Reducing Misclassification Costs

Michael Pazzani, Christopher Merz, Patrick Murphy
Kamal Ali, Timothy Hume and Clifford Brunk
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
{pazzani, cmerz, pmurphy, ali, hume, brunk}@ics.uci.edu

Abstract

We explore algorithms for learning classification procedures that attempt to minimize the cost of misclassifying examples. First, we consider inductive learning of classification rules. The Reduced Cost Ordering algorithm, a new method for creating a decision list (i.e., an ordered set of rules) is described and compared to a variety of inductive learning approaches. Next, we describe approaches that attempt to minimize costs while avoiding overfitting, and introduce the Clause Prefix method for pruning decision lists. Finally, we consider reducing misclassification costs when a prior domain theory is available.

1 INTRODUCTION

Many learning programs create procedures (e.g., decision trees (e.g., Quinlan, 1986) or rules (e.g., Clark and Niblett, 1989)) whose goal is to minimize the number of errors made when predicting the classification of unseen examples. Here, we investigate approaches to learning or revising classification procedures that attempt to reduce the cost of misclassified examples rather than the number of misclassified examples. The general idea is that in many problems, the cost of all errors is not equal. Rather, the cost of making an error can depend upon both the predicted class of the example and the actual class of an example. In this paper, we introduce three algorithms that we have developed to minimize classification costs: the Reduced Cost Ordering for creating decision lists, the Clause Prefix method for avoiding overfitting in decision lists, and a greedy method for revising numeric parameters in a rule based expert system.

This research was motivated by a problem of learning to troubleshoot a telephone network using a hypothetical cost data set provided by NYNEX (the primary local phone company for New York and New England). The training data contains information on the type of switching equipment, and various voltages and resistance measurements. The task is to predict the location to which a repairman should be dispatched (e.g., the problem is in the customer's wiring (PDO), the cable facilities (PDF), or the central office (PDI)). In addition, an example may be classified as a PDT in which some additional testing must be performed. NYNEX has implemented a rule-based expert system, MAX (Rabinowitz, et al., 1991) that is used to determine the location of a malfunction for customer-reported telephone troubles. Like all expert systems, MAX requires occasional maintenance to its knowledge base. In addition, MAX is used at many different sites in New York and New England and there are small differences in how examples should be classified at each site. The designers of MAX have facilitated this customization by having a set of numeric parameters (e.g., indicating when a voltage is too high) that are set at each site or adjusted periodically to improve its performance. The rule base for MAX can serve as an important source of domain knowledge for knowledge-intensive learning programs by biasing the learning process. The training and test data for the learning programs are examples that have been processed by MAX. The classification of the examples is determined by the technician who actually solved the problem in the field. However, this classification is subject to a variety of errors (Danyluk & Provost, 1993) that necessitate the use of methods to avoid overfitting the data.

In the telephone network troubleshooting problem some of the classes are easier (i.e., less expensive) to attempt to repair than others. Mistaking a simple repair for a more complex one can be quite expensive (e.g., by dispatching a repairperson to the wrong location and incorrectly replacing an expensive functional component) compared to mistaking a complex repair for a simple one. An additional complexity arises because some repairs are similar, so that mistaking one expensive repair for another may not be very expensive (e.g., if both repairs involve dispatching a repairperson to the same location). After the methods were developed for this problem, we also evaluated them on several problems from the UCI Repository of Machine Learning Databases such as the diagnosis of heart diseases, the diagnosis of diabetes, and the identification of poisonous mushrooms.

	PDT	PDI	PDO	PDF
PDT	0	126	142	173
PDI	122	0	156	187
PDO	135	153	0	200
PDF	160	178	194	0

	H	S1	S2	S3	S4
H	0	1	2	3	4
S1	2	0	1	2	3
S2	4	2	0	1	2
S3	6	4	2	0	1
S4	8	6	4	2	0

Table 1. (Upper). A hypothetical cost matrix for the NYNEX telephone network troubleshooting problem. The rows are actual classes and the columns are predicted classes. For example, the cost of predicting class PDT when the example actually belongs to class PDI is 122. (Lower) A cost matrix for the Detrano heart disease database.

In the remainder of this paper, we first introduce some preliminary definitions related to the cost of misclassifying an example. Next we discuss inductive learning in the context of minimizing misclassification costs. Then, we describe methods for avoiding overfitting. Next, we discuss reducing misclassification costs in a special purpose "theory revision" system for revising numeric parameter values in an expert system for troubleshooting a telephone network. Then, we describe reducing misclassification costs in FOCL, a general purpose knowledge-intensive learning system.

