Improving Regressors using Boosting Techniques

Harris Drucker
Monmouth University
West Long Branch, NJ 07764
drucker@monmouth.edu

Abstract

In the regression context, boosting and bagging are techniques to build a committee of regressors that may be superior to a single regressor. We use regression trees as fundamental building blocks in bagging committee machines and boosting committee machines. Performance is analyzed on three non-linear functions and the Boston housing database. In all cases, boosting is at least equivalent, and in most cases better than bagging in terms of prediction error.

1. INTRODUCTION

Both bagging [Breiman (1996a,1996b)] and boosting [Drucker *et. al.* (1994, 1996, 1993), Freund and Schapire (1996a,1996b), Schapire (1990)] are techniques to obtain smaller prediction errors (in regression) and lower error rates (in classification) using multiple predictors. Several studies of boosting and bagging in classification [Breiman (1996b), Freund and Schapire (1996a)] have shown the superiority of boosting over bagging but this is the first experimental study of combining regressors using boosting techniques. In both boosting and bagging, each regressor machine is trained on different subsets of the training set. In bagging, each machine is independently trained on N_1 samples picked *with replacement* from the N_1 original samples of the training set. Each machine is thereby trained on different (but overlapping) subsets of the original training set and will therefore give different predictions. Since each machine can be trained independently, the task of training each individual predictor may be assigned to different CPU's or a parallel processor. In bagging regressors, the ensemble prediction is the average of the predictions of all the machines.

In boosting, machines are trained sequentially. As in bagging, the first machine is trained on examples picked with replacement (of size N_1) from the original training set. We then pass *all* the training patterns through this first machine and note which ones are most in error. For regression machines, those patterns whose predicted values differ most from their observed values are defined to be "most" in error (this will be defined rigorously later). For those patterns most in error, their sampling probabilities are adjusted so that they are more likely to be picked as members of the training set for the second machine. Therefore, as we proceed in constructing machines, patterns that are difficult are more likely to appear in the training sets. Thus, different machines are better in different parts of the observation space. Regressors are combined using the weighted median, whereby those predictors that are more "confident" about their predictions are weighted more heavily. The details of this weighting scheme are discussed later.

Suppose we are given a set of *observations*, $(y_i, \boldsymbol{x}_i)$ $i=1,...,N_1$ where N_1 is the number of training set observations, and $\boldsymbol{x}$ is an M-variate vector. The probability density function of $(y,\boldsymbol{x})$ is fixed but unknown. Based on these observations, we form a predictor $y^{(p)}(\boldsymbol{x})$. We define a sample modeling error (ME) and prediction error (PE):

$$PE = \frac{1}{N_2}\sum_{i=1}^{N_2}[y_i - y_i^{(p)}(\boldsymbol{x}_i)]^2$$

$$ME = \frac{1}{N_2}\sum_{i=1}^{N_2}[y_i^{(t)} - y_i^{(p)}(\boldsymbol{x}_i)]^2$$

where $y_i^{(p)}(\boldsymbol{x}_i)$ is the prediction for the i'th test example, y_i is the i'th observation, and $y_i^{(t)}$ is the "truth". The parameters of $y^{(p)}(\boldsymbol{x})$ are obtained from the N_1 training set observations but the y_i and $\boldsymbol{x}_i$ in the above summations are obtained from a set of N_2 observations (the test set) never seen before. If the noise is additive, then $y_i = y_i^{(t)} + n_i$ where n_i is the i'th sample of the noise. Furthermore, if E[n] = 0 and $E[n_i n_j] = \delta_{ij}\sigma^2$, then we may take the expectation with respect to $(y,\boldsymbol{x})$ and obtain (Breiman and Spector, 1992):

$$E[PE] = \sigma^2 + E[ME]$$

This shows us that even if we know the model exactly, there is a minimum prediction error due to the noise. Our problem will be that the modeling error is also nonzero because we have to determine the model in the presence of noise. Since we don't know the probability distributions, we approximate the expectation of the ME and PE using the sample ME (if the truth is known) and sample PE and then average over multiple experiments.

