AN ABSTRACT OF THE THESIS OF

Daniel L. K. Forrest for the degree of Master of Science in Computer Science presented on November 30, 2001.

Title:

Shrunken Learning Rates Do Not Improve AdaBoost On Benchmark Datasets.

Redacted for Privacy

Abstract approved: _____

Thomas G. Dietterich

Recent work has shown that AdaBoost can be viewed as an algorithm that maximizes the margin on the training data via functional gradient descent. Under this interpretation, the weight computed by AdaBoost, for each hypothesis generated, can be viewed as a step size parameter in a gradient descent search. Friedman has suggested that shrinking these step sizes could produce improved generalization and reduce overfitting. In a series of experiments, he showed that very small step sizes did indeed reduce overfitting and improve generalization for three variants of Gradient_Boost, his generic functional gradient descent algorithm. For this report, we tested whether reduced learning rates can also improve generalization in AdaBoost. We tested AdaBoost (applied to C4.5 decision trees) with reduced learning rates on 28 benchmark datasets. The results show that reduced learning rates provide no statistically significant improvement on these datasets. We conclude that reduced learning rates cannot be recommended for use with boosted decision trees on datasets similar to these benchmark datasets.

Shrunken Learning Rates Do Not Improve AdaBoost On Benchmark Datasets

by
Daniel L. K. Forrest

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented November 30, 2001
Commencement June 2002

Master of Science thesis of Daniel L. K. Forrest presented on November 30, 2001

APPROVED:

Redacted for Privacy

Major Professor, representing Computer Science

Redacted for Privacy

Chair of the Department of Computer Science

Redacted for Privacy

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Daniel L. K. Forrest, Author

ACKNOWLEDGMENT

First, I give my thanks to the National Science Foundation for their support through grants IRI-9626584 and IIS-0085836 as well as the Air Force Office of Scientific Research for their support through grant F49620-98-1-0375. Second, I thank my major professor, Dr. Thomas Dietterich, for his countless hours of advice on all of my many research projects. Third, I would like to thank those who served on my masters committee: Dr. Prasad Tadepalli, Dr. Curtis Cook and Dr. Wen-Tsong Shiue. Finally, I thank my wife, Lori. Without her support, this adventure would not have been possible.

TABLE OF CONTENTS

Page

1 INTRODUCTION 1

2 GENERALIZING GRADIENT DESCENT 3

2.1 Standard Gradient Descent 3

2.2 Functional Gradient Descent 3

2.3 Gradient_Boost 5

2.4 Regularization 6

3 ADABOOST 8

4 METHODS 11

5 RESULTS 13

6 DISCUSSION 17

7 CONCLUSIONS AND FUTURE WORK 19

BIBLIOGRAPHY 20

LIST OF FIGURES

Figure Page

5.1 Plot of the difference in the average number of errors for C4.5 AdaBoost with a learning rate of 0.01 for 2000 steps with early stopping and C4.5 AdaBoost with a learning rate of 1.0. 14

5.2 Plot of the difference in the average number of errors for C4.5 AdaBoost with a learning rate of 0.08 for 2000 steps with early stopping and C4.5 AdaBoost with a learning rate of 1.0. 15

LIST OF TABLES

Table | Page

2.1 The Gradient_Boost algorithm 6

3.1 The AdaBoost algorithm. 9

5.1 The relationship of the chosen stopping point to the value of the shrinkage parameter ν. 16

SHRUNKEN LEARNING RATES DO NOT IMPROVE ADABOOST ON BENCHMARK DATASETS

1. INTRODUCTION

Supervised learning algorithms are methods for constructing a predictive classifier based on a limited set of examples. More formally, given a set of inputs in the form $(\mathbf{x}_i, y_i)$, called the training data, from an unknown distribution $P(\mathbf{x}, y)$, we want to find a function F, where $F(\mathbf{x}) = y$, that minimizes the expected loss with respect to some loss function over as yet unseen $(\mathbf{x}, y)$ pairs. The function F can be thought of as predicting a y value when given a particular feature vector $\mathbf{x}$. There are two types of problems that fit into this framework depending on the values of y. If $y \in \Re$, the problem is called regression. If the set of values for y is a distinct set of numbers, such as $y \in \{-1, +1\}$, the problem is called classification.