2 DEFINITIONS

A classifier is typically evaluated by estimating its error rate from a sample of test data by finding the proportion of examples that are incorrectly classified. We will evaluate classifiers by looking at the cost of the errors made. The cost of misclassifying an example will be a function of the predicted class and the actual class. We will represent this function as a cost matrix, cost(actual-class, predicted-class). The cost matrix will be an additional input to the learning programs and will also be used to evaluate the ability of the learned procedure to reduce misclassification costs. Table 1 shows the cost matrix that we will use for the telephone network troubleshooting problem. Table 1 also contains a cost matrix we will use for the Detrano heart disease database. This database has 5 classes, H (healthy), S1 (slightly ill), S2 (moderately ill), S3 (very ill) and S4 (extremely ill). This cost matrix reflects our intuition that it is more costly to underestimate how ill someone is than to overestimate, and that it less costly to be slightly wrong than very wrong. For the diagnosis of diabetes (a two class problem), mistaking a healthy patient for one with diabetes had a cost of 1 and the cost of mistaking a patient with diabetes for a healthy patient had a cost of 2. For the mushroom classification problem, another two class problem, mistaking a poisonous mushroom for an edible one had a cost of 10, while mistaking an edible mushroom for a poisonous one had a cost of 1.

Note that costs may have any units. In the telephone troubleshooting problem, costs represent dollars expended (although the cost matrix shown in Table 1 that we use does not contain the actual costs which are proprietary). In the heart disease case, the costs represent the importance of avoiding each type of error. Error may be viewed as a special case of cost by defining a cost matrix that is 0 along the diagonal (when actual-class(i) = predicted-class(i)) and 1 otherwise. We'll call such a cost matrix a uniform cost matrix. We'll define a metric, average error per example that we'll use to evaluate learners:

$$\text{average-cost} = \frac{\sum_{i}^{N} \text{cost(actual-class(i), predicted-class(i))}}{N}$$

We'd like to distinguish a cost matrix from what we'll call a cost vector that has been used in some previous learning research (Miyauchi & Rendell, 1992; Provost & Buchanan, 1992; Provost, 1994; Tcheng, Lambert, Lu & Rendell, 1989) which has incorporated cost into the inductive learning process. With a cost vector, the cost of misclassifying an example depends on either the actual class of the example or the predicted class but not both.

A simple algorithm for predicting classes might guess the most frequently occurring class in the training data. An analogous method, guessing the least expected cost class, minimizes the average cost of guessing class *c* on all training examples:

$$\frac{\sum_{i}^{N} \text{cost(actual-class(i), c)}}{N}$$

Clearly, if a learning algorithm is intended to reduce the cost of misclassification, we'd like it to have a lower average cost than simply guessing the least expected cost class. In addition, some of the learning and pruning methods proposed will identify the least expected cost class of a set of training examples. For example, in pruning a decision tree, the class label associated with a node will be the least expected cost class of the examples at that node.

3 INDUCTIVE LEARNING WITH MISCLASSIFICATION COSTS

We consider three alternative approaches to making inductive learners sensitive to misclassification costs. First, we investigate alternative heuristic evaluation functions for decision tree learners. Next, we look at learners that predict the probabilities that examples belong to each class. Given this information, a simple application of statistical decision theory (e.g., Berger,

1980) can be used to assign an example to the class with the least expected cost of an error. Finally, we investigate algorithms that given an unordered set of rules, produce an ordered decision list and select the order of the rules in an attempt to minimize classification errors.

3.1 SELECTING TESTS IN A DECISION TREE

Brieman et al. (1984) advocate two methods for incorporating costs into the test selection process of a decision tree. One method, referred to as variable misclassification costs, does not work unless there are more than two classes. Here, we evaluate the altered priors method which works with any number of classes. See Section 4.4.3 of Brieman et al. (1984) for details on this method. Briefly, it operates by replacing the term for the prior probability ($\pi(j)$) that an example belongs to class j with an altered probability $\pi'(j)$:

$$\pi'(j)=\frac{C(j)\pi(j)}{\sum_i C(i)\pi(i)} \text{ where } C(j)=\sum_i cost(j,i)$$

To our knowledge, this metric has not been subject to rigorous experimentation. We compared learning decision trees using the GINI criterion with and without the altered priors methods. We compared the two methods on the telephone troubleshooting problem (learning on 600 examples and testing on 400), the Pima Indians diabetes database (training on 440 examples and testing on the remainder), the mushroom database (training on 500 and testing on 500) and the Detrano Heart Disease database (training on 200 and testing on the reminder). The results are summarized in Table 2. We ran 50 trials of each algorithm to obtain the mean cost and accuracy. The altered priors method differs little from using the unaltered priors and in one case (mushrooms) it significantly increases the cost of errors (at the 0.05 level using a paired, two-tailed t-test). One possible explanation for the poor results is that this method requires converting a cost matrix cost(j,i) to a cost vector C(j) resulting in a single quantity to represent the importance of avoiding a particular type of error. Accurately performing this conversion is nontrivial since it depends both on the frequency of examples of each class as well as the frequency that an example of one class might be mistaken for another.

