# Fast Decision Tree Ensembles for Optical Character Recognition

Harris Drucker
AT&T Bell Laboratories
Holmdel, New Jersey 07733
drucker@monmouth.edu

## Abstract

A new boosting algorithm of Freund and Schapire is used to improve the performance *of an* ensemble of decision trees which are constructed using the information ratio criterion of Quinlan's C4.5 algorithm. This boosting algorithm iteratively constructs a series of decision trees, each decision tree being trained and pruned on examples that have been filtered by previously trained trees. Examples that have been incorrectly classified by the previous trees in the ensemble are resampled with higher probability to give a new probability distribution for the next tree in the ensemble to train on. By combining the very fast decision tree ensemble with a more accurate (but slower) neural network, we are able to obtain a speed up of a factor of eight over the neural network and yet achieve a much lower error rate than the tree ensemble alone.

## 1 INTRODUCTION

A *new* boosting algorithm termed by their inventors (Freund and Schapire, 1995) as AdaBoost can be used to implement an ensemble of decision trees. 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 better than 50%) and combine them to give an ensemble that has very good performance (termed a strong learner). The first practical implementation of boosting was in OCR (Drucker, 1993, 1994) using neural networks as the weak learners. In a series of comparisons (Bottou, 1994) boosting was shown to be superior to other techniques on a large OCR problem.

The general configuration of AdaBoost is shown in Figure 1 where each box contains a weak learner. Here, our weak learner is a decision tree built using Quinlans C4.5 algorithm (Quinlan, 1993). Each weak learner is trained sequentially. The first weak learner is trained on a set of patterns picked randomly (with replacement) from an original training set. After training and pruning, the training patterns are passed through this first decision tree. In the two class case the hypothesis $h_1$ is either class 0 or class 1. Some of the patterns will be in error. The training set for the second weak learner will consist of patterns picked randomly from the original training set with higher probability assigned to those patterns the first weak learner classified incorrectly. Since the training patterns are essentially filtered by the first decision tree, the new training set for the second decision tree is termed the filtered training set. Thus as we proceed to build each member of the ensemble, difficult patterns are more likely to appear in the filtered training set for the particular tree being grown. The training error rate of an ***individual weak learner*** tends to grow as we increase the number of weak learners because each weak learner is asked to

"Fast Decision Tree Ensembles for Optical Character Recognition", Proceeding *of the Fifth Annual Symposium on Document Analysis and Information Retrieval,* ***1996,*** pp.137-147

