Fast Committee Machines for Regression and Classification

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

Abstract

In many data mining applications we are given a set of training examples and asked to construct a regression machine or a classifier that has low prediction error or low error rate on new examples. An important issue is speed especially when there are large amounts of data. We show how both classification accuracy and prediction error can be reduced by using boosting techniques to implement committee machines. In our implementation of committees using either classification trees or regression trees, we show how we can trade off speed against either error rate or prediction error.

keywords: regression, classification, committee machines, boosting, data mining

"Fast Committee Machines for Regression and Classification", Proceedings *Third International Conference of and Data Mining, eds.* David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, AAAI Press, Menlo Park, CA. 1997

1. INTRODUCTION

Boosting techniques [Drucker (1993, 1994, 1996a), Freund and Schapire (1996a,b), Schapire (1990)] allows one to obtain smaller prediction errors (in regression) and lower error rates (in classification) using multiple predictors. The ensemble of classifiers is often called a committee machine. As shown in Figure 1, the same set of input features (independent variables) is applied to what are termed the "weak learners" (defined later) and each learner puts forth an hypothesis h_i with a confidence inversely related to β_i. For classification, h_i is either 0 or 1 (for the two class case) or the predicted value of the dependent variable (in the case of regression).

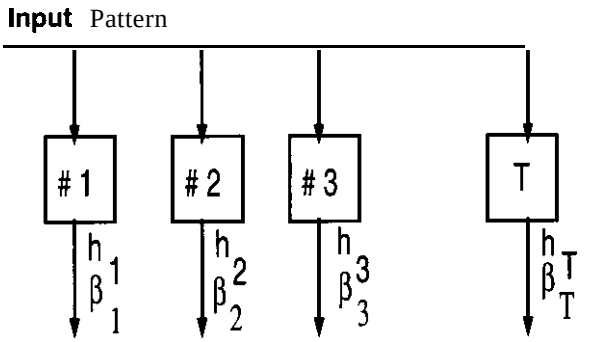

FIGURE 1. **BOOSTING ENSEMBLE**

In a serial machine, the outputs of the weak learners are output sequentially so as we move down the chain of learning machines, the execution speed decreases but generally the prediction error or classification error also decreases. Therefore, by choosing to use only i $(i \le T)$ of the T learners, we can choose the appropriate combination of execution speed and either classification accuracy or prediction accuracy. If parallel implementation is feasible (either each learner or blocks of N learners), then speed is governed by the slowest weak learner or the slowest block of N learners.

Several studies of boosting in classification [Breiman (1996b), Freund and Schapire (1996a)] have shown the superiority of boosting over another committee machine approach termed bagging [Breiman (1994,1996a)] but this is the first experimental study where we also investigate combining regressors using boosting techniques. In boosting each machine is trained on different subsets of the training set. The first machine is trained on examples picked with replacement from the original training set. We then pass all the training patterns through this first machine and note which ones are in error (in classification). For regression machines, those patterns whose predicted values differ most from their observed values are defined to be "most" in error. For those patterns in error, the sampling probabilities are adjusted so that difficult examples are more likely to be picked as members of the training set for the second machine. Thus, different machines are better in different parts of the observation space. Regressors are combined using the weighted median while classification machines are combined using a weighted sum of the hypotheses. Details of these weighting scheme are discussed later.

For classification, the parameter of interest is the error rate, while for regression, the parameters of interest are the modeling error (ME) and the prediction" error (PE). We discuss regression first: Suppose we are given a set of *observations*, $(y_i, \mathbf{x}_i)$ $i=1,...,N_1$ where N_1 is the number of training set observations, and $\mathbf{x}$ is an M-variate vector forming the features. The probability density function of $(y,\mathbf{x})$ is fixed but unknown. Based on these observations, we form a predictor $y^{(p)}(\mathbf{x})$. We define a sample modeling error (ME) and prediction error (PE):

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

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

where $y_i^{(p)}(\mathbf{x}_i)$ 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 $y^{(p)}(\mathbf{x})$ are obtained from the N_1 training set observations but the y_i and $\mathbf{x}_i$ 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 $(y, \mathbf{X})$ 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.