We also looked at the obvious method of using the cost of misclassification as a test selection metric. For each possible test, the partitions of the training data made by that test are found. For each partition, the class that minimizes misclassification costs is found, and the cost of the errors of the partition is computed. The test that minimizes the sum of the costs of all partitions is selected. We compared the accuracy and misclassification cost of selecting tests using this approach, which we call Cost-Minimization to the information gain metric of ID3. For a control, we also ran an analogous approach with a uniform cost matrix. We call this approach Error-Minimization in Table 2. The results of experiments run in the same manner as those described previously indicate that selecting tests in a greedy fashion in an attempt to minimize costs does not have the desired effect of minimizing cost when compared to standard decision tree induction methods. Note that selecting tests to minimize error does not perform as well at minimizing error as selecting tests to maximize information gain. Adding weights to reflect costs does not appear to change this fact. Both minimizing error and minimizing costs may fail as selection criteria since they solely attempt to fit the data, without attempting to minimize the complexity of the learned concept. In all cases, the information gain metric produced more compact trees than the Error-Minimization and Cost-Minimization methods.

3.2 ESTIMATING CLASS PROBABILITIES

Bayesian classifiers (e.g., Duda & Hart, 1973) estimate the probability that an example belongs to each class. Similarly, a decision tree can also produce an estimate of the probability that an example belongs to each class given that it is classified by a particular leaf by examining the classes of the training examples at the leaf. Here, we use a Laplace estimate of the probability of a class given a leaf. If there are N_i examples of class i at a leaf and k classes, then the probability that an example at the leaf is a member of class i is given by:

$$p(class = i) = \frac{N_i+1}{k + \sum_j^k N_j}$$

For example, if there are 2 examples of class H, 7 examples of class S1, 1 example of S2, and 0 examples of classes S3 and S4 at a leaf of a tree to predict heart disease, the Laplace estimate of the probability that an example classified at that leaf should be assigned to the various class is 3/15 for H, 8/15 for S1, 2/15 for S2, 1/15 for S3 and 1/15 for S4. Typically, an example is assigned to the class with the lowest expected error. Given these probabilities, it is straightforward to compute the expected cost of an error if an example were assigned to each class, (e.g., by using the values in Table 1, the expected cost of predicting class S2 is 4*3/15 + 2*8/15 + 1*1/15 + 2*1/15). An example is assigned to the class with the least expected error. This strategy has an advantage over the approaches of changing the decision tree metrics to reflect costs in that the cost matrix does not effect the learned concept description so that a different cost matrix may be used for different test examples (e.g., not all patients would place the same cost on various potential outcomes of a risky medical treatment).

Table 2 also presents the results of learning with probabilistic classifiers. Bayes-Error is the standard Bayesian classifier while Bayes-Cost assigns an example to the least expected cost class as a function of the probability estimates returned by the classifier. The decision tree algorithm described in this section is called

Algorithm	Heart Disease		Diabetes		Mushroom		Telephone	
	Accuracy	Cost	Accuracy	Cost	Accuracy	Cost	Accuracy	Cost
Decision Tree Metrics								
I-gain	.500	1.19	.702	.447	.995	.043	.373	105.1
GINI	.496	1.20	.695	.458	.995	.044	.375	104.7
GINI (Altered Priors)	.492	1.19	.692	.476	.987	.063	.375	104.6
Cost-Minimization	.487	1.24	.668	.507	.993	.050	.374	104.7
Error-Minimization	.457	1.24	.665	.494	.997	.026	.374	105.0
Estimating Class Probabilities								
Bayes-Cost	.544	0.84	.742	.370	.951	.390	.299	113.6
Bayes-Error	.558	0.92	.752	.391	.937	.597	.301	113.9
I-gain (Cost-Laplace Prob)	.414	1.16	.697	.449	.957	.062	.309	103.0
Decision List Ordering								
Reduced Cost Ordering	.454	1.14	.684	.417	.959	.153	.355	101.3
Reduced Error Ordering	.531	1.68	.716	.504	.977	.217	.408	106.7
Better Safe Than Sorry	.403	1.39	.676	.412	.971	.155	.352	105.3
Pruning								
RCO+Clause Prefix	.532	1.04	.681	.378	.966	.138	.391	97.3
I-gain+RCP	.468	1.20	.696	.419	.981	.101	.375	103.0
I-gain+REP	.514	1.26	.717	.449	.984	.130	.385	103.5
I-gain+RCP (Full Training)	.493	1.08	.718	.398	.986	.096	.407	97.7
Guessing a Single Class								
Most Frequent Class	.541	1.89	.650	.700	.528	4.720	.393	105.4
Least Expected Cost Class	.179	1.51	.350	.650	.472	.528	.393	105.4