One popular method of supervised learning is gradient descent. With gradient descent, the function F is controlled by a set of parameters Θ. The goal is to find the set of parameters that minimizes the expected loss. This is done by incrementally generating a parameter set θ_i that when added to Θ will reduce the error of F on the training data.

Friedman generalizes the gradient descent algorithm by applying it to functions rather than parameters. That is, instead of modifying the parameters of a function to minimize the expected error, he suggests we modify the functions them-

selves. This amounts to iteratively generating functions and combining them to minimize expected error. This approach is referred to as functional gradient descent.

One well-known functional gradient descent algorithm is AdaBoost. With AdaBoost, the incremental functions are generated via any supervised learning algorithm. The final classifier is simply a weighted vote of the learned functions.

One problem with all supervised learning algorithms is that they tend to lose their generalization accuracy eventually if they are run too long, a phenomenon known as overfitting. Friedman suggests that for any functional gradient descent algorithm, shrinking the learning rate will counter this phenomenon. He conducted several experiments using three functional gradient descent algorithms on synthetic data to support his position.

In this paper, we describe an experimental study to test whether reduced learning rates can improve the generalization accuracy of AdaBoost applied to classification problems. Our study concludes that reduced learning rates do not provide any statistically significant gains and, in several cases, produced worse performance than standard AdaBoost.

2. GENERALIZING GRADIENT DESCENT

2.1. Standard Gradient Descent

To understand Friedman's approach to functional gradient descent, we need to first understand standard gradient descent. With standard gradient descent, the goal is to find the set of parameters, θ^*, that minimizes some parameterized objective function $J(\theta)$. That is, we want to find

$$\theta^* = \underset{\theta}{\operatorname{argmin}} J(\theta). \tag{2.1}$$

To do this via gradient descent, we incrementally adjust the parameter set in the direction of the negative gradient. Let

$$\Theta_m = \sum_{\ell=0}^{m} \theta_\ell \tag{2.2}$$

be the parameter set on iteration m, where θ_0 is the initial parameter set. The goal on iteration m is then to chose $\theta_m = -\rho_m \mathbf{g}_m$ to minimize $J(\Theta_{m-1} + \theta_m)$. Usually,

$$\mathbf{g}_m = \left.\frac{\partial J(\theta)}{\partial \theta}\right|_{\theta=\Theta_{m-1}}. \tag{2.3}$$

However, $\mathbf{g}_m$ need only be in a direction that reduces the objective function. ρ_m is the stepsize and is given by the line search

$$\rho_m = \underset{\rho}{\operatorname{argmin}} J(\Theta_{m-1} - \rho \mathbf{g}_m). \tag{2.4}$$

2.2. Functional Gradient Descent

Friedman extends gradient descent by using a classification function $F(\mathbf{x})$ evaluated at each point $\mathbf{x}$ as the parameter set. That is,

$\theta = (F(\mathbf{x}_1), F(\mathbf{x}_2), ..., F(\mathbf{x}_N))$. The objective function is some loss function L. The goal being to find

$$F^* = \underset{F}{\operatorname{argmin}} \sum_{i=0}^{N} L(F(\mathbf{x}_i), y_i). \tag{2.5}$$

As with standard gradient descent, we will use an incremental approach to find F^* such that on the m^{th} iteration

$$F_m(\mathbf{x}_i) = \sum_{\ell=0}^{m} f_\ell(\mathbf{x}_i), \tag{2.6}$$

where f_0 is our initial classifier and $f_m = -\rho_m \mathbf{g}_m$ is the increment at step m. In this case,

$$\mathbf{g}_m(\mathbf{x}) = \left. \frac{\partial L(F(\mathbf{x}), y)}{\partial F(\mathbf{x})} \right|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{2.7}$$

and

$$\rho_m = \underset{\rho}{\operatorname{argmin}} \sum_{i=0}^{N} L(F_{m-1}(\mathbf{x}_i) - \rho \mathbf{g}_m(\mathbf{x}_i), y_i). \tag{2.8}$$