Similarly, for the classification case, we have a training error for those examples we train on and a test error for the test examples tested on a machine constructed using the training examples.

Our implementation of each weak learner is a tree, either a regression tree or a classification tree because of their ease of implementation, small training time, and fast execution speed. In part II of this paper, we discuss the implementation of the boosting algorithm for both classification and regression. In part III we discuss the issues of implementation of the constituent weak learners while in Part IV we show the results of regression and classification machines and the speed versus performance trade-offs. Finally, in the conclusion, we summarize our findings and discuss the issues of neural networks versus trees as the constituent weak leaners.

II. BOOSTING

The implementation shown in Figure I is that of AdaBoost (Freund and Schapire, 1996b), which is the third in a series of boosting algorithms [Schapire (1990), Freund, (1990)]. The implications of a boosting algorithm is that one can take a series of learning machines (termed weak learners) each having a poor error rate (but less than 50%) and combine them to give an ensemble that has very good performance (termed a strong learner). Boosting arises from the PAC (probably approximately correct) learning model (Schapire, 1990) which has as one of its primary interests the efficiency of learning. X is a set called the domain. We discuss boosting as applied to classification in the two-class case (although it can be extended to the multi-class case): A concept class C is a collection of concepts c: $X \rightarrow \{0,1\}$ One has access to a set of labeled examples of the form (x. c(x)) where x is chosen randomly according to some fixed but unknown and arbitrary distribution D on the domain X. The learner must output a hypothesis h:$X \rightarrow [0,1]$. A *strong* PAC-learning algorithm is an algorithm that outputs (in polynomial time with respect to $1/\varepsilon$ and $1/\delta$) with probability $1-\delta$ a hypothesis with error at most ε where $\varepsilon, \delta > 0$. A ***weak*** PAC-learning algorithm is only required to to have an $\varepsilon \leq .5 - \gamma$ where γ is a small positive constant. Schapire (1990) was the first one to show that a series of weak learners could be converted to a strong learner. In AdaBoost, a series of weak learners are constructed iteratively. The latest weak learner must learn a distribution filtered by the previous weak learners. Difficult examples tend to be over-represented after filtering, easy examples less so. Thus each weak learner must learn a progressively more difficult distribution. The trick is to have a weak learner that can learn that more difficult distribution with error rate less than .5. Although the boosting algorithm is originally described in terms of classification where the definition of error is obvious, we can generalize the definition of error to a loss function for regression:

This is the boosting algorithm for regression:

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. Inititally, the w_i are set equal to unity. 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: $h_t: x \rightarrow y$

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

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

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

then we have three candidate loss functions we examined:

$$L_i=\frac{|\,y_i^{(p)}(x_i)-y_i\,|}{D} \qquad \text{(linear)}$$

$$L_i=\frac{|\,y_i^{(p)}(x_i)-y_i\,|^2}{D^2} \qquad \text{(square law)}$$

$$L_i=1-\exp\left[\frac{-|\,y_i^{(p)}(x_i)-y_i\,|}{D}\right] \qquad \text{(exponential)}$$

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: $w_i \rightarrow w_i \beta^{**}[1-L_i]$, 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 x_i, each of the T machines makes a prediction $h_t, t=1,...,T$. Obtain the cumulative prediction h_f "sing 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, 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: we have: $y_i^{(1)}<y_i^{(2)}<,,<y_i^{(T)}$ (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.

9. Repeat

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. 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.

For classification, we replace steps 4 and 8 above. In step 4, the loss for pattern i becomes $L_i=|$ h,(i) -c(i) $|$ where h,(i) is the hypothesis of machine t (0 or 1) and c(i) is the correct classification (either 0 or 1). The issue of multiclass classification will be discussed in the conclusion. In step 8, the final hypothesis is:

$$h_f(i)=\begin{cases}1, & \sum_{t=1}^{T}(\log\frac{1}{\beta_t})\, h_t(i)\geq\frac{1}{2}\sum_{t=1}^{T}\log\frac{1}{\beta_t}\\ 0, & \text{otherwise}\end{cases}$$

Note that the examples referred to in the algorithm above only refers to training examples and we usually observe convergence to zero classification error or zero prediction error on these training example on these training exampless. However, we are interested in the performance on a test set. Therefore, at each stage of boosting, we pass the test set through the ensemble and use step 8 to obtain the test performance. How well we generalize to the test set will depend on our choice of weak learner, e.g.. neural net, tree, nearest neighbor classifier. Generally, our experiments show that nearest neighbor classifiers do not do well in an ensemble while trees and neural nets do.