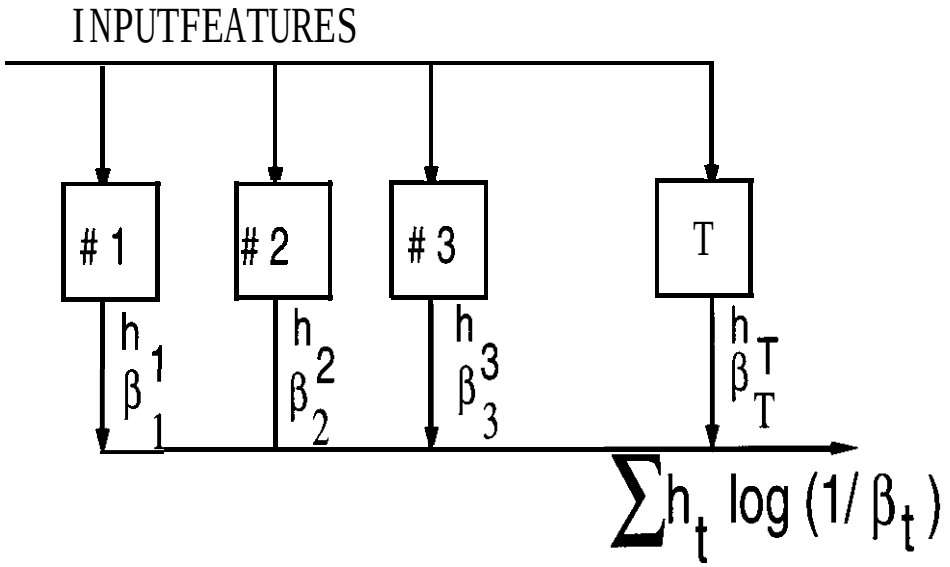


FIGURE 1. BOOSTING ENSEMBLE

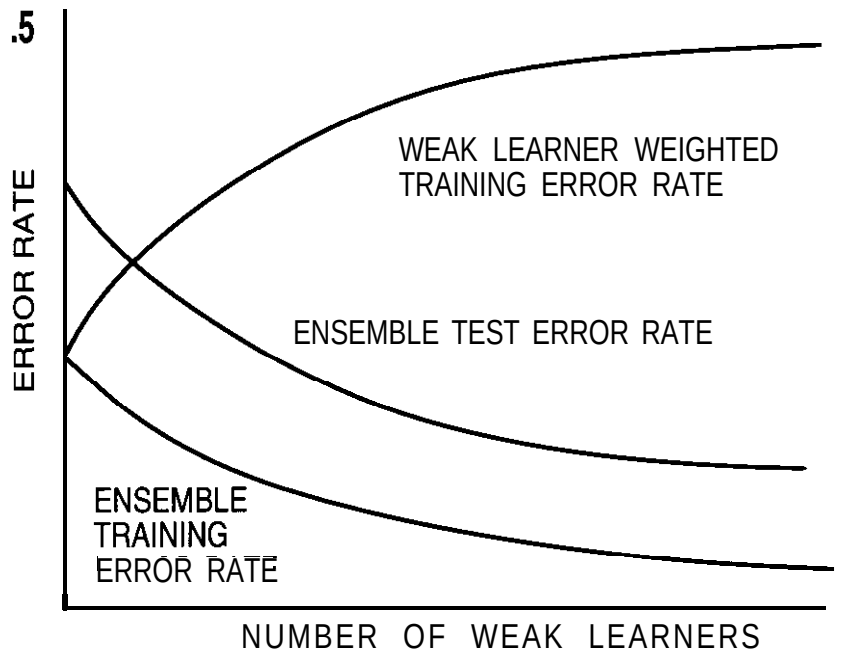


FIGURE 2. INDIVIDUAL WEAK LEARNER ERROR RATE AND ENSEMBLE TRAINING AND TEST ERROR RATES

classify progressively more difficult patterns. However the boosting algorithm shows us that the ensemble training and test error rate decrease as we increase the number of weak learners. Note that the ensemble training error rate is obtaining by passing all the original training patterns through each member of the ensemble. The ensemble classification is determined by weighting the hypotheses with the log of $(1/\beta_i)$, summing over all outputs, and comparing to a threshold to obtain a final decision, class 0 or 1. It turns out that $\beta$ is proportional to the error rate of the weak leaner so that because of the log $(1/\beta)$ term in the summation, those weak learners with smallest error rate influence the summation and hence the decision more than those weak learners with higher error rate.

Figure 2 shows the general shape of the error curves we would expect for the two class case. As we start the process, the first weak learner is the ensemble with a training error rate less than the test error rate. As we increase the number of weak learners, the training set for the N'th weak learner is more difficult than the previous weak learner, therefore the training error for **that weak** learner tends to be larger, eventually approaching .5. However, the ensemble error rates which includes the effects of all weak learners tends to decrease. Once the error rate of the weak learner is .5, then the weak learner is doing no better than guessing and does not contribute to a reduction of the ensemble error rates. Typically, the ensemble training error rate on the original set of training examples approaches zero. Note that these are generic curves. In practice, the weak learner error rate may be quite erratic although it eventually approaches .5. The ensemble training and test error rates tend to be much smoother.