It should be noted that we can only calculate $\mathbf{g}_m$ at the training data points. This prevents us from directly adding it to F. To generalize F, to allow new data points, we must approximate $\mathbf{g}_m$ in some manner. We do this by using a parameterized function $h(\mathbf{x}; \mathbf{a})$[1] and calculate it with

$$\mathbf{a}_m = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{i=1}^{N} [-\mathbf{g}_m(\mathbf{x}_i) - h(\mathbf{x}; \mathbf{a})]^2, \tag{2.9}$$

which minimizes the difference between h and $-\mathbf{g}_m$. Thus, $-\mathbf{g}_m$ becomes the target value for h, and we can apply a standard machine learning method to find $\mathbf{a}_m$. In

---

[1]The notation $h(\mathbf{x}; \mathbf{a})$ refers to a function h with input vector $\mathbf{x}$ and parameterized by the vector $\mathbf{a}$.

this formulation, $-\mathbf{g}_m$ is simply a pseudo target value and can be represented as such using the standard notation $\tilde{y}$. With this in mind, let

$$\tilde{y}_i = -\mathbf{g}_m(\mathbf{x}_i). \tag{2.10}$$

We can then rewrite the equation for $\mathbf{a}_m$ as

$$\mathbf{a}_m = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{i=1}^{N} [\tilde{y}_i - h(\mathbf{x}; \mathbf{a})]^2 \tag{2.11}$$

and the equation for ρ_m as

$$\rho_m = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} L(F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}; \mathbf{a}_m), y_i) \tag{2.12}$$

2.3. Gradient_Boost

Using this approach to gradient descent, Friedman proposes a new generalized learning algorithm shown in Table 2.1 called Gradient_Boost. Note that in Gradient_Boost, the initial classifier is set to the constant value with minimal difference from the y_i's. For classification problems where $y \in \{-1, +1\}$, this is simply the value of y_i that occurs most frequently. For regression, this is the average value of the y_i's.

In his experiments, Friedman created three variants of Gradient_Boost: L_2TreeBoost, LS_Boost, and LAD_TreeBoost. For each of these, he used 11-terminal node regression trees to calculate the function h. The difference between the three algorithms is in their loss functions. For L_2TreeBoost, he used the negative binomial log likelihood,

$$L(F(\mathbf{x}), y) = log(1 + e^{-2yF(\mathbf{x})}). \tag{2.13}$$

Algorithm 1: Gradient_Boost

1 $F_0(\mathbf{x}) = \text{argmin}_{\hat{y}} \sum_{i=1}^{N} L(\hat{y}, y_i)$

2 For $m = 1$ to M do:

3 $\tilde{y}_i = -\mathbf{g}_m(\mathbf{x}_i) = -\left[\frac{\partial L(F(\mathbf{x}_i), y_i)}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x}_i)=F_{m-1}(\mathbf{x}_i)}, i = 1, 2, ..., N$

4 $\mathbf{a}_m = \text{argmin}_{\mathbf{a}} \sum_{i=1}^{N} [\tilde{y}_i - h(\mathbf{x}; \mathbf{a})]^2$

5 $\rho_m = \text{argmin}_{\rho} \sum_{i=1}^{N} L(F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m), y_i)$

6 $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$

7 end For

TABLE 2.1. The Gradient_Boost algorithm

For LS_Boost, the loss function is $\frac{1}{2}(y - F(\mathbf{x}))^2$, and for LAD_TreeBoost, the loss function is $|y - F(\mathbf{x})|$. Note that Friedman uses L_2TreeBoost for classification and the other two methods for regression.

2.4. Regularization

In all learning problems, we want to find a function that not only minimizes error on the training set but is also able to generalize to data not in the training set. If we find a function that fits the training data too closely, we risk losing the ability to generalize to new data. This phenomenon is commonly referred to as overfitting.

One way of minimizing overfitting is to use a regularization method. Several such methods exist for gradient descent algorithms including limiting the number of gradient updates and shrinking the impact of new components (the h's). Friedman proposes that, for algorithms such as Gradient_Boost, regularization by shrinkage

is the most appropriate method. This amounts to modifying line 6 of the Gradient_Boost algorithm to

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu\rho_m h(\mathbf{x}; \mathbf{a}_m), 0 < \nu \leq 1. \tag{2.14}$$

The variable ν can be viewed as a learning rate and has the effect of shrinking each update to F.