III. Trees

Trees are used to implement our weak learners. First we discuss regression trees based on Breiman's CART [Breiman, 1994]. In the implementation of regression trees, one starts (at the top) with a main node that examines a specified feature of the input pattern and compares its value to a "separator" value assigned to that node. If the value of the feature is less than or equal to tbe separator, we continue down the left side of the tree, else the right side. Each of the children of the main node has a specified feature and separator value for that feature. Each parent node has two children nodes and those children nodes become parents of their children. Depending on the values of the features for the input pattern, we continue down the tree, at each node going left or right until we reach a terminal leaf. At the terminal leaves, we assign a hypothesis that is the predicted value of the regression.

The details of the construction of a regression tree are in [Breiman, 1984]. Basically, we recursively examine the members of the training set that reach each child node and generate two children until there are no samples left in any child node. Tbe feature and the separator value are such that the training set in each node is best split into two subsets obtained in the following manner: we order each independent value and recursively examine each value of each independent variable temporarily assigning that value as a separator. This separator will split the ***dependent values*** into two subsets, each of which has a mean and a sum of squared deviations from those means. That feature, and that separator value that minimizes the sum of the squared errors in two new child nodes is-the optimum feature and separator value. Thus the ideal situation occurs at a node if the separator, splitting on a value of some feature sends all training examples with the same value of the dependent variable to the left and all training examples with another value to the right. In this way, the mean squared errors within each child node arc zero and the nodes become terminal nodes with predicted values that are the means of the training examples in that node. In general, this does

not happen and each child node has an assigned value of the dependent variable that is equal to the mean of the dependent variable within that node. When there is only one member of the training set within that node, the predicted value is the value of the dependent variable within that node.

For classification trees, the building criterion is based on an information theoretic basis found in C4.5 [Quinlan, 1993].

Trees can be constructed until there is only one example of the training set in each node in which case both the training classification rate or the prediction error is zero. Other stopping criterions can also be used such as a fixed number of examples in a node or the classification or prediction error must be reduced by a certain percentage when adding children nodes. To generalize well to as an yet unseen test set, we then prune back the trees using a separate pruning set. Pruning [Mingers, 1989] performs two essential tasks (a) it increases the execution speed because the tree is smaller and (b) the generalization is better.

IV Experiments

We tried several cases of classification and regression using the techniques described above. The first regression problem was based on that of Friedman (1991), which is a nonlinear prediction problem which has 10 independent variables that are uniform in [0,1] Here we can determine both the modeling and prediction error.

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

where n is normal(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). For different training set sizes (pruning size is 20% of the training set size) and a test set of of size 10,000 we plot (Figure 2) execution speed (on a SparcStation 20) versus the prediction error on the test set.

