has extensive industrial collaborations on the practical application of machine-learning techniques.

Nitin Indurkhya is a research scientist in the Artificial Intelligence Section at Telecom Australia Research Laboratories. He received his BS in computer science and engineering from the Indian Institute of Technology, Kanpur, India, and his MS and PhD in computer science from Rutgers University. His current research focuses on automatic model construction and time-series data analysis, with particular interest in hidden Markov modeling, rule-based induction, decision trees, neural nets, and nonparametric statistics. He is a member of AAAI.

The authors can be reached in care of Weiss at the Department of Computer Science, Rutgers University, New Brunswick, NJ 08903; Internet, weiss@cs.rutgers.edu