Friedman experimented with using shrinkage values of $\nu \in \{1.0, 0.5, 0.25, 0.125, 0.06, 0.03\}$ for the three algorithms mentioned above, each of which is a derivative of Gradient_Boost. For each of the three algorithms, he showed that choosing $\nu \in \{0.125, 0.06, 0.03\}$ yielded the best generalization.

3. ADABOOST

As mentioned before, we chose to use AdaBoost as our implementation of a functional gradient descent learning algorithm. The AdaBoost algorithm [7] is shown in Table 3.1. The input to AdaBoost is the set of training examples, a learning algorithm (to calculate $h(\mathbf{x};\mathbf{a})$), and the number of iterations to run the algorithm. The learning algorithm must be able to accept a set of weights, $\mathbf{w}$, that tell for each example, $(\mathbf{x}_i, y_i)$, how important it is to classify that example correctly. On iteration m, the learning algorithm seeks to find the hypothesis h_m that minimizes the weighted 0/1 loss

$$\mathbf{a}_m = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{i}^{N} w_i^m L_{0/1}(h_m(\mathbf{x}_i;\mathbf{a}), y_i), \tag{3.1}$$

where $L_{0/1}$ is defined as

$$L_{0/1}(y_1, y_2) = \begin{cases} 0 \text{ if } y_1 = y_2 \\ 1 \text{ if } y_1 \neq y_2 \end{cases}. \tag{3.2}$$

Initially, the weights are calculated based on the current classifier[1] and passed to WeightedLearn. After WeightedLearn returns h_m, AdaBoost computes a stepsize, ρ_m, from the weighted 0/1 loss, updates $F_m = F_{m-1} + \rho_m h_m$, and repeats until $m = M$.

Several authors [10, 13, 5, 11] have shown that the AdaBoost algorithm for ensemble learning can be viewed as performing a functional gradient descent search to minimize the negative of the margin of the training examples. The *margin* of prediction $F(\mathbf{x}_i)$ is defined to be $y_i F(\mathbf{x}_i)$, which is positive if the prediction is

---

[1]Note that the initial classifier F_0 is always 0. This forces the initial set of weights to all be 1, since the exponent to e is 0.

Algorithm 2: AdaBoost

Input: sequence of N labeled examples $\langle(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\rangle$

learning algorithm WeightedLearn

integer M specifying number of iterations

1 Let $F_0(\mathbf{x}) = 0$

2 For $m = 1$ to M do:

3 $w_i^m = e^{-y_i F_{m-1}(\mathbf{x}_i)} \;\; i = 1...N$

4 Call WeightedLearn with weights $\mathbf{w}^m$; get back hypothesis $h_m(\mathbf{x}; \mathbf{a}_m)$

5 Calculate the error of h_m: $\epsilon_m = \frac{\sum_{i=1}^N w_i^m L_{0/1}(h_m(\mathbf{x}_i; \mathbf{a}_m), y_i)}{\sum_{i=1}^N w_i^m}$

6 $\rho_m = log\frac{1-\epsilon_m}{\epsilon_m}$

7 $F_m = F_{m-1} + \rho_m h_m$

8 end For

Output the hypothesis $F^*(\mathbf{x}) = \begin{cases} 1 & \text{if } F_M(\mathbf{x}) \geq 0 \\ -1 & \text{otherwise} \end{cases}$.

TABLE 3.1. The AdaBoost algorithm.

correct and negative if it is incorrect. AdaBoost can be viewed as attempting to find a classifier F that minimizes the loss function

$$L(F(\mathbf{x}), y) = e^{-yF(\mathbf{x})}. \tag{3.3}$$

This loss function can be seen as a transformed version of the margins of the training examples. Therefore, minimizing equation (3.3) will also maximize the margin for a given training example.

With this in mind, we can derive AdaBoost from Friedman's functional gradient descent perspective. Using the loss function given in equation(3.3), the gradient becomes

$$\mathbf{g}_m(\mathbf{x}) = \left.\frac{\partial L(F(\mathbf{x}), y)}{\partial F(\mathbf{x})}\right|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = \left.\frac{\partial e^{-yF(\mathbf{x})}}{\partial F(\mathbf{x})}\right|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = -ye^{-yF(\mathbf{x})}. \tag{3.4}$$