In the following discussion, we detail both bagging and boosting. We then discuss how to build trees which are the basic building blocks of our regression machines and use these ensembles on some standard test functions.

2. BAGGING

The following is a paraphrase of Breiman (1996b) with some difference in notation. Suppose we pick with replacement N_1 examples from the training set of size N_1 and call the k'th set of observations O_k. Based on these observations, we form a predictor $y^{(p)}(\boldsymbol{x},O_k)$. Because we are sampling with replacement, we may have multiple observations or no observations of a particular training example. Sampling with replacement is sometimes termed bootstrap sampling [Efron and Tibshirani (1993)] and therefore this method is called **b**ootstrap **agg**regating or bagging for short. The ensemble predictor is formed from the approximation to the expectation over all the observation sets, i.e. $E_O[y^{(p)}(\boldsymbol{x},O)]$ by using the average of the outputs of all the predictors. Breiman discusses which algorithms are good candidates for predictors and concludes that the best predictors are unstable, i.e., a small change in the training set O_k causes a large change in the predictor $y^{(p)}(\boldsymbol{x},O_k)$. Good candidates are regression trees and neural nets.

3. BOOSTING

In bagging, each training example is equally likely to be picked. In boosting, the probability of a particular example being in the training set of a particular machine depends on the performance of the prior machines on that example. The following is a modification of *Adaboost.R* [Freund and Schapire (1996a)].

Initially, to each training pattern we assign a weight $w_i=1 \quad i=1,...,N_1$

Repeat the following while the average loss $\bar{L}$ defined below is less than .5 .

1. The probability that training sample i is in the training set is $p_i=w_i/\sum w_i$ where the summation is over all members of the training set. Pick N_1 samples (with replacement) to form the training set. This may be implemented by marking a line of length $\sum w_i$ and subsections of length w_i . A uniform number picked from the range $[0,\sum w_i]$ and landing in section i corresponds to picking pattern i.

2. Construct a regression machine t from that training set. Each machine makes a hypothesis: $h_t : x \to y$

3. Pass *every* member of the training set through this machine to obtain a prediction $y_i^{(p)}(\boldsymbol{x}_i) \quad i=1,...N_1$.

4. Calculate a loss for each training sample $L_i = L\left[\,|y_i^{(p)}(\boldsymbol{x}_i)-y_i|\,\right]$. The loss L may be of any functional form as long as $L \in [0,1]$. If we let

$$D=sup\,|y_i^{(p)}(\boldsymbol{x}_i)-y_i| \quad i=1,...,N_1$$

then we have three candidate loss functions:

$$L_i=\frac{|y_i^{(p)}(\boldsymbol{x}_i)-y_i|}{D} \qquad (linear)$$

$$L_i=\frac{|y_i^{(p)}(\boldsymbol{x}_i)-y_i|^2}{D^2} \qquad (square\ \ law)$$

$$L_i=1-exp\left[\frac{-|y_i^{(p)}(\boldsymbol{x}_i)-y_i|}{D}\right] \qquad (exponential)$$

5. Calculate an average loss: $\bar{L}=\sum_{i=1}^{N_1} L_i p_i$

6. Form $\beta=\frac{\bar{L}}{1-\bar{L}}$. β is a measure of confidence in the predictor. Low β means high confidence in the prediction.

7. Update the weights: $w_i \to w_i\beta^{**}[1-L_i]$, where ** indicates exponentiation. The smaller the loss, the more the weight is reduced making the probability smaller that this pattern will be picked as a member of the training set for the next machine in the ensemble.

8. For a particular input $\boldsymbol{x}_i$, each of the T machines makes a prediction h_t, $t=1,...,T$. Obtain the cumulative prediction h_f using the T predictors:

$$h_f = inf\left\{ y \in Y: \sum_{t:h_t \le y} \log(1/\beta_t) \ge \frac{1}{2}\sum_t \log(1/\beta_t) \right\}$$

This is the weighted median. Equivalently, each machine h_t has a prediction $y_i^{(t)}$ on the i'th pattern and an associated β_t. For pattern i the predictions are relabeled such that for pattern i we have:

$$y_i^{(1)}<y_i^{(2)}<,\ldots,<y_i^{(T)}$$

(retain the association of the β_t with its $y_i^{(t)}$). Then sum the $\log(1/\beta_t)$ until we reach the smallest t so that the inequality is satisfied. The prediction from that machine t we take as the ensemble prediction. If the β_t were all equal, this would be the median.