Table 2. Accuracy and Cost of Inductive Learning Algorithms

"I-gain (Cost-Laplace Prob)" in Table 2 since it uses information gain to build trees, and minimizes costs by using a Laplace estimate of the probability of a class given a leaf. With the exception of variants of the Reduced Cost Ordering algorithm on the diabetes database, the Bayes-Cost algorithm has costs that are much lower than the other algorithms tried on the heart disease and diabetes databases ($p < .05$ level using paired two-tailed t-tests). However, on the mushroom and telephone troubleshooting data, its costs are considerably higher than the other algorithms. It appears that when the Bayesian classifier is accurate (perhaps when the data do not violate the independence assumption and there are few irrelevant features), it does very well at minimizing costs, but when it is not accurate (perhaps due to feature interaction and irrelevant features), it does not perform well at minimizing costs. In contrast, estimating class probabilities with the decision tree learner does not significantly reduce error compared to a standard decision tree learner, suggesting that the Laplace probabilities of a class given a leaf are not very accurate, since they tend to be based on a few examples.

3.3 DECISION LISTS

One objective of this research was to minimize the misclassification costs of FOCL, a knowledge-intensive rule learning program that has been applied to the NYNEX MAX data and knowledge-base. FOCL, like most inductive logic programming systems, was designed to deal with two-class problems in which it learns clauses to cover the positive examples. At test time, any example that does not satisfy a learned clause is considered negative. For multi-class problems, FOCL has been modified to learn a set of clauses for each class. This is done by running FOCL once for each class, treating the examples of that class as positive examples and the examples of all other classes as negative examples. This results in a set of clauses for each class. A policy must be chosen to deal with the situation in which clauses of more than one class are satisfied by a test example and when no clause is satisfied by a test example. In our previous work, (Pazzani & Brunk, 1993) we created a decision list (i.e., an ordered set of clauses) using a fixed ordering of classes and a default class provided by a domain expert. The classes were ordered so that the rules of classes with the least expensive average errors are checked first. Provost and Buchanan (1992) call this the better safe than sorry strategy. Attempts to change the evaluation function of FOCL in an analogous manner to the evaluation function of the decision tree learners also failed to reduce classification costs. Therefore, we adopted a strategy that would serve as a post-processor to FOCL (or any rule learner such as C4.5 (Quinlan, 1993) or CN2 (Clark & Niblet, 1992)) that selects and orders the learned rules to minimize misclassification costs. Here, we introduce an algorithm, Reduced Cost Ordering

```
Given:
    Pool- an ordered set of rules of the form (class_i, conjunction_i)
    CostMatrix- cost of misclassifying examples of Class_i as Class_j
    Examples- a set of classified examples

Produce
    DecisionList- an ordered decision list

DecisionList = ()
DefaultClass = LeastExpectedCostClass(Examples, CostMatrix)
LowestCostSofar = Cost(Examples, CostMatrix, DefaultClass)
Remaining Examples = Examples

AddRules: FindBestRule(RemainingExamples, CostMatrix, Pool,
                       NewRule, NewDefaultClass)
           Add NewRule and NewDefaultClass to end of DecisionList
          NewCost = Cost(Examples, CostMatrix, DecisionList)

If NewCost < LowestCostSofar
   then Remove NewRule from Pool
        Remove examples that satisfy NewRule from Remain Examples
        DefaultClass = NewDefaultClass
        Remove NewDefaultClass from DecisionList
        LowestCostSofar = NewCost
        GoTo AddRules
   else Add DefaultClass to the end of DecisionList
        Return DecisionList
```

Table 3. The procedure for creating an ordered decision list.

(RCO), that takes as input a set of unordered rules for each class and produces an ordered decision list that uses a subset of these rules.

Table 3 presents an overview of RCO. The algorithm initializes the decision list to a default rule that guesses the least expected cost class. Next, it iteratively tries to improve upon the current decision list with an operator that replaces the default rule with a new rule and a new default rule. The procedure `FindBestRule` tests the impact of adding each remaining rule to the end of the current decision list and assigning the examples that match no rule to the least expected cost class of the unmatched examples. It returns the rule and default class with the lowest cost. The intuition behind this algorithm is that it favors adding a new rule to the end of the current decision list that makes inexpensive errors and that correctly classifies many examples that are currently incorrectly classified by the default rule. Since adding a new rule can change the distribution of uncovered examples, it also can update the default class. The complexity of the algorithm is $O(R^2E)$ where R is the number of rules and E is the number of examples since the decision list contains at most R rules and determining which rule to add to the decision list is O(RE).

Using the same examples to learn the initial pool of rules and to create the ordered decision list can cause a problem since the learned rules rarely make errors on the data used to learn the rules. Therefore, the cost of errors of such rules would be underestimated. In our experiments, we always divide the training data into a learning set consisting of 2/3 of the training data and an ordering set consisting of the remaining 1/3 of the training data. This may be a disadvantage of this algorithm especially on small training sets. Table 2 presents the results of Reduced Cost Ordering using FOCL to learn rules. Table 2 also includes the results of the Better Safe Than Sorry strategy, and an algorithm we call Reduced Error Ordering which is RCO with a uniform cost matrix in an attempt to minimize error. The RCO algorithm has equivalent or lower cost than the other the decision list policies on these databases and is lower than any of the decision tree variants described in Section 3.1 except on the mushroom database. Decision tree learners have an advantage over rule learners on the mushroom database, because there is one multi-valued feature that is very predictive on this problem. Decision trees can take advantage of this property to learn a single multi-valued split, while rule learners typically require many rules to represent this information.