The next step is to find a hypothesis h_m that is close to $-\mathbf{g}_m$ for some measure of distance, D, between $-\mathbf{g}_m$ and h_m. That is, we want to find

$$\mathbf{a}_m = \underset{\mathbf{a}}{\operatorname{argmin}}\, D(-\mathbf{g}_m(\mathbf{x}), h_m(\mathbf{x}; \mathbf{a})) = \underset{\mathbf{a}}{\operatorname{argmin}} \sum_{i=1}^{N} D(y_i e^{-y_i F(\mathbf{x}_i)}, h_m(\mathbf{x}_i; \mathbf{a})). \tag{3.5}$$

AdaBoost employs the weighted 0/1 loss,

$$e^{-yF(\mathbf{x})} L_{0/1}(h(\mathbf{x}; \mathbf{a}), y) = D(ye^{-yF(\mathbf{x})}, h(\mathbf{x}; \mathbf{a})) \tag{3.6}$$

as its distance measure D. Because $h(\mathbf{x}; \mathbf{a}) \in \{-1, +1\}$, this distance is never zero, but reducing this distance brings h_m closer to $-\mathbf{g}_m$.

Instead of performing a line search to choose ρ_m, it has been shown [10] that the formula in line 6 of Table 3.1 can be derived from a quadratic approximation of the loss function $e^{-y(F(\mathbf{x})+\rho h(\mathbf{x};\mathbf{a}))}$.

4. METHODS

We implemented AdaBoost by modifying C4.5 [12] to implement AdaBoost.M1 [6] with boosting by weighting (as opposed to boosting by sampling). Our modified C4.5 includes all of the improvements to C4.5 through Release 7 as well as improved file handling (e.g., to read compressed data files) and other conveniences. When a hypothesis produces a weighted error greater than 0.5, we terminate the boosting iterations.

We employed 28 data sets from the Irvine repository [1]. The data sets cover a wide range of real world problems including medical diagnosis, chess move selection, chemical analysis, voting behavior prediction, and many others. We did not select data sets according to any particular rule, but instead simply chose those data sets that we had employed in an earlier study [4] with a few of the larger data sets omitted because of their computational expense. Specifically, the following data sets were employed: annealing, audiology, autos, breast-cancer-wisconsin, breast-cancer-yugoslovia, credit-australia, credit-german, glass, heart-cleveland, heart-hungarian, heart-switzerland, heart-va, hepatitis, horse-collic, hypo, iris, king-rook-vs-king-pawn, labor-negotiations, lymphography, primary-tumor, segment, sick, sonar, soybean, splice, vehicle, voting-records, and waveform. Performance was evaluated via 10-fold cross validation. To estimate the statistical significance of any observed differences in performance, we employed the 10-fold cross-validated t test [3] with a significance level of 0.05. This test tends to have an elevated probability of Type I error, so it sometimes finds differences between two classifiers when there is in fact no difference. Conversely, if it finds no difference between two classifiers, this result can be trusted.

There are two parameters that can affect the degree of overfitting: the value of ν and the number of iterations M. We computed results for 7 values of ν (0.01, 0.02, 0.04, 0.08, 0.16, 0.32 and 0.64). To determine the number of iterations M, we implemented a form of cross-validated early stopping as follows. Each training set (within a 10-fold cross-validation) is first divided into a sub-training set (80% of the examples) and a sub-validation set (20% of the examples). AdaBoost is executed on the sub-training set for $M = 2000$ iterations. After each iteration, the performance of F_M is measured on the sub-validation set, and the value of M giving the best performance (over the 2000 possible values) is chosen as the stopping point M_{halt}. AdaBoost is then applied to the entire training set for M_{halt} steps and the generalization error of $F_{M_{halt}}$ is measured on the test set.

This approach to choosing M_{halt} was adopted after a series of pilot experiments with the two-norm data set described in [2]. In these experiments, we evaluated an alternative approach in which the best F learned from the sub-training set was output as the final classifier, rather than recombining the sub-training and sub-validation sets and retraining F. For all of the learning rates that we tested, the combine-and-retrain method worked better.