AdaBoost has certain advantages over Version I (Schapire, 1990) and Version II (Freud, 1990), the two previous instantiations of boosting algorithms. The implications of a boosting algorithm is that one can take a series of learning machines (the weak learners) each having a poor error rate (but no worse than .5-y, where typically y is some small positive number) and combine them to give an ensemble that has a very good performance (the strong learner). It is required to know yin order to prove. certain bounds but only in Version II is it actually required to implement the algorithm. Version I is implemented by first training on the original data, then training a second network on examples 50% of which the first machine classifies correctly and 50% of which are. classified incorrectly, and then finally training a third classifier on new examples that the first two neural networks disagree on. In the testing phase, a test pattern is presented to the three weak learners. If the first two weak learners agree, then that classification is used, else the classification of the third weak learner is used. Version II depends on knowing y in advance and how many weak learners are to be used. However, in AdaBoost the knowledge of y and the number of weak learners to be used are not required. Therefore, one can continuously build up the number of weak learners in the ensemble and stop whenever desired. Even though y may be small, we will converge to the ensemble error rate with fewer weak learners the larger y is. Therefore, we search for good weak learners with large $\gamma$.

For our implementation of decision trees, we have a set of features (attributes) which are the grey-level values of the pixel array. The rational for using the grey-level values as the features is to make a very fast tree since nodes of the tree then just make comparisons of the values of selected pixel to that of a threshold determined during the training of the tree. Although we may achieve lower error rates on a single tree

using pre-processing of the data (feature extraction) or using some nonlinear functions of the pixel values at a node, we will depend on the boosting algorithm to give us the low error rate but still maintain the high classification speed.

For our implementation of digit detection, we use ten ensembles. For instance, one ensemble is used to decide the digit 0 against all non-O's, another ensemble for the digit 1 against all non-1's,etc. We refer to the class which has all images of a particular digit class as class 0 and the other as class 1. In building the tree, we pick a feature (which is one of the 400 pixels in the 20x20 pixel array) that based on some criterion, best splits the examples into two subsets. Each of these two subsets will usually not contain examples of just one class, so we recursively divide the subsets until the final subsets each contain examples of just one class. Thus, each internal node specifies a feature and a value for that feature that determines whether one should take the left or right branch emanating from that node. At terminal nodes, we make the final decision, class 0 or 1. Thus, in decision trees one starts at a root node and progressively traverses the tree from the root node to one of the terminal nodes where a final decision is made. CART (Breiman, 1984) and C4.5 are perhaps the two most popular tree building algorithms.

The next sections discuss the theory behind our particular implementation of the boosting algorithm and decision trees. We describe experiments, do a comparison with some other methods of building tree ensembles and show how to build a multi-class ensemble. We then show how to combine the speed of the decision trees with tbe accuracy of the neural network.

## 2 BOOSTING

Boosting arises from the PAC (probably approximately correct) learning model which has as one of its primary interests the efficiency of learning. X is a set called the domain. 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 probability $1-\delta$ a hypothesis with error at most $\varepsilon$ where $\varepsilon, \delta > 0$. A *weak* PAC-learning algorithm is only required to to have an $\varepsilon \leq .5 - \gamma$. 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

"Weak" learner may be considered somewhat of a misnomer because it must attain an error rate less than .5 for any input distribution. In theory then, if we could find a weak learner that has an error rate less than .5 for any distribution of the input, we could have a arbitrarily low ensemble error rate. Practically, this does not happen. At some point, the learning machine can do no better than a 50% error rate and at that time we must terminate the process. Although strictly speaking the error rate is only required to be slightly less than .5, the algorithms converges faster if the error rate is much better that .5.

When dealing with the two-class case, the requirement that the error rate be slightly better than .5 is equivalent to doing slightly better than guessing. We still have the $.5-\gamma$ criterion even for the multi-class case which does not have the "random guessing" equivalency. In the N class case, random guessing is an (N-1)/N error rate. We must do much better than that, namely slightly better than .5. The intuition (although it can be proved) is this: in the multi-class case, the number of examples of each class in a particular filtered training set varies as we build weak learners. At some point, we may just have a preponderance of two (of N classes) present to train on. A requirement that we only do slightly better than an error rate of (N-1)/N means that the weak learner is doing an incredibly bad job and would not contribute to a reduction of the ensemble error rate.

