Random Decision Forests

Tin Kam Ho
AT&T Bell Laboratories
600 Mountain Avenue, 2C–548C
Murray Hill, NJ 07974, USA

Abstract

Decision trees are attractive classifiers due to their high execution speed. But trees derived with traditional methods often cannot be grown to arbitrary complexity for possible loss of generalization accuracy on unseen data. The limitation on complexity usually means suboptimal accuracy on training data. Following the principles of stochastic modeling, we propose a method to construct tree-based classifiers whose capacity can be arbitrarily expanded for increases in accuracy for both training and unseen data. The essence of the method is to build multiple trees in randomly selected subspaces of the feature space. Trees in different subspaces generalize their classification in complementary ways, and their combined classification can be monotonically improved. The validity of the method is demonstrated through experiments on the recognition of handwritten digits.

1 Introduction

Decision-tree classifiers are attractive because of their many advantages – the idea is intuitively appealing, training is often straight-forward, and best of all, classification is extremely fast. They have been studied extensively in the past two decades and used heavily in practical applications. Prior studies include many tree construction methods [3] [14] [15] and, recently, relationship to other classifiers like HMM methods [8] and multi-layer perceptrons [12].

Many studies propose heuristics to construct a tree for optimal classification accuracy or to minimize its size. Yet trees constructed with fixed training data are prone to be overly adapted to the training data. Pruning back a fully-grown tree may increase generalization accuracy on unseen data, often at the expense of the accuracy on the training data. Probabilistic methods that allow descent through multiple branches with different confidence measures also do not guarantee optimization of the training set accuracy.

Apparently there is a fundamental limitation on the complexity of tree classifiers – they should not be grown too complex to overfit the training data. No method is known that can grow trees to arbitrary complexity, and increase both training and testing set accuracy at the same time.

Our study shows that this difficulty is not intrinsic to tree classifiers. In this paper we describe a method to overcome this apparent limitation. We will illustrate the ideas using oblique decision trees which are convenient for optimizing training set accuracy. We begin by describing oblique decision trees and their construction, and then present the method for increasing generalization accuracy through systematic creation and use of multiple trees. Afterwards, experimental results on handwritten digits are presented and discussed.

2 Oblique Decision Trees

Binary decision trees studied in prior literature often use a single feature at each nonterminal (decision) node. A test point is assigned to the left or right branch by its value of that feature. Geometrically this corresponds to assigning the point to one side of a hyperplane that is parallel to one axis of the feature space.

Oblique decision trees [5] are more general in that the hyperplanes are not necessarily parallel to any of the axes. Each hyperplane is represented by a linear function of the feature components. Using oblique hyperplanes usually yields a smaller tree that can fully split the data to leaves containing a single class. Sizes of the trees may differ drastically depending on how the hyperplanes are selected.

Most of the sophistication in tree growing algorithms is in the attempt to minimize the tree size, but there is little promise on the generalization accuracy. Instead of investigating these algorithms, we focus our attention on general methods for improving generalization accuracy. We therefore starts with two simple methods for tree construction, neither of which involves any sophisticated optimization procedure.

In either method the stopping rule is until all the terminal nodes (leaves) contain points of a single class, or until it is impossible to split further (this occurs in principle when identical samples exist across two or more classes, or in practice by limitations of the hyperplane search algorithm, e.g. a coarse quantization of the search space). Since we do not want to lose any accuracy on classifying the training data, we do not consider methods to prune back the tree.

0-8186-7128-9/95 $4.00 © 1995 IEEE

Central axis projection

The first method for tree growing finds a splitting hyperplane among those that are perpendicular to a line connecting two data clusters. It aims at separating at least two classes at each nonterminal node.

Assume that we have training points of two or more classes at any nonterminal node including the root. We first find the two classes whose means are farthest apart by Euclidean distance. The sample means of these two classes are then connected by a straight line (for convenience we call this line the central axis), and all the data points are projected onto this line. We then search along the line segment between the two means at a fixed step, and evaluate an error function at each hyperplane passing through the stop points and perpendicular to the line. For each class, the error function counts the number of points that are not on the side of the hyperplane where the majority of the points of that class fall. The hyperplane that minimizes the sum of these counts is chosen for that node.

The procedure runs in one pass and is very fast. Other than minor difficulties caused by the size of the search steps or identical samples, it always stops when all leaves contain a single class. The drawback is that the crude optimization often leads to a very large tree.