5. RESULTS

Figure 5.1 compares the performance of AdaBoost/C4.5 with a learning rate $\nu = 0.01$ to standard AdaBoost/C4.5 using a "Kohavi diagram." The plot consists of 28 confidence intervals corresponding to the 28 domains. Each interval depicts the difference in the error rate of AdaBoost/C4.5 with $\nu = 0.01$ and the error rate of standard AdaBoost/C4.5 (i.e., with $\nu = 1.0$). If the confidence interval includes the zero axis, then the paired differences cross-validated t test has found no statistically significant difference between the two error rates. If the interval lies completely below the zero axis, then standard AdaBoost is superior to the $\nu = 0.01$ variant. If the interval lies entirely above the line, then standard AdaBoost is inferior to the $\nu = 0.01$ variant. To make the diagram easy to read, the domains are sorted by ascending values of the mean difference between the two error rates, and the mean values are connected by a line.

From the figure, we can see that there is no case where the $\nu = 0.01$ shrinkage version of AdaBoost outperformed standard AdaBoost. However, there were 15 cases in which the cross-validated t test concluded that the shrunken AdaBoost performed significantly worse than standard AdaBoost.

Figure 5.2 shows the corresponding plot for $\nu = 0.08$. A similar result is obtained here. As ν is increased, the two methods become more and more similar, and fewer statistically significant differences are found. Out of the 196 runs (7 learning rates; 28 domains), only 2 runs showed a statistically significant improvement in performance as a result of employing a shrinkage factor with early stopping. In contrast, 86 runs shows a statistically significant worsening of performance.

Figures 5.1 and 5.2 do not reveal the stopping points (M_{halt}) chosen by the sub-training/sub-validation process. This information is summarized in Table 5,

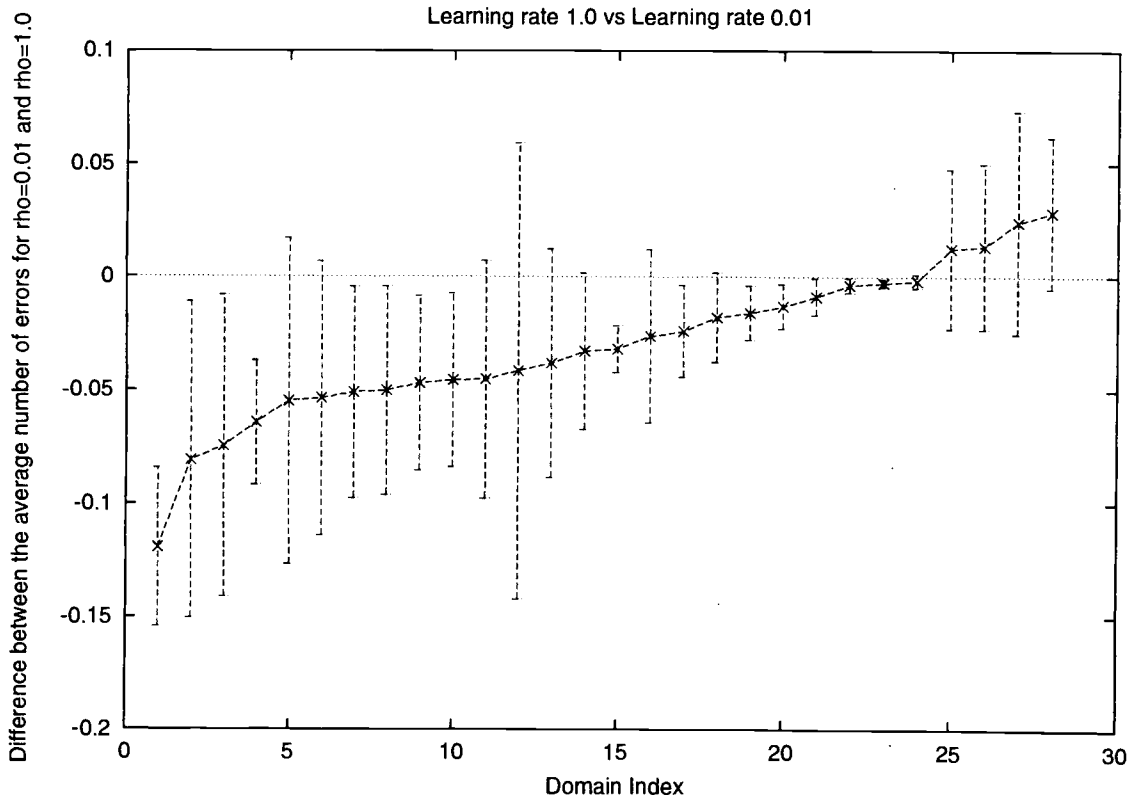

FIGURE 5.1. Plot of the difference in the average number of errors for C4.5 AdaBoost with a learning rate of 0.01 for 2000 steps with early stopping and C4.5 AdaBoost with a learning rate of 1.0.