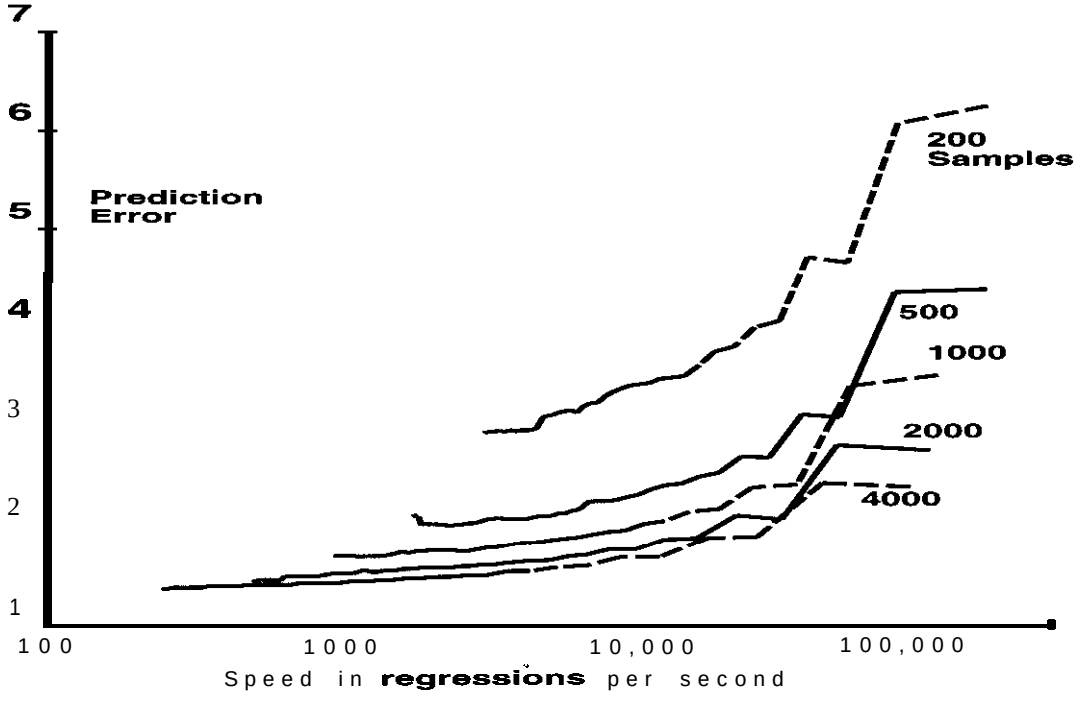

Figure 2. Prediction error versus speed for a regression problem

Each data point is the average of ten samples. Generally as we increase the number of trees in the ensemble

(thereby decreasing the execution speed), the prediction error decreases although there are some idiosyncrasies at the beginning and end. Generally, the trees in a particular ensemble are very different in size ranging from approximately depth log, [number of examples] (before pruning) down to depth 2 (after pruning). Because of the inherent noise floor in this prediction problem, the PE tend to asymptote with increasing number of trees and large enough number of examples. There is a continuum of prediction error versus speed trade-offs. We performed another experiment with similar results. In Table I, we show the modeling and predictions error for the function above (termed fr1) and another function (termed fr3) which are discussed in Friedman (1991). frl is of dimensionality 10 while fr3 is of dimensionality four. Boosting on frl uses the linear loss function detailed in the regression algorithm while boosting on fr3 uses the square loss function. The number of training patterns is incremented until the PE asymptotes. Normally, the modeling error cannot be determined because the "truth" is not known. Boosting with more than 4000 training examples does not seem to increase performance.

Table I. *Modeling and prediction error on two functions. Boosting refers to the ensemble performance; single* ***refers*** *to the performance of a single learning machine. The prediction error of function frl is in Figure 1. Test set size is 10,000.*

training	function **frl**				**function fr3**			
set size	**ME boosting**	**ME single**	**PE boosting**	**PE single**	**ME boosting**	**ME single**	**PE boosting**	**PE single**
200	**1.9221**	**5.024**	**3.087**	**6.213**	**.02005**	**.05418**	**.05973**	**.09364**
500	**0.9128**	**3.220**	**2.068**	**4.364**	**.01154**	**.04156**	**.05113**	**.08118**
1000	**0.5523**	**2.331**	**1.704**	**3.490**	**.00786**	**.02668**	**.04732**	**.06596**
2000	**0.3663**	**1.578**	**1.511**	**2.733**	**.00576**	**.02122**	**.04524**	**.06057**
4000	**0.2292**	**1.208**	**1.375**	**2.367**	**.00448**	**.01551**	**.04395**	**.05504**

A two-class classification problem of dimensionality 100 is obtained from a 10x10 image size database (total size 120,000) and assigning digits zero to four to class 0 and digits five through nine as class 1. A subclass of these digits is considered a "hard problem" and performance on a test set of size 10,000 is plotted in Figure 3. There is another subset termed an "easy problem" and the details of how to prefilter the digits into hard and easy problems are discussed in Drucker (1996). A comparison of the two problems is shown in Table II.