Intuitively, the effect of varying the weight w_i to give more emphasis to "difficult" examples means that each subsequent machine has a disproportionately harder set of examples to train on. Thus, the average loss tends to increase as we iterate through the algorithm and ultimately the bound on $\bar{L}$ is not satisfied and the algorithm terminates.

Although the above algorithm is similar to *Adaboost.R*, we have no proof that it will terminate in a finite number of steps to zero training error. Convergence to zero training error depends on obtaining a learning machine such that $\bar{L}<.5$. On real data and with real learning machines, as the number of hard training examples increases in the training set, it becomes more difficult to meet this bound. It should be noted that even if we could find a real learning machine that always meets this bound, boosting algorithms state nothing about performance on a separate test set. Therefore, we must insure that as we decrease the training error, we are not overfitting to the idiosyncrasies of a training set and that there will be good generatization to the test data. How this is done depends on the particular implementation of the learning machine and will be discussed in the sections on the construction of trees.

From now on $y_i^{(p)}(\boldsymbol{x}_i)$ refers to the ensemble prediction for pattern i when we are referring to multiple (committee) machines.

4. TREES

4.1 Construction

We basically use Breiman's Classification and Regression Trees with acronym CART [Breiman, *el. al* (1984)] with a pruning modification. To simplify this discussion, we first assume that x is univariate and we have N_1 training examples arranged so that $x_1 \le x_2 \le \cdots \le x_{N_1}$. Because we sample with replacement, some of the samples may be identical. We first ask ourselves, that given a set of observations of the dependent variable y, what is the one number y_A that best characterizes those N_1 values of y. If we are to minimize $E[y-y_A]^2$, then the value that minimizes this is $y_A=E[y]$. Our approximation to E[y] is the sample mean which is $\bar{y}_A$, and the A indicates "Above" where the tree is to be constructed so that the initial root node is at the top. The root node becomes a parent node which spawns two children. Each of the children becomes a parent node which in turn spawn two more children. Hence a parent node is always "above" its children. Thus, given an input x, and using only the root node, we would make a prediction $\bar{y}_A$, (independent of the value of x). and this is the value assigned to the root node.

After the root node is constructed, we generate two children node in the following manner: We determine a "split" value of x, termed x_s so that if $x \le x_s$ we predict $y^{(p)}=\bar{y}_L$ and if $x>x_s$, we predict $y^{(p)}=\bar{y}_R$ (L for "left" and R for "right"). It would be delightful if that split on x simultaneously split y such that all the sample values of y that arrive in the left node are identical (standard deviation is zero) and all the values of y in the right node were a different value of y, but again with a standard deviation of zero. This is probably not going to happen so we find the value of x_s such that the total squared error is minimized:

$$TSE=\sum_{i:x\le x_s}[y_i-\bar{y}_L]^2+\sum_{i:x>x_s}[y_i-\bar{y}_R]^2$$

This is equivalent to minimizing $TSE=n_L\sigma_L^2+n_R\sigma_R^2$ where n_L is the number of training samples that end up in the left node and σ_L is the sample standard deviation (with n_L in the denominator) and σ_R and n_R refer to the right hand node. At this stage, all the training examples sit in one of the two children nodes and the predicted value of the examples in each of the nodes is the mean of the samples in that node. For a new test example, depending on whether the value of the independent variable is less than or greater than the split value of the root node, the predicted value would either $\bar{y}_R$ or $\bar{y}_L$.