Comparing Reduced Cost Ordering to Reduced Error Ordering clearly illustrates the tradeoff between minimizing error and minimizing the cost of errors. On each database, Reduced Cost Ordering results in significantly lower misclassification costs than Reduced Error Ordering, and it also has significantly lower accuracy. It achieves lower misclassification costs by allowing more inexpensive errors to occur in expectation that fewer expensive errors will occur.

4 OVERFITTING AVOIDANCE

FOCL attempts to learn clauses that cover every positive training example. This strategy might not be the best strategy on noisy data sets such as the telephone troubleshooting data because it tends to learn a large number of very specific rules. Here we present a new rule pruning algorithm, designed to be used in combination with the Reduced Cost Ordering algorithm. First, FOCL is run as before learning a set of clauses for each class. For each clause that is learned, all *prefixes* of that clause are found and added to the pool of clauses from which the reduced cost ordering algorithm selects clauses. A clause prefix is an initial sequence of literals. The clause `(ca < 1.0 ∧ thalach ≥ 160 ∧ oldpeak < 1.0)`, has two prefixes: `(ca < 1.0)` and `(ca < 1.0 ∧ thalach ≥ 160)`. The Reduced Cost Ordering algorithm does not require any changes to select from a pool of clauses that includes all of the prefixes of the learned clauses.

We decided upon using clause prefixes for a variety of reasons that are likely to be true of any separate-and-conquer inductive rule learner. Due to the way in which clauses are constructed (by favoring literals that cover many positive and few negative examples), the first literals in a clause cover many positive examples, while the last literals exclude the remaining negative examples including perhaps those positive examples which due to classification noise are labeled as negative examples. If there were no noise in the data, the last literals might not have been added to the clause. It is these last literals that are not included in some prefixes of a clause. A variety of pruning algorithms, (e.g., Pagallo & Haussler, 1987; Pazzani & Brunk, 1991; Ali & Pazzani, 1992 Cohen, 1993) take advantage of this property. A second consideration was efficiency. The number of rule prefixes is, of course, linear in the number of literals in the clause, while some alternatives, such as all subsets of the clause are exponential. In the telephone troubleshooting problem, it is not uncommon to have 12 literals in a clause. Adding all prefixes of the learned clauses increases the complexity of creating a decision list to $O(R^2L^2E)$ where L is the number of literals per clause (which has an upper bound of E). Finally, deleting literals only from the end of clauses is a sufficient (but not necessary) condition to insure that no literal that introduces a new variable is deleted unless all literals that use that variable are also deleted.

An interesting property of the RCO with Clause Prefix algorithm is that it can include multiple prefixes of the same initial clause. This can occur if a more general clause that makes many mistakes at the beginning of the decision list makes few mistakes at the end of a decision list. Note that the RCO algorithm will never add a more specific clause to the end of a decision list containing a more general clause, since a more specific clause will not cover any remaining examples.

Cohen (1993) presents a rule pruning algorithm related to the Clause Prefix strategy. There are a few important differences however. Cohen's algorithm prunes each class independently, while ours prunes all classes simultaneously as it builds a decision list. Cohen's algorithm is intended to be used with a classification procedure that resolves conflicts that arise when clauses from different classes are satisfied by selecting the clause that covers the most positive examples of that class. Furthermore, our algorithm can use different prefixes of the same rule in a decision list, while Cohen's does not.

Table 2 illustrates the result of using the Clause Prefix strategy with the Reduced Cost Ordering algorithm. It significantly ($p < .05$) reduced the cost of mistakes made by learned rules on the telephone trouble shooting and the diabetes databases when compared to RCO without the Clause Prefix strategy. On the other databases, using the Clause Prefix strategy had no significant effect.

Since decision trees are a common representation for machine learning programs, we also considered adapting reduced error pruning (REP– Quinlan, 1987), to reduce the cost of misclassification errors. We call the new algorithm reduced cost pruning (RCP). In reduced cost pruning, the training data is divided into a learning set and a pruning set. A decision tree is created on the learning set and is pruned using the pruning set as follows. For each leaf of the decision tree, the examples from the pruning set that are sorted to that leaf are found, and the leaf is given the class label of the least expected cost class of those examples. Next, RCP prunes the tree by considering turning each test node into a leaf node. A test node is converted to a leaf node if the cost of the least expected cost class of all pruning examples sorted to the test node is less than or equal to the sum of the cost of using the subtree under that test node to classify the pruning examples. When a test node is converted to a leaf node, it is given the class label of the least expected cost class of the pruning examples at that node. Table 2 contains the results of RCP and REP on the same databases using information gain to select tests. Although RCP significantly ($p < .05$) reduced the cost of errors on the telephone troubleshooting and diabetes databases, it significantly increases the cost of errors on the mushroom database. One potential reason for the increased error is that RCP (and REP) require the training data to be further subdivided into a learning and pruning set, possibly reducing the accuracy of the tree learned. In the experiments reported here with pruning, 30% of the training data is used for pruning and 70% of the training data is used for learning.