Table II. *Error rate (in %) for boosted classification trees and a single tree for two problems: a "hard" problem and an "easy" problem. Test set size is 10,000*

training	**HARD**	**PROBLEM**	**EASY**	**PROBLEM**
set size	**error-rate boosting**	**error-rate single**	**error-rate boosting**	**error-rate single**
500	**29.7**	**39.8**	**9.11**	**18.2**
1000	**12.9**	**24.4**	**6.34**	**15.6**
2000	**8.86**	**21.5**	**4.45**	**14.4**
4000	**6.70**	**18.6**	**2.91**	**12.0**
8000	**4.92**	**15.6**	**2.30**	**10.5**

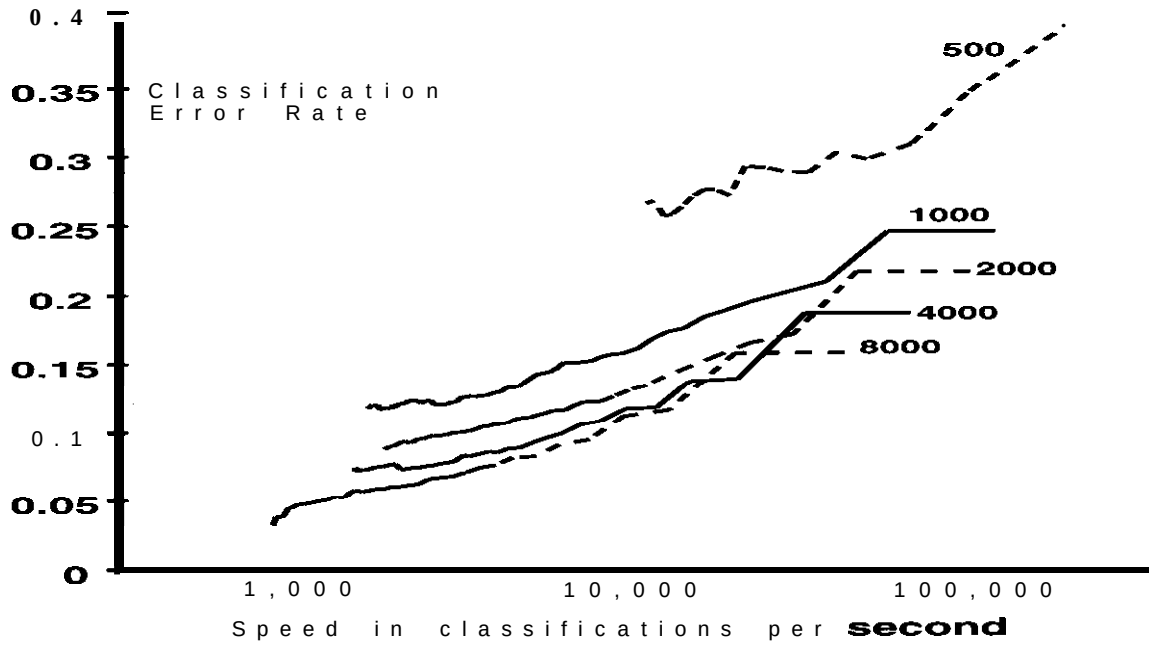

Figure 3. Classification error rate versus speed with number of training examples as a parameter.

V. Discussion and Conclusions

We have shown how to use boosting to reduce error rates in classification and prediction error in regression using classification trees and regression trees, respectively. Of critical importance is the pruning of the trees by using a separate set of pruning data after the tree has been initially grown using the training set. Pruning both increases the speed and improves the generalization to samples as yet unseen.

By picking the appropriate number of trees in the ensemble, one can trade off speed versus performance. If the speed performance is not acceptable (but the error rate or prediction error is), then one choice is a parallel architecture. Generally, there are between forty and seventy trees in an ensemble and therefore the speed would increase by those factors (but only approximately since not all trees are the same size). If error rate or prediction error is the issue, then another possibility is to build a single neural network. Generally a single neural network is slower than these ensembles because trees just implement simple IF statements, while neural networks need to implement both multiplication and some sigmoid function as the transfer functions. However, these issues must be decided on a case-by-case basis.