For univariate x, we would recursively split at each node, generating two children nodes from each parent node until the variance within each node is zero. This could happen if there is only a single sample in a node or all the samples have the same value of the dependent variable. If $\boldsymbol{x}$ is multivariate, then at each node, we examine each feature of $\boldsymbol{x}$, and determine its best split value. That feature with the minimum TSE becomes the feature to split on. Every time we determine two new nodes from a parent node, the TSE decreases until we decrease to zero. However, this is done on the training set, and there is no guarantee that we will do as well on a separate test set. Therefore we prune the tree to improve generalization.

4.2 Pruning

CART does pruning using cross-validation. However, we prefer a separate pruning set that is 20% of the training set size. The value of 20% is somewhat ad hoc but is based on considerations of the best training-test split for classification [Kearns (1996)]. The primary rationale for a separate pruning set is as follows: In boosting, each machine is constructed on a different probability distribution space because the probability of a particular pattern being in the training set has been altered by the weighting process. We would like to prune on a data set that has similar statistics. For the first machine, the probability of any member of the original pruning set being in the pruning set for that machine are equal. Subsequent to the first machine, we alter the weights similar to that of the training set. That is, the weight w_i for a member of the pruning set is $w_i \rightarrow w_i\beta^{**}(1-L_i)$ where L_i is the loss of that pruning pattern (but β is calculated from the average loss of the training patterns).

Because pruning tends to severely reduce the number of nodes in a tree, there is no point in building the tree until terminal nodes only have one member. (A terminal node may have more than one sample if all the samples have the same value of the dependent variable). Therefore, we modify our building process as follows: Recursively, build (using the training set) to minimize TSE until one of the following happens: (a) there are less than six samples in a node (b) the variance of the samples in the node is zero or (c) the TSE obtained by generating two children nodes from a parent node does not reduce the TSE (from the parent node) by at least 5%. These conditions prevent building nodes that would have been eliminated by pruning anyhow. Pruning takes place as follow: pass all the pruning examples through the tree and store the observed value of y_i at *every* node it passes through, including the terminal nodes. Starting at parent nodes which have at least one child as a terminal node, make that parent node (called "A" for Above node here) a terminal node if:

$$\sum_{A\ node}[y_i-\bar{y}_A]^2 < \sum_{L\ node}[y_i-\bar{y}_L]^2 + \sum_{R\ node}[y_i-\bar{y}_R]^2$$

Note that the y_i are from the pruning set while the $\bar{y}$'s are from the training set. If the y_i were from the training set, the above condition would never be true. Each parent node which becomes a terminal node is in turn examined to see if its parent should be made a terminal node. We recursively repeat this, working our way towards the root node at the top of the tree. Let us define a generic mean square error (MSE):

$$MSE = \frac{1}{N}\left[\sum_{i=1}^{N}[y_i-y_i^{(p)}(\boldsymbol{x}_i)]^2\right]$$

If the $(y,\boldsymbol{x})$ come from the training set and N refers to the number of training set examples, then this is the training MSE. Similarly, if we sum over the pruning set examples and N refers to the number of members of the pruning set, then this is the pruning MSE. Summing over the test set gives us the test MSE (and is identical to the PE). In all these cases, the parameters of $y^{(p)}$ are obtained from the training set. After the tree is initially built, the training MSE is smaller than the pruning MSE on this unpruned tree. If the tree is built until each node has a variance (on the training data) of zero, then the training MSE is zero. After pruning, the training MSE on the pruned tree is larger than the training MSE on the unpruned tree. However, after pruning, the pruning MSE on the pruned tree is lower than the pruning MSE on the unpruned tree. Thus pruning not only makes the tree smaller but also improves the generalization to examples never trained on (that is, the pruning set). In step 2 of the algorithm, when we construct a tree, we refer to the pruned tree.

Another advantage of pruned trees are that they are faster than unpruned trees. In some cases, during the pruning process trees have been noted to prune to depth two (one root node and two terminal nodes).

4.3 Advantages and Disadvantages

The primary advantages of trees are that they can be quickly trained (say, as compared to neural networks), and are non-parametric. The main disadvantages are that the decision space has boundaries that are parallel to the features axes and do not allow modeling based on powers or products of features. It is possible to make decision surfaces that are oblique to the axes, using so-called oblique decision trees [Brodley and Utgoff (1995), Ittner and Schlosser (1996), Murthy *et. al.* (1993), Mruth *et. al.* (1994)] and in fact CART has that option. Also, instead of the input to the tree being the features, we could have as inputs both products of features and features raised to some powers. All these options make the building of trees much slower.