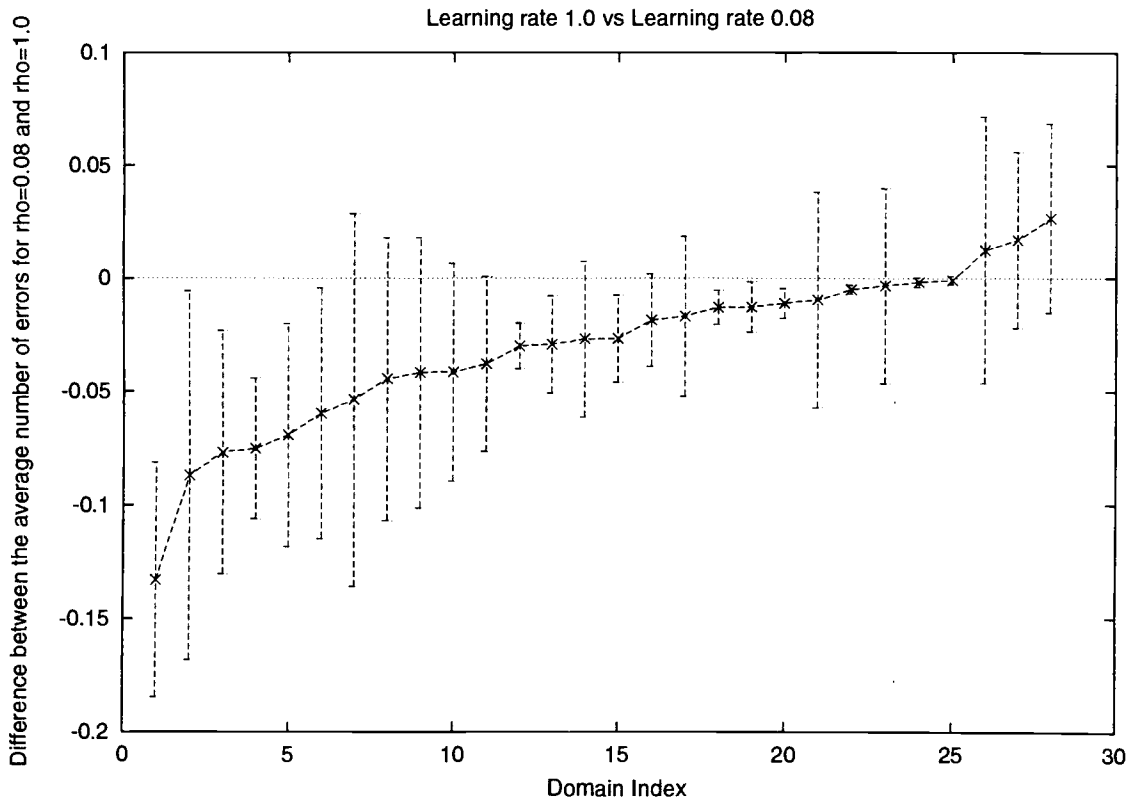

FIGURE 5.2. Plot of the difference in the average number of errors for C4.5 AdaBoost with a learning rate of 0.08 for 2000 steps with early stopping and C4.5 AdaBoost with a learning rate of 1.0.

ν	mean	median	min	max
0.01	129.50	19.5	1	1884
0.02	88.04	13	1	1994
0.04	65.67	8	1	1862
0.08	54.25	6.5	1	1562
0.16	43.58	5	1	1483
0.32	51.11	6	1	1062
0.64	82.04	9	1	1951
1.00	78.10	11.5	1	1769
overall	74.03	8	1	1994

TABLE 5.1. The relationship of the chosen stopping point to the value of the shrinkage parameter ν.

which shows the mean, median, minimum, and maximum values of M_{halt} for the various values of ν.

6. DISCUSSION

Our experiments are inconsistent with Friedman's experimental results on synthetic data. What are the possible explanations for this inconsistency?