Perceptron training

The second method uses the fixed-increment perceptron training algorithm to choose the hyperplane at each nonterminal node.

As in the first method, at each nonterminal node, the two classes that are farthest apart are found and their means are chosen as anchor points. Two sets S_1 and S_2 are initialized each containing one of these two classes. The other classes are then assigned to either S_1 or S_2 depending on which anchor point is closer to their means by Euclidean distance. The method then searches for a hyperplane to separate S_1 and S_2, minimizing the number of points on the wrong side of the hyperplane.

The algorithm terminates when the error count decreases to zero. However, since there is no test on the linear separability of S_1 and S_2, there is no guarantee on the convergence of the algorithm. Therefore the algorithm is also forced to terminate after a fixed number of iterations. In the case that the perceptron thus obtained does not separate points of any class from the rest, the method continues by switching to central axis projection.

Although the formation of the two sets is somewhat arbitrary, the iterative optimization does lead to a substantial reduction of tree sizes. Training is more expensive, but the smaller trees take less storage and are faster for classifying new samples.

3 Systematic Creation of Multiple Trees

Both tree-growing methods are able to grow complex trees that perfectly classify the training data. Yet because of the biases of the particular ways in which the hyperplanes are chosen, the generalization accuracy is rarely as good. Retreating to non-fully split trees would mean losing 100% accuracy on training data, which hardly provides any confidence on doing well on unseen test data.

Past experience in other contexts [6] has shown that the use of multiple classifiers can often compensate for the bias of a single classifier. It turns out that the same methodology is applicable here. We will look into using multiple trees – i.e., a forest – to overcome the generalization biases.

To be successful with multiple trees, we need a way to create trees that generalize independently. We also need a discriminant function that combines the classification given by the individual trees and preserves their accuracies.

How can we systematically create multiple decision trees using the same set of data? There are many ways to construct different trees, but an arbitrarily introduced difference does not necessarily give trees we need – trees that are 100% accurate on training data and yet have different generalization errors. For instance, consider building trees using different subsets of the training data. Those trees may not classify the full training set perfectly.

Randomization has been a powerful tool for introducing differences in classifiers. Previously it has been used to initialize training algorithms with different configurations that eventually yield different classifiers [4] [7].

Our method to create multiple trees is to construct trees in randomly selected subspaces of the feature space. For a given feature space of m dimensions, there are 2^m subspaces in which a decision tree can be constructed. The use of randomization in selecting components of the feature vector is merely a convenient way to explore the possibilities.

A decision tree is constructed in each selected subspace using the entire training set and the algorithms given in the previous section. Notice that each of these trees classifies the training data 100% correctly. Yet the classification is invariant for points that are different from the training points only in the unselected dimensions. Thus each tree generalizes its classification in a different way. The vast number of subspaces in high dimensional feature spaces provides more choices than can be used in practice.

There are many interesting theoretical questions following this idea. How many of the subspaces must we use before we can achieve a certain accuracy with the combined classification? What will happen if we use all the possible subspaces? How do the results differ if we restrict ourselves to subspaces with certain properties?

Some of these questions are addressed in the theory of stochastic modeling, where the combination of various ways to partition the feature spaces is studied [1] [9] [10] [11]. In the theory, classification accuracies are related to the statistical properties of the combination function, and it is shown that very high accuracies can

be achieved far before all the possible combinations are used.

4 The Discriminant Function

Given t trees created in random subspaces, a discriminant function is needed to combine their classification of a test point. Here we use the combination function proposed in [7].

For a point x, let $v_j(x)$ be the terminal node that x is assigned to when it descends down tree T_j $(j = 1, 2, ..., t)$. Given this, let the posterior probability that x belongs to class c $(c = 1, 2, ..., n)$ be denoted by $P(c|v_j(x))$.

$$P(c|v_j(x)) = \frac{P(c, v_j(x))}{\sum_{l=1}^{n} P(c_l, v_j(x))}$$

can be estimated by the fraction of class c points over all points that are assigned to $v_j(x)$. Notice that in this context, since the trees are fully split, most terminal nodes contain only a single class (except for abnormal stops) and thus the value of the estimate $\hat{P}(c|v_j(x))$ is almost always 1.