5. PREDICTION PROBLEMS

There are four sets of problems we considered. The first three problems were used by Friedman (1991) in his work on multivariate adaptive regression splines (MARS):

Friedman #1 is a nonlinear prediction problem which has 10 independent variables that are uniform in [0,1].

$$y=10\sin(\pi x_1 x_2)+20(x_3-.5)^2+10x_4+5x_5+n$$

where n is N(0,1). Therefore, only five predictor variables are really needed, but the predictor is faced with the problem of trying to distinguish the variables that have no prediction ability (x_6 to x_{10}) from those that have predictive ability (x_1 to x_5).

Friedman #2,#3 have four independent variables and are respectively:

$$\#2 \qquad y=(x_1^2+(x_2x_3-(1/(x_2x_4)))^2)^{1/2}+n$$

$$\#3 \qquad y=\tan^{-1}\left[\frac{x_2x_2-(1/x_2x_4)}{x_1}\right]+n$$

where the noise is adjusted to give 3:1 ratio of signal power to noise power and the variables are uniformly distributed in the following ranges:

$$0 \le x_1 \le 100$$
$$20 \le (x_2/2\pi) \le 280$$
$$0 \le x_3 \le 1$$
$$1 \le' x_4 \le 11$$

We use 200 training examples, 40 pruning samples, and 5000 test examples per run and 10 runs to determine the best loss function. We follow the ten runs with 100 runs using the best loss function for each of the three prediction problems. The reason for the large number of test examples is to get a reliable estimate of the modeling and prediction error. Because we have access to the truth, we report both the modeling and prediction error.

Boston Housing: This has 506 cases with the dependent variable being the median price of housing in the Boston area. There are twelve continuous predictor variables. This data was obtaining from the UCI database (anonymous ftp at ftp.ics.uci.edu in directory /pub/machine-learning-databases). In this case, we have no "truth", only the observations. Thus we report only the prediction error. We use 25 test cases. Of the remaining 481 cases, 80 are used for pruning and 401 for training. We repeat the boosting and bagging procedures 100 times, each time picking (without replacement) a different training, pruning, and test set.

6. BOOSTING AND BAGGING RESULTS

6.1 Trees on Nonlinear Functions

We report the results on the Friedman functions (Tables I-III) using either a single tree, or bagging (using 50 trees), or boosting (maximum of 75 trees or when the average loss exceeds .5) with three loss functions. The rationale for using these two choices of maximum number of trees is that asymptotic behavior has been reached when the number of trees is substantially less than these two choices. Averages are over ten runs. As we increase the number of trees in an ensemble, the prediction and modeling errors reach a minimum and it is an average of those minima for the 10 runs that is reported under ME and PE. In other simulations, we can't determine the ME because the truth is unknown. Therefore, it would be comforting to know that when the PE reaches a minimum, the minimum of the ME is reached. This is ME2, the value of the modeling error when the minimum of the prediction error is reached. It is comforting to see that ME2 and ME are close or identical.

It may be the case that the standard deviation of the ME or PE is such that a clear winner for boosting against bagging cannot be determined. Experiments were therefore done as follows. We always used the same test set of size 5000. There were 10 sets of training and pruning data. In the first experiment, we used the same training (size 200) and validation set (size 40) on both the boosting machines and bagging machines and then compared the results on the test set. We call an experiment a success if boosting is better than bagging on the same test set when trained and pruned on the same data. We then used the second set of training and pruning data and then compared the test results on the original test data. This was iterated for the rest of the ten training and validation sets. If boosting and bagging are approximately equal in performance , then one would expect boosting to win half the time and lose half the time of the total ten experiments. Let p be the probability of boosting beating bagging. If we make the hypothesis that p=.5, then we have a binomial experiment with the probabilities:

$$P[\#successes \ge 8] \le 5\% \qquad P[\#successes = 10] \le 1\%$$

Therefore, if the number of times boosting is better than bagging is 8 or better, we can reject the hypothesis that the two algorithms are equal (with reject level of 5%). Similarly, if in all the 10 experiments, boosting is better than bagging, than this could happen with probability less than 1% under this hypothesis that the performances are equal. As can be seen in the tables, boosting is better than bagging if the best loss function is used for the first and third Friedman functions. In function #2, there is no clear winner if the best loss function is used. Function #2 is unusual in that the contribution to the prediction error is primarily due to noise.

Subsequent to the original set of ten runs, we did 100 runs on the best results from these first three tables. We used 100 sets of training data (size 200), 100 sets of validation data (size 40) and one set of test data (size 5000). If using the same training and validation data, boosting was better than bagging on the test set, we count it as a win for boosting (Table IV). We reach the same conclusions as we did using only ten runs.

Breiman has results on these three functions using bagging. Our bagging results are much better than his. The major difference in procedure is that he used a total of 200 training examples, while we used 200 training examples plus 40 pruning examples. Therefore, we did another ten runs on function #1 using bagging with 166 training examples and 34 pruning examples. Our modeling error for bagging is 2.58 which is still substantially less than Breiman's results of 6.2. We can only attribute the difference to the fact that we use a separate pruning set which tends to give better generalization.

6.2 Stacking Trees on Nonlinear Functions

Stacking [Wolpert (1992)] is not a particular algorithm but a generic name for the following observation: if we train on part of the training set, then the performance of the learning machine on training data that was not part of

Table I. *Results of ten runs on single trees, bagged trees and boosted trees on Friedman #1. #better indicates how many times (out of ten runs) that boosting is better than bagging on the same training, pruning, and test sets. ME2 is the average modeling error at the minimum of the prediction error*

	modeling error	**#better**	**ME2**	**prediction error**
single	3.58		3.20	4.65
bagging	2.20		2.20	3.31
boost (linear loss)	1.65	10	1.65	2.75
boost (exponential)	1.67	9	1.68	2.79
boost (square law)	1.73	8	1.77	2.84

Table II. *Results of ten runs on Friedman #2*

	modeling error	**#better**	**ME2**	**prediction error**
single	29310		29310	77511
bagging	11463		11463	65316
boost (linear loss)	11684	3	11708	68622
boost (exponential)	10980	5	11108	64703
boost (square law)	15615	0	15660	69585

Table III. *Results of ten runs on Friedman #3*

	modeling error	**#better**	**ME2**	**prediction error**
single	.0491		.0491	.0840
bagging	.0312		.0312	.0697
boost (linear loss)	.0218	8	.0218	.0604
boost (exponential)	.0213	9	.0214	.0602
boost (square law)	.0202	10	.0202	.0588

Table IV. *Results of 100 runs on bagged trees and boosted trees on the three Friedman functions. #better indicates how many times (out of 100 runs) that boosting is better than bagging on the same training, pruning, and test sets. The loss functions used were the best of Tables I, II and III.*

	ME-bagging	**ME-boosting**	**# better**	**PE-bagging**	**PE-boosting**	**loss-type**
#1	**2.26**	**1.74**	**94**	**3.36**	**2.84**	**linear**
#2	**10,093**	**10,446**	**43**	**66,077**	**65,955**	**exponential**
#3	**.0303**	**.0206**	**90**	**.0677**	**.0596**	**square law**

Table V. *Stacking on Friedman #1 (boosting uses linear loss function))*

	modeling error	**prediction error**	**#trees**
bagging (no stacking)	2.20	3.31	50.
bagging (stacking)	1.91	3.02	14.4
boosting (no stacking)	1.65	2.75	59.2
boosting (stacking)	1.43	2.56	10.4

Table VI. *Stacking on Friedman #2 (boosting uses exponential loss)*

	modeling error	**prediction error**	**#trees**
bagging (no stacking)	11463	65316	50.
bagging (stacking)	13702	67795	14.3
boosting (no stacking)	10980	64703	58.4
boosting (stacking)	13129	67110	22.2

Table VII. *Stacking on Friedman #3 (boosting uses square loss)*

	modeling error	**prediction error**	**#trees**
bagging (no stacking)	.0312	.0697	50.
bagging (stacking)	.0256	.0644	8.9
boosting (no stacking)	.0203	.0604	46.3
boosting (stacking)	.0202	.0590	15.6

the training set for that particular machine gives us additional information.