We also considered a variant of RCP that first creates and prunes the trees exactly like RCP. However, it then uses the entire training set (i.e., both the learning set and the pruning set) to find the least expected cost class of a leaf. The intent of this variant is to provide more data from which to assign the least cost class. This algorithm, called "I-gain+RCP (Full Training)" in Table 2, significantly ($p < .05$) reduces misclassification costs when compared to RCP on every database.

5 KNOWLEDGE INTENSIVE METHODS

In this section, we consider a special purpose system for revising the numeric parameters of MAX, and applying FOCL, a general purpose knowledge intensive learner to this same problem.

5.1 ITERATIVE REFINEMENT OF NUMERIC PARAMETERS

We have designed a special purpose system for revising the numeric parameters of MAX, the expert system used by NYNEX to dispatch a repair person. Recall that MAX contains a set of numeric parameters used to customize it to specific sites. The parameter revision system for MAX changes the values of these numeric parameters to reduce the misclassification costs on a set of training data. It tries adding or subtracting a small amount from each parameter (2.5% of its value) and as soon as it finds a parameter for which a change reduces cost, it makes that change. In addition, if a change has no effect on cost, it is made with 0.5 probability. If no parameter change results in an improvement, (i.e., a local minimum or plateau in parameter space is reached), it tries making larger changes to the parameter values (incrementing the range by 2.5% each cycle), giving up after attempting to change parameter values by up to 50% of their value. In addition to the greedy approach described here, we have also tried steepest descent (making only the change that has the most effect on cost in each cycle) and simulated annealing approaches to changing parameters. Neither of these two methods produce better results, suggesting that local minima, perhaps caused by interactions between parameters, are not a problem in this search space. We are currently experimenting with genetic algorithms on this task.

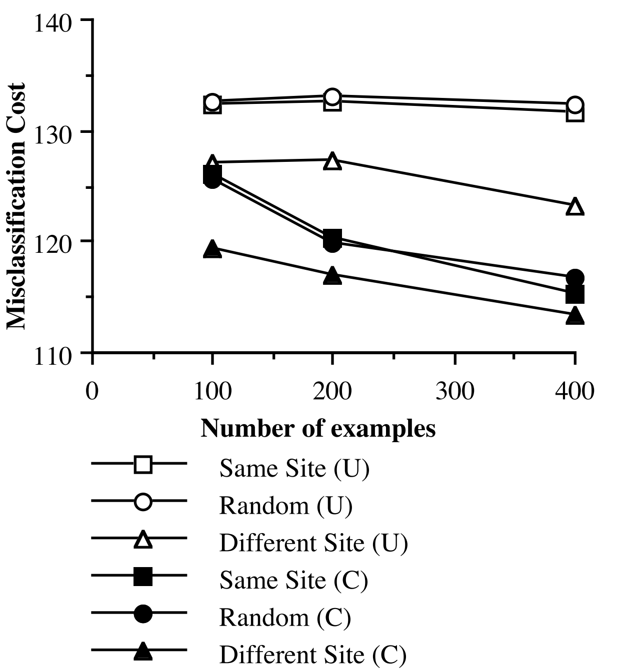

Figure 1. Cost after revising the numeric parameters of MAX optimizing with respect to the cost matrix from Table 1 (C) with solid plot symbols and a uniform cost

We ran experiments in which we started with 3 different sets of initial parameters and trained and tested on data from a single site. The parameters used were:

1. The actual parameters used by MAX in the same site from which the training and test data was collected. This tests the ability of theory revision system to fine tune MAX in a simulated operational setting. We'll call this condition the Same Site setting. The initial accuracy of MAX was .324 and the initial cost of MAX is 134.2 in this condition. We have access to only a subset of the MAX knowledge-base and the MAX training and test data that is intended to be processed by this subset. The actual MAX accuracy is considerably higher than that these figures indicate (Rabinowitz, et al., 1991).
2. Random values were chosen for each parameter from a uniform distribution in the range of the actual value of parameter minus 25% of its value and the actual value plus 25% of its actual value. This tests the ability to tune the system starting with "reasonable" but erroneous parameter settings. We'll call this condition the Random setting. The initial accuracy of MAX was .319 and the cost of MAX was 134.9 in this condition.
3. The actual parameters used by MAX at a different site from which the data is collected. We'll call this condition the Different Site setting. This tests the ability to customize MAX to a new site, starting with the parameters of a different site. The initial accuracy of MAX was .322 and the initial cost of MAX was 130.6 in this condition.