First, Friedman's results were obtained with a synthetic data set having a very high noise rate (the signal-to-noise ratio was 1:1). It is unlikely that the Irvine data sets have such high levels of noise.

Second, Friedman's results were obtained for regression rather than classification. Many researchers, including Friedman [8], have argued that classification problems are fundamentally easier than regression problems, because the classifier only needs to determine whether $P(y_i = 1|\mathbf{x}_i) > 0.5$, whereas a regression estimator must predict the expected value of y_i given $\mathbf{x}_i$. Many researchers have noted experimentally that regression methods generally suffer more from overfitting and benefit more from regularization than classification methods.

On the other hand, there are good reasons to expect that overfitting would still be a problem with AdaBoost/C4.5, especially compared to Friedman's L_2_TREEBOOST procedure. Friedman employed regression trees having a small, fixed size (6 or 11 nodes), whereas the C4.5 trees are not being pruned (and trees are permitted to grow until each node contains only 2 training examples). Furthermore, Friedman found that using larger trees gave much worse results.

Friedman's experiments with small learning rates not only prevented overfitting but also led to improvements in the accuracy of the learned regression estimates. It was our hope that even in cases where AdaBoost/C4.5 does not overfit, it would still benefit from this finer-grained gradient descent search.

The experiments presented in this paper cannot distinguish among these possibilities. However, the fact that previous experiments have not found significant

evidence of overfitting with AdaBoost and C4.5 is consistent with the first hypothesis that these Irvine data sets do not have large amounts of noise.

7. CONCLUSIONS AND FUTURE WORK

Friedman [9] showed experimentally that employing a shrinkage factor to reduce the learning rate of functional gradient descent search gave improved results in some synthetic regression tasks. We tested whether this same strategy would improve the performance of AdaBoost (with C4.5) on 28 realistic data sets. Our experiments conclusively show that reducing the learning rate does not provide any improvement in performance and in many cases leads to a loss of accuracy. Based on these results, the application of shrinkage methods to AdaBoost/C4.5 cannot be recommended for classification problems.

Future work in this area primarily includes a more thorough analysis of why AdaBoost does not show any improvements when the learning rate is shrunk while Gradient_Boost variants do. This could include a more thorough examination of how the level of noise affects the performance of AdaBoost when shrinking the learning rate. As mentioned before, the data sets that Friedman used for his experiments contained a high amount of noise compared to the data sets in the Irvine Repository. This could be a reason for the differences between the Gradient_Boost variants and AdaBoost and could be checked. Another approach is to test if our use of decision trees was the source of difference between Friedman's results and our own. Implementing AdaBoost with regression trees and redoing the experiment could provide some useful information.

BIBLIOGRAPHY

[1] Blake, C. L. and Merz, C. J. UCI Repository of machine learning databases. [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA, University of California, Department of Information and Computer Science.

[2] Breiman L. (1996). Bias, variance, and arcing classifiers. Technical Report 460, Department of Statistics, University of California, Berkeley, CA.

[3] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7), pages 1895-1924.

[4] Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2), page 1.

[5] Duffy, N.and Helmbold, D. P. (1999). A geometric approach to leveraging weak learners. In *Proceedings of the 4th European Conference on Computational Learning Theory*, Vol. 1572 of *Lecture Notes in Artificial Intelligence*, pages 18-33. Springer-Verlag.

[6] Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148-146. Morgan Kaufmann.

[7] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119-139.

[8] Friedman, J. H. (1996). On bias, variance, 0/1 loss, and the curse of dimensionality. Technical Report, Department of Statistics, Stanford University, Stanford, CA.

[9] Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. Technical Report, Department of Statistics, Stanford University. IMS 1999 Reitz Lecture.

[10] Friedman, J. H., Hastie, T. and Tibshirani, R. (1998). Additive logistic regression: A statistical view of boosting. Technical Report, Department of Statistics, Stanford University.

[11] Mason, L., Baxter, J., Bartlett, P., and Frean, M. (2000). Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems*, Vol. 12, pages 512-518. Cambridge: MIT Press.

[12] Quinlan, J. R. (1993). *C4.5: Programs for Empirical Learning.* San Francisco: Morgan Kaufmann.

[13] Schapire, R. E. and Singer, Y. (1998). Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 80-91, New York: ACM Press.