Suppose we are now given the individual machines, whether obtained from boosting or bagging and ask the best way to combine them rather than using averaging (for bagging) or the weighted median (for boosting). Breiman (1996a) suggests the following stacking technique: Suppose that once again pattern i has an observed value y_i on the training set. Suppose machine k has a predicted value $y_i^{(k)}$ for pattern i on the training set where there are a total of K machines. Then find the γ_k that minimizes:

$$W = \sum_{i=1}^{N_1} \left[y_i - \sum_{k=1}^{K} \gamma_k y_i^{(k)} \right]^2 \qquad \gamma_i \geq 0.$$

This is a constrained quadratic optimization problem for which the use of standardized quadratic programming packages is recommended. The above is equivalent to minimizing (with respect to $\boldsymbol{\gamma}$):

$$W = \boldsymbol{C}^t \boldsymbol{\gamma} + \frac{1}{2} \boldsymbol{\gamma}^t \boldsymbol{H} \boldsymbol{\gamma}$$

where $\boldsymbol{C}$ is a vector and $\boldsymbol{H}$ is a Hessian whose elements are:

$$c_k = -2\sum_{i=1}^{N_1} y_i y_i^{(k)} \quad k=1,...,K$$

$$h_{jk}=2\sum_{i=1}^{N_1} y_i^{(j)} y_i^{(k)} \quad j,k=1,...,K \quad \text{and} \quad h_{jk}=h_{kj}$$

Tables V, VI, and VII show the results of using stacking using the best loss functions (for boosting, from Tables I-III). Note that if $\gamma_k=0$, machine k is not needed. The last column of these tables indicate the number of trees in the stacked or unstacked implementation. There are mixed results. On Friedman #1, both bagging and boosting improve over their unstacked results at a 5% significance level, but boosting still is better than bagging. For Freidman #2, stacking makes both boosting and bagging worse while for Friedman #3, stacking makes bagging better but boosting is still better than stacking.

6.3 Trees on Boston Housing

For Boston housing, using 100 runs, and the normal approximation to the binomial, we find since $\sigma=\sqrt{(100)(.5)(.5)}=5$, that $P[\#successess \geq 61.65] < 1\%$. Therefore, if boosting beats bagging more than 61.65% of the time, we can reject the hypothesis that the two algorithms are equally good. Over 100 runs, we get an average prediction error of 12.4 using bagging and 10.7 using boosting. Boosting is better than bagging 72% of the time. On this data Breiman (1996a) obtained 11.7 while we obtained 12.4. Without knowing the error bars on his results, we have no way of telling whether our results on bagging are significantly different. There is a procedural difference: Breiman uses a training set of size 481 and a test set of size 25, while we use a training set of size 401, a pruning set of size 80,and a test set of size 25. Stacking on top of boosting or bagging does not improve results.

8. CONCLUSIONS

Boosting is a viable approach to reducing prediction error. It gives statistically significant improvement in most cases and never is statistically worse than bagging. In constructing trees, it is important to generate a pruning set of data whose statistics are similar to those of the training set. Pruning improves generalization and increases speed. Stacking sometimes helps, sometimes hurts.

Futhure studies will examine the use of neural networks or oblique decision tree as learning machines, and the feasibility of using nonlinear functions of features as inputs to decision trees.

Acknowledgements

This author gratefully acknowledges the many illuminating discussions held with Yoav Freund and Robert Schapire. This work was supported by ARPA contract N00014-94-C-0186 while employed as a consultant to Lucent Technologies-Bell Labs.