We have have not been able to build weak learner multi-class decision trees because the weak learner error rates quickly reach .5 before the ensemble error decreases significantly. Therefore we build a ten sets of two-class ensembles as described previously. We have been able to build multi-class neural networks with the requisite performance (Drucker, 1993). but for our problem here where we want to build many weak learners and test them quickly, the slow training and evaluation times of large neural networks are prohibitively expensive.

Figure 3 details AdaBoost for the particular case where the weak learner is a classification tree. We call the set of $N_1$ distinct training examples the original training set. We distinguish the original training set from what we will call the filtered training set which consists of $N_1$ examples picked with replacement from the original training set. Sampling with replacement is sometimes called "bootstrap sampling" (Efron, 1993) and means that even if all samples are equally likely, the bootstrap sample may have multiple samples of some patterns and none of others. Each of the $N_1$ original training examples is assigned a weight which is proportional to the probability that the example will appear in the filtered training set (these weights have nothing to do with the weights usually associated with neural networks). Initially all examples are assigned a weight of unity so that all the examples are equally likely to show up in the initial set of training examples. However, the weights are altered at each state of boosting and if the weights are high we may have multiple copies of some of the original examples appearing in the filtered training set. In step three of this algorithm, we calculate what is called the weighted training error and this is the error rate over **all** the original $N_1$ training examples weighted by their current respective probabilities. Although not called for in the original C4.5 algorithm, we also have an original set of pruning examples which also are assigned weights to form a filtered pruning set and used to prone the classification trees constructed using the filtered training set. It is known (Mingers, 1989a) that reducing the size of the tree (pruning) improves generalization. The weights of the pruning and training examples are also obtained by step 5 of the algorithm but the values of of $\beta$ and $\varepsilon$ are obtained from the performance of the pruned tree on the filtered training set.

The algorithm terminates when the weak learner weighted error rate either is .5 or 0 (a highly singular event). When a zero error rate is achieved, this means that the weak learner has learned the difficult examples. Therefore, the weights will not change and the new bootstrap examples will have the same training set (statistically speaking) as the present training set. It is not absolutely

Inputs: $N_1$ training patterns, $N_2$ pruning patterns, $N_3$ test patterns

Initialize the weight vector of the $N_1$ training patterns: $w_i^1 = I$ for $i=1,...,N_1$
Initialize the weight vector of the $N_2$ pruning patterns: $s_i^1 = 1$ for $i=1,...,N_2$
Initialize the number of trees in the ensemble tot = 1

Do **Until** weighted training error rate is 0 or .5 or ensemble test error rate asymptotes

1. For the training set and pruning sets

$$p^t = \frac{w^t}{\sum_{i=1}^{N_1} w_i^t} \qquad r^t = \frac{s^t}{\sum_{i=1}^{N_2} s_i^t}$$

Pick $N_1$ samples from original training set with probability p(i) to form filtered training set.
Pick $N_2$ samples from original pruning set with probability r(i) to form filtered pruning set.

2. Train tree t using filtered training set and prune using filtered pruning set

3. Pass the $N_1$ original training examples through the pruned tree whose output $h_t(i)$ is either 0 or 1 and classification c(i) is either 0 or 1. Calculate the weighted training error rate: $\varepsilon_t = \sum_{i=1}^{N_1} p_i^t | h,(i) - c(i) |$

4. set $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$

5. Set the new training weight vector to be

$$w_i^{t+1} = w_i^t \{\beta_t ** (1 - | h,(i) - c(i) |)\} \qquad i = 1,...,N_1$$

Pass the $N_2$ original pruning patterns through the pruned tree and calculate new pruning weight vector:

$$s_i^{t+1} = s_i^t \{\beta_t ** (1 - I\, h,(i) - c(i) | )\} \qquad i = 1,...,N_2$$