Figure 1 compares the misclassification cost of the algorithm that changes parameter values to minimize the cost of errors to a related algorithm that attempts to reduce the number of errors (by using a uniform cost matrix). The values plotted are the means of 20 runs computed using a distinct test set of 554 examples.

For each set of parameters, and each number of training examples, we performed a t-test comparing the misclassification cost when minimizing cost to the cost of misclassification when minimizing error. In all cases, the minimizing costs had significantly lower costs than minimizing error ($p < .0001$). In data not graphed here, the accuracy of the system also significantly increased from the initial accuracy in all conditions.

5.2 COMBINING EXPLANATION-BASED AND INDUCTIVE LEARNING

FOCL (Pazzani & Kibler, 1992) is a general purpose relational learning algorithm that combines inductive and explanation-based learning. The explanation-based part of FOCL can use the knowledge-base of an expert system to create rules by operationalizing (Mitchell, Keller & Kedar-Cabelli, 1986) explanations of examples. The operationalizations found by explanation-based learning are further generalized by greedily deleting conditions from them if doing so improves the rule. The inductive component can specialize an operationalization if it also explains some negative examples. The inductive component can also create new rules to account for positive examples that are not correctly classified by the knowledge-base. The decisions of whether to use EBL, generalize operationalizations, specialize operationalizations, or use inductive learning are controlled by an information gain metric (Quinlan, 1990) to judge the quality of hypotheses.

Unlike the special purpose parameter revision system discussed in the previous section, FOCL does not contain any operator for changing numeric constants. Instead changing a constant occurs as the result of learning a rule via operationalization such as `Voltage > 17.5 ∧ Resistance > 2400`, deleting a condition to produce `Voltage > 17.5` and adding a condition with induction to create `Voltage > 17.5 ∧ Resistance > 2500`. Of course, nothing prevents FOCL from adding a condition involving a different feature, such as `Voltage > 17.5 ∧ Type-of-service = PBX`. This additional degree of freedom, together with the ability to learn entirely new rules gives FOCL a different sort of bias. FOCL is not as constrained by the existing knowledge-base but only uses those parts of it that are useful as determined by the fact that they have more information gain than rules learned via induction alone.

We ran an experiment is which we compared FOCL with a domain theory to FOCL without a domain theory at minimizing the cost of misclassifying examples. In both conditions, the Reduced Cost Ordering algorithm was used in an attempt to minimize misclassification costs. The partial MAX knowledge-base was used as a domain theory with parameters from the same site as the training and test data. We ran 20 trials under each condition using training sets of 100, 200 and 400 examples and testing on 300 examples. Figure 2 graphs the cost of misclassifying examples under these conditions. Paired t-tests indicate that using the domain theory significantly decreases the cost of misclassifying the examples ($p < .05$) with 200 and 400 training examples. Using the domain theory only significantly improved the accuracy of the learned rules with 200 examples in this experiment. The cost results are substantially better than that of iterative refinement approach to numeric parameter testing. This result indicates that the partial MAX knowledge base can be improved by adding and deleting rules as well as by revising the existing numeric parameters.

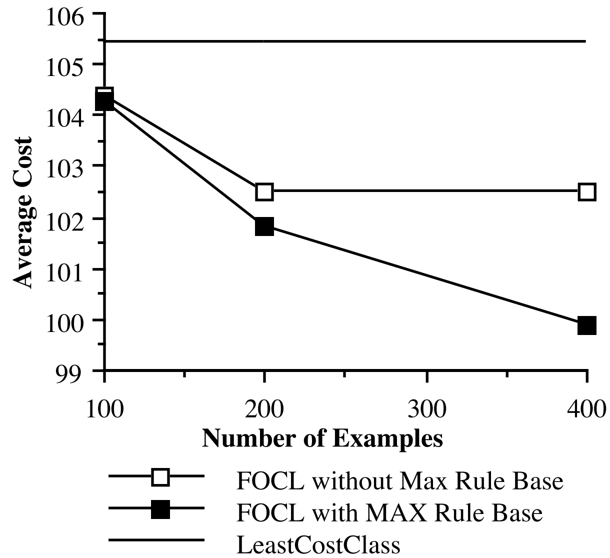

Figure 2. The cost of misclassifying examples in

We have also ran experiments in which the partial NYNEX MAX domain theory was used together with the Clause Prefix overfitting avoidance strategy. In these experiments, there was no significant difference between using the overfitting avoidance strategy and not using it.