References

Breiman, L.(1996a). "Stacked Regressions", *Machine Learning*, vol. 24, No. 1, pp. 41-64. Also at anonymous ftp site: ftp.stat.berkeley.edu/pub/tech-reports/367.ps.Z.

Breiman, L. (1996b). "Bagging Predictors" *Machine Learning*, vol. 26, No. 2, pp. 123-140. Also at anonymous ftp site: ftp.stat.berkeley.edu/pub/tech-reports/421.ps.Z.

Breiman, L. (1996) "Bias, Variance, and Arcing Classifiers" Technical Report 460, Department of Statistics, University of California, Berkeley, CA. *Annals of Statistics* (to be published) Also at anonymous ftp site: ftp.stat.berkeley.edu/pub/tech-reports/460.ps.Z.

Breiman, L., and Spector, P. (1992). "Submodel Selection and Evaluation in Regression. The X-Random Case", *International Statistical Review*, **60**, 3, pp.291-319.

Breiman, L., Friedman, H.H., Olshen, R.A., and Stone, C.J. (1984). *Classification and Regression Trees*, Wadsworth International Group.

Brodley, C.E., and Utgoff, P.E. (1995), "Multivariate Decision Trees", *Machine Learning*, **19**, pp. 45-77.

Drucker, H., (1996) and Cortes, C. (1996) "Boosting Decision Trees", *Neural Information Processing 8*, ed: D.S. Touretzky, M,C. Mozer and M.E. Hasselmo. Morgan Kaufmann, pp.479-485.

Drucker, H., Cortes, C., Jackel, LD., LeCun, Y., Vapnik V. (1994). "Boosting and Other Ensemble Methods",

Drucker, H., Schapire, R.E., Simard, P., (1993), "Boosting Performance in Neural Networks",

Efron, B., and Tibshirani, R. (1993), *An Introduction to the Bootstrap*, Chapman and Hall.

Fisz, M., (1963). *Probability Theory and Mathematical Statistics*, John Wiley.

Freedman, J.H., (1991), "Multivariate Adaptive Regression Splines", *The Annals of Statistics*, **19**, No. 1, pp. 1-82

Freund, Y., and Schapire, R.E. (1996a). "Experiments with a New Boosting Algorithm", *Machine Learning: Proceedings of the Thirteenth Conference*, ed: L. Saitta, Morgan Kaufmann, pp. 148-156. Also at web site: http://www.research.att.com/˜yoav. or http://www.research.att.com/orgs/ssr/people/˜yoav.

Freund, Y., and Schapire, R.E. (1996b). "A decision-theoretic generalization of on-line learning and an application to boosting", at web site: http://www.research.att.com/˜yoav. or http://www.research.att.com/orgs/ssr/people/˜yoav. An extended abstract appears in the *Proceedings of the Second European Conference on Computational Learning Theory*, Barcelona, Springer-Verlag, March, 1995, pp. 23-37.

Ittner, A., and Schlosser, M.(1996), "Non-Linear Decision Trees-NDT", *Machine Learning: Proceedings of the Thirteenth Conference*, ed: L. Saitta, Morgan Kaufmann, pp. 252-257.

Kearns, M. (1996). "A Bound on the Error of Cross Validation Using the Approximation and Estimation Rates, with Consequences for Training-Test Split", *Neural Information Processing 8*, ed: D.S. Touretzky, M,C. Mozer and M.E. Hasselmo. Morgan Kaufmann, pp. 183-189.

Murthy, S., Kasif, S., Salzberg, S., and Beigel, R. (1993). "OC!: Randomized induction of oblique decision trees" *Proceeding of the 11th National Conference on Artificial Intelligence (AAAI-93)*, MIT-Press.

Mruth, S., Kasif, S., Salzber, S. (1994), "A System for Induction of Oblique Decision Trees", *Journal of Artificial Intelligence Research,* **2**, Morgan Kaufmann.

Schapire, R.E., (1990), "The Strength of Weak Learnability", *Machine Learning*, **5**, 2, pp. 197-227.

Wolbert, D.H. (1992), "Stacked Generalization", *Neural Networks*, **5**, pp. 241-259.