6. For each tree t in the ensemble (total trees T) , pass the j'th test pattern through and obtain h,(i) for each t. The final hypothesis $h_f(j)$ for this pattern:

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

Do for each test pattern and calculate the ensemble test error rate:

7. t = t + 1

**End_Until**

Figure 3: Boosting Algorithm

necessary to track the test error rate (step 6 of Figure 3) and may be moved to outside the loop. However, by tracking the test error rate as one increases the number of weak learners we can stop the process when the test error rate asymptotes. (One should really call this a validation set rather than a test set.) Our experience. is that the ensemble never overtrains, that is, the test error rate never starts to increase as one increases the number of weak learners. The intuition for this is that as the weighted training error rate for the weak learner approaches .5, the value of $(1/\beta)$ becomes relatively small and has negligible influence on the overall summation.

## 3 DECISION TREES

We construct decision trees based on Quinlan's C4.5 algorithms (Quinlan, 1993) which uses an information theoretic criterion termed an information ratio (IR). However, as others have pointed out (Mingers, 1989b), the IR is not the only possible choice and many of the criteria end up giving similar performance. We initially search through all features and find both the feature and the values of the feature to split on. Based on this feature, the filtered training examples are split into two subsets. For each of these subsets, the feature with highest IR is chosen as the next decision node and we progress iteratively until we stop based on some criterion. In all cases, our experiments consisted of 10,000 original training examples, 2000 pruning examples and 2000 test examples.

The tree is initially constructed so that it has zero training error rate. Once the tree is constructed, it is pruned using the filtered pruning set to give (hopefully) better generalization performance than if the original tree was used. By better generalization, we mean that the performance is better on a separate test set than if no pruning had taken place. The other rationale for pruning other than better generalization is that the boosting algorithm would terminate if the weighted training error rate is zero. The original C4.5 algorithm uses the training set for what is called "pessimistic pruning" justified by the fact that there may not be enough extra examples to form a set of pruning examples. However, we prefer to use an independent set of examples to prune this tree whose statistics mirror that of the training set. In our case, we have (for each tree in the ensemble) an independent filtered pruning set of examples whose statistical distribution is similar to that of the filtered training set. Since the filtering imposed by the previous members of the ensemble can severely distort the original training distribution, we tmst this technique more than pessimistic pruning. In pruning (Mingers, 1989a), we pass the pruning set though the tree recording at each node how many errors there would be if the tree was terminated there. Then, for each node (except for terminal nodes), we examine the subtree of that node. We calculate the number of errors that would be obtained if that node would be made a terminal node and compare it to the number of errors at the terminal nodes of that subtree. If the number of errors at the root node of this subtree are less than or equal to that of the subtree, we replace the subtree with that node and make it a terminal node. Pruning tends to substantially reduce the size of the tree, even if the error rates are not substantially decreased.

## 4 EXPERIMENTS

To provide the data for this experiment we used a NIST (National Institute of Standards and Technolody) database of 120,000 digits. Each ensemble of decision trees will separate a particular digit (which we call class 0) from all the other digits (which we call class 1). Thus, there will be ten ensembles. To test the final set of ten ensembles, we held out 10,000 patterns. In building a particular ensemble, we construct a training set of size 10,000, half

from class 0 (a particular digit) and half from class 1 (equal number of samples from the other digits) picked randomly from the 110,000 remaining. To prune the tree, we pick a total of 2000 and to test that tree we have another set of 2000. To summarize: for each ensemble, we have training, pruning and testing sets for a total of 14,000 samples. After each stage of the boosting algorithm, and thus for each tree, we have filtered the original training and original pruning sets (but not the test set) to obtain a total of 10,000 filtered training and 2,000 filtered pruning samples by sampling with replacement from the original training and pruning sets.