6 CONCLUSIONS

Some of the modifications we tried on decision tree and decision list learners with the intention of reducing the costs of misclassification errors actually had the effect of increasing the cost. Before drawing any conclusions from the data we have presented so far, we'd also like to mention a few modifications that did not reduce costs in experiments not reported here:

- Ordering rules by an estimate of the cost and benefit of the rule. It's clear that the cost of the errors made by a rule can be estimated by using a set of examples that are distinct from the training data used to learn the rule. Although we'd also like to favor rules that correctly classify many examples, it's not clear how to trade off these factors in a principled way. One approach that did not work was to order the rules by the *benefit–cost*, where *benefit* is defined as the number of examples correctly classified multiplied by some constant representing the importance of correctly classifying examples of that class. To determine the importance of a class, we estimate the cost of assigning an example of that class to each class with a probability equal to the relative frequency of examples of that class:

$$importance(j) = \sum_i Cost(i,j)\frac{N_i}{N}$$

where j is the class of a rule, there are N total examples and N_i examples of class i.

- Ordering rules by a weighted Laplace estimate of the rules' accuracy on the training data. As with the previous approach the weight for the examples correctly classified by the rule is given by the importance of the class of the rule, while the weight for examples incorrectly matched is given by the cost of mistaking examples whose true class is i for examples of class j:

$$\frac{importance(n_j+1)}{importance(n_j+1) + \sum_{i \neq j} \left(n_i+1\right) Cost(i,j)}$$

where the rule for class j matches the n_i examples of class i. Using a Laplace estimate of accuracy (without the weights) performs as well at minimizing error as using Reduced Error Ordering. However, adding the weights to reflect the cost of making errors and the importance of covering examples did not achieve the desired goals.

Comparing the cases in which we were successful in reducing the costs of misclassification errors (the Reduced Cost Ordering algorithm and the iterative improvement parameter adjusting algorithm) to the cases that we have shown experimentally not to reduce the cost of learning, we find that the successful approaches have one feature in common: they estimate the global cost of the current hypothesis, and then estimate the global cost of possible revision or extension to that hypothesis. In contrast, those approaches that did not reduce costs attempt to evaluate the desirability of a local decision, without considering how that decision might interact with future decisions that must be made. In the case of the two alternative decision list ordering algorithms discussed in this section, the cost of a single rule is estimated without taking into account how examples not matched by that rule might be classified. In the case of the decision tree metrics, particularly using misclassification costs directly as a test selection heuristic, the true misclassification cost of a test will depend upon the additional tests that are created as subtrees to that node.

Acknowledgements

The research reported here was supported in part by NYNEX, NSF grant IRI-9310413, ARPA grant F49620-92-J-0430, and AFOSR AASERT grant F49620-93-1-0569. We'd like to thank Dennis Kibler and the other members of the machine learning group at UCI who have commented on this paper. In addition, comments by Andrea Danyluk, Thomas Fawcett, and Foster Provost are gratefully acknowledged. Finally, we'd like to thank Pat Langley for providing the code for the Bayesian classifier.

References

Ali, K. & Pazzani ,M. (1992). *Reducing the small disjuncts problem by learning probabilistic concept descriptions*. CLNL.

Berger, J. (1980) *Statistical Decision Theory*. Springer-Verlag: NY

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

Clark, P. & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3, 261-284.

Cohen, W. (1993). Efficient pruning methods for separate-and-conquer rule learning systems, IJCAI pp 988-994.

Danyluk, A. & Provost, F. (1993). *Small disjuncts in action: Learning to diagnose errors in the telephone network local loop*. Machine Learning Conference, pp 81-88.

Duda, R. & Hart, P. (1973). *Pattern classification and scene analysis*. New York: John Wiley & Sons.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based learning: A unifying view. *Machine Learning, 1,* 47–80.

Miyauchi, R. & Rendell, L. (1991). *Inductive learning with COBLER*. University of Illinois Tech Report.

Pagallo, G., & Haussler, D. (1990). Boolean Feature Discovery in Empirical Learning. *Machine Learning, 5,* 71–99.

Pazzani, M. & Brunk, C. (1993). Finding accurate frontiers: A knowledge-intensive approach to relational learning. *The National Conference on Artificial Intelligence* (pp. 328-334). Washington, D.C: AAAI Press.

Pazzani, M., & Kibler, D. (1992). The role of prior knowledge in inductive learning. *Machine Learning, 9*, 54-97.

Provost, F & Buchanan, B. (1991). *Inductive Policy*. AAAI pp. 255-261.

Provost, F. (1994). *Goal-directed inductive learning: Trading off accuracy for reduced error cost*. AAAI Spring Symposium on Goal-Drive Learning.

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning, 1,* 81–106.

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

Quinlan, J.R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Quinlan, J.R. (1992). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Rabinowitz, H., Flamholz, J., Wolin, E.,& Euchner, J. (1991). Nynex MAX: A telephone trouble screening expert system. In R. Smith & C. Scott (Ed.) *Innovative applications of artificial intelligence*, 3, 213–230.

Tcheng, D., Lambert, B., Lu, S. & Rendell, L. (1989). *Building robust learning systems by combining induction and optimization*. IJCAI pp 806-812.