Here are the general advantages and disadvantages of neural network versus trees whether implemented in an ensemble or as a single machine:

1. Because the decision surfaces of trees must be parallel to the feature space axes, their performance is usually worse than that of neural networks which can implement arbitrary decision surfaces.

2. Neural networks in regression are very slow to learn. Unlike the classification case where stochastic gradient descent (pattern by pattern backprop) is used efficiently (Haykin,1994), in our experience neural nets for regression must be trained using some type of batch training. Conjugate gradient [Haykin(1994), Press(1988)] seems to be the most effective but it is very, very slow to converge. Trees can be trained much more quickly.

3. The optimum architecture (number of hidden layers, number of units in a hidden layer) of a neural network must be obtained through trial and error. Furthermore, the optimum architecture will be a function of the number of training cases. Training of trees is straightforward in that the data defines the architecture of the tree.

Disadvantages of any technique may become moot. For example, even though neural networks in regression are slow to learn and slow in execution (as compared to trees), if the processor is fast enough and the time to learn reasonable for the task at hand, then neural networks with their lower prediction error may be acceptable in spite of those disadvantages.

We have done a study (Drucker, 1997) of using neural networks in a boosting ensemble of regressors for 200 training examples. The performance of the ensemble is a factor of two better but the execution speed is much, much slower (between one and two orders of magnitude smaller). We have also done a unreported study of using trees as a multi-class classifier. Unfortunately, there seems to be technical problems in implementing a boosting ensemble using multi-class decision trees. The reason for this is that a weak learner must have an error rate less than SO% whether the problem be multi-class or two-class. This is much easier to achieve in the two-class problem where random guessing gives an error rate of SO% and a weak learner must do only slightly better than this. In the multi-class case, the random error rate is $(n-1)/n$ where n is the number of classes. As the boosting algorithm iterates, the difficult examples are more likely to appear making it more difficult to obtain the required SO% error rate. Our solution is to design one boosting ensemble for each of n classes. The outputs of the n ensembles are compared in order to determine the classification. We have shown how to combine the very fast ensembles with a highly accurate neural network to achieve a faster speed than the neural network alone but obtain a much higher performance than the ensemble of trees alone (Drucker, 1996)

Acknowledgements

Part of this work was performed under ARPA contract N00014-94-C-0186 while at Lucent Technologies.

References

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.

Breiman, L. (1996) "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.

Drucker, H., (1997), "Improving Regressors using Boosting Techniques", submitted to the 1997 conference on Machine Learning.

Drucker, H., Schapire, R.E., Simard, P., (1993), "Boosting Performance in Neural Networks", *International Journal of Pattern Recognition and Artificial Intelligence,* 7, 4, pp. 705-719.

Drucker, H., Cortes, C., Jackel, LD., LeCun, Y., Vapnik V., (1994). "Boosting and Other Ensemble Methods", Neural *Computation* , 6, 6, pp. 1287.1299. pp. 1287.1299.

Drucker, H., (1996), "Fast Decision Tree Ensembles for Optical Character Recognition", *Proceedings of the Fifth Annual Symposium on Documnet Analysis and Information Retrieval*

Drucker, H., (1996) and Cortes, C., "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., (1997), "Improving Regressors using Boosting Techniques", submitted to the 1997 conference on Machine Learning.

Freund, Y., (1990), "Boosting a Weak Learning Algorithm by Majority", *Proceedings of the Third Workshop on Computational Learning Theory,* Morgan-Kaufmann, 202.216

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.

Friedman J., (1991), "Multivariate Adaptive Regression Splines", *Annal of Statistics,* vol *19, No.* 1, pp. l-141

Haykin, S. (1994). Neural *Networks,* Macmillan

J. Mingers (1989a), "An Empirical Comparison of Pruning Methods for Decision Tree Induction", Machine ***Learning,*** 4:227-243.

Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.,(1988) *Numerical* ***Recipes in C,*** Cambridge

J.R.Quinlan (1993), C4.5: Programs *For Machine* ***Learning,*** Morgan Kauffman.

Schapire, R.E., (1990). "The Strength of Weak Learnability", ***Machine Learning, 5.2,*** pp. 197.227.