TABLE 1. Test error rate in percent

| digit | single tree | boosted trees | number of trees |
|---|---|---|---|
| 0 | 4.15 | .70 | 23 |
| 1 | 3.55 | .65 | 14 |
| 2 | 8.00 | 1.15 | 35 |
| 3 | 11.00 | 2.00 | 41 |
| 4 | 6.70 | 1.50 | 28 |
| 5 | 7.00 | 2.10 | 10 |
| 6 | 4.05 | .90 | 18 |
| 7 | 6.05 | 1.10 | 21 |
| 8 | 8.60 | 3.20 | 17 |
| 9 | 6.15 | 2.95 | 15 |

Table I compares the error rates of the ten different ensembles against single trees. As expected, the ensemble error rates are much lower than the single trees. A fair question might be to ask if there is a single decision tree that gives better performance than this ensemble. However, we were unable to find such a tree. This is not to say that there is not some decision tree where perhaps the decisions at each node are nonlinear combinations of many attributes. However, then one should try to boost **that** tree.

We wanted to compare the boosting ensemble to other techniques for constructing ensembles. Our experience with neural networks is that one can train multiple neural networks on the same data and then combine the outputs of the neural networks. This is feasible because each neural network learns a different decision surface due to the fact that, depending on the weight initializations, each neural network reaches a different local minima of the optimization criterion. This is not true of trees. So (in an unfair comparison) we built trees, each trained on different sets of 14,000 examples picked at random from the 110,000 and combined them. To combine the outputs of these independently trained trees we built a linear classifier that was trained on the outputs of the trees (plus a bias term) using least mean squares minimization with targets of -1 (for class 0) and + 1 for class 1. The training data for the linear classifier is another random set of 2000 examples. In spite of the fact that we are using many more examples than needed to construct the boosting ensemble, the performance was worse than the boosting ensemble and after three trees, this technique reached asymptotic performance.

The problem with decision trees is that invariably, even if the training data is different (but drawn from the same distribution), the features chosen for the first few nodes are usually the same (at least for the OCR data). Thus, different decision surfaces are not created. In order to create different decision regions for each tree, we can force each decision tree to consider another attribute as the root node, perhaps choosing that attribute from the first few attributes with largest information ratio. This is similar to what Kwok and Carter (1990) have suggested but we have many more trees and their interactive approach did not look feasible here. Another technique suggested by T.K. Ho (1995) is to constmct independent trees on the same 10,000 examples by randomly striking out the use of some of the

100 possible features. Thus, for each tree, we randomly pick a different set of 50 features to construct the tree. When we use up to ten trees, the results using Ho's technique gives similar results to that of boosting but the asymptotic performance is far better for boosting. Other ensemble techniques may be found in Breiman (1994) , Casey (1983), Schlien (1990) and Oliver (1994). All these techniques basically depend on training on a different subsets of examples and somehow averaging the results. Boosting has similarities to all of these in that the different weak learners are trained on different subsets of the examples. However, in boosting the weak learners are forced to learn on progressively more difficult parts of the input space.

## 5 COMBINING CLASSIFIERS

We now have ten ensembles, separating a particular digit from the other digits. Suppose now, we input a character and want to determine the classification. The character is put through each of the ten ensembles and each of the ensemble outputs for pattern j: $\sum_{t=1}^{T}(\log\frac{1}{\beta_t})$ h,(j) is normalized by $\sum_{t=1}^{T}\log\frac{1}{\beta_t}$. Since the class 0 examples for each ensemble were a particular digit, we find the **smallest** normalized output and classify the input as belonging to the ensemble with the smallest normalized output. On the 10,000 examples that were held out for final testing, we have an error rate of 3.8%. much better than the 17% error rate obtained with ten single trees and 12% using the multi-class decision tree of the original C4.5 algorithm.

The ensemble of trees is much faster (1 msec) than a boosted neural network we have built (Drucker,1993), which has an error rate of .7% but an evaluation time of 210 msec (on a Sun SPARCstation 2). It is possible to combine the speed of the decision tree ensemble with the accuracy of the neural network in the following manner: Instead of using a decision tree to make all the decisions, we use the decision tree ensemble as a preclassifier, sending patterns to the neural network that we are unsure of. We have experimentally determined that the most important, measurement of the confidence of an ensemble in its classification is the (absolute) difference between the smallest output and the next smallest output (all outputs are positive). Therefore, one takes initially as the classification the ensemble with the smallest output. If the difference between this output and the next smallest output is large, then one can be highly confident of the classification. If very small, one is not as confident. We therefore want to determine a threshold of the difference of the two smallest outputs, above which we will accept the decision of the tree classifiers and below which we send the image to the neural network. This threshold is determined by sending a training set of size ten thousand through the ensembles and ranking the difference between the two smallest output ensembles. Start at the top of the list (with the largest differences) and run down the list, counting errors, until the error rate is .5% on the patterns counted so far. This is the required difference threshold. The remaining members of the list in this case constitutes 12% of the total training patterns and are the patterns we will reject and send to the neural network for further processing. Thus, if for a particular input, the difference of the two smallest normalized outputs (of the ten ensembles) is above this threshold, we accept the output of the classifier knowing that .5% of our classifications will be in error. Roughly 88% of the images will have this .5% error rate. Since the neural network is only used 12% of the time, the average recognition time is the neural network recognition time divided by a factor of approximately eight. The overall error rate, using both the neural network and the tree ensembles as just described is 0.9%.

## 6 CONCLUSIONS

We have shown how to build tree classifiers using a boosting algorithm. These are much more accurate than single trees. They are not as accurate as neural networks but are much faster, and therefore can be used as pre-classifiers for highly accurate recognition devices. One key to success is to have a separate pruning set whose statistics mirror that of the filtered training set.

## REFERENCES


L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik (1994), "Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition", 1994 International Conference on Pattern Recognition, Jerusalem

L. Breiman (1994) "Bagging Predictors", Technical Report No. 421, Department of Statistics, University of California, Berkeley, California 94720.

L. Breiman, J.Friedman, R.A.Olshen, and C.J.Stone (1984), *Classification and Regression Trees,* Chapman and Hall.

R.G. Casey and CR. Jih (1983), "A Processor-based OCR System", *IBM Journal of R&D,* vol *27, No. 4.* pp. 386-399.

H. Drucker, R.E. Schapire, and P. Simard (1993) "Boosting Performance in Neural Networks", *International Journal of Pattern Recognition and Artificial Intelligence,* Vol 7. No 4, pp. 705-719.

H. Drucker (1994), C. Cortes, LD Jackel, Y. LeCunn "Boosting and Other Ensemble Methods", Neural *Computation,* vol *6, no. 6,* pp. 1287-1299

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

Y. Freund (1990), "Boosting a Weak Learning Algorithm by Majority", *Proceedings of the Third Workshop on Computational Learning Theory,* Morgan-Kaufman", 202-216

Y. Freund and R.E. Schapire (1995), "A decision-theoretic generalization of on-line learning and a" application to boosting", *Proceeding of the Second European Conference on Computational Learning.*

T.K. Ho (1995), "Random Decision Forests", *Proceedings of the Third International Conference on Document Analysis and Recognition,* IEEE Computer Society Press, pp. 278-282.

S.W. Kwok and C.Carter (1990), "Multiple Decision Trees", *Uncertainty in Artificial Intelligence 4,* R.D. Shachter, T.S. Levitt, L.N. Kanal, J.F Lemmer (eds) Elsevier Science Publishers.

J. Oliver and D. Hand (1994), "Averaging Over Decision Stumps", *Proceedings European Conference on Machine Learning,* Springer-Verlag, pp. 231-241.

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

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

J.Mingers (1989b), "A" Empirical Comparison of Selection Measures for Decision-Tree Induction", *Machine Learning* 3:319-342.

R.E. Schapire (1990), The strength of weak learnability, *Machine Learning, X2):197-227.*

S. Shlien (1990), "Multiple Binary Decision Tree Classifiers", *Pattern* Recognition, vol. 23, No. 7, pp. 757-763.