The discriminant function is defined as

$$g_c(x) = \frac{1}{t} \sum_{j=1}^{t} \hat{P}(c|v_j(x))$$

and the decision rule is to assign x to class c for which $g_c(x)$ is the maximum.

It is obvious that the discriminant preserves 100% accuracy on the training set. For an unseen point, $g(x)$ averages over the posterior probabilities that are conditioned on reaching a particular terminal node. Geometrically, each terminal node defines a neighborhood around the points assigned to that node in the chosen subspace. By averaging over the posterior probabilities in these neighborhoods (decision regions), the discriminant approximates the posterior probability for a given x in the original feature space. This is similar to other kernel-based techniques for estimating posterior probabilities, except that here the kernels are of irregular shapes and sizes.

In [7] the discriminant is used to combine multiple classifiers trained by learning vector quantization, and it is shown experimentally that the accuracy improves with increases in t. The analytical properties of the function and its several variants have been studied extensively by Berlind [1].

5 Experiments with Handwritten Digits

We now show the effectiveness of the method with experiments in a difficult recognition problem – the recognition of isolated handwritten digits. Notice that neither the algorithm nor the features have been tuned specifically for the data – similar experiments have been performed on machine-printed symbols, with essentially the same findings. The method is most effective for problems involving high dimensional data because of the existence of more subspaces.

The data

The experiments were performed on handwritten digits of 10 classes. The images are from the 1992 NIST (National Institute of Standards and Technology) Competition (for details see [2]). The training and testing sets from the competition are mixed, and from the mixed set 60,000 samples are drawn to form the training set TR, and 10,000 samples are drawn to form the test set TE. The images are binary and normalized to a size of 20×20 pixels. There are roughly the same number of samples in each class in both the training and testing sets.

The features

We first use the raw pixel maps of the binary, normalized images as input feature vectors. For convenience we call this the pixel vector (f_1) which has $20 \times 20 = 400$ components. To see how simple knowledge of the domain can help, another feature vector is constructed by exploring the neighbor relationships of the pixels. These features are similar to those used in constructing quadratic polynomial discriminants [13]. They are conjunctions and disjunctions of neighboring pixels in various directions. For a given pixel $I(i, j)$ at row i and column j, we take

$H(i,j) = I(i,j) \wedge I(i,j+2)$	horizontal neighbor
$V(i,j) = I(i,j) \wedge I(i+2,j)$	vertical neighbor
$N(i,j) = I(i,j) \wedge I(i+2,j+2)$	NW-SE diagonal neighbor
$S(i,j) = I(i,j) \wedge I(i+2,j-2)$	SW-NE diagonal neighbor

and

$$H'(i,j) = H(i,j) \vee H(i+1,j) \vee H(i+2,j) \vee H(i+3,j)$$
$$V'(i,j) = V(i,j) \vee V(i,j+1) \vee V(i,j+2) \vee V(i,j+3)$$
$$N'(i,j) = N(i,j) \vee N(i+1,j-1) \vee N(i+2,j-2) \vee N(i+3,j-3)$$
$$S'(i,j) = S(i,j) \vee S(i+1,j+1) \vee S(i+2,j+2) \vee S(i+3,j+3),$$

where $\wedge$ is the binary AND and $\vee$ is the binary OR operations. The second vector f_2 is formed by concatenating f_1 with the well-defined values of $H'(i,j), V'(i,j), N'(i,j)$, and $S'(i,j)$ for each (i,j). For an image of 20×20 pixels, f_2 has 852 components.

Results with single trees

First we show the results when single trees are constructed in the full feature space, as in the conventional practice. We tested both vectors f_1 and f_2 and both of the tree growing methods.

Using central axis projection (abbreviated as CAP) and f_1, a tree of 3949 nodes (including both terminal and nonterminal nodes) was obtained. When f_2 was used, the tree is slightly smaller (3255 nodes). Using the perceptron training algorithm (abbreviated as PER) and f_1, a tree of only 307 nodes was obtained. When f_2 was used, the resultant tree has only 87 nodes. These are the results of the interaction between the complexity of the distributions and the algorithm for deriving the discriminating hyperplanes. These dramatic differences in the sizes of the trees show the importance in the choice of tree-growing heuristics on classification efficiency. Table 1 shows the number of

Table 1: Number of Terminal Nodes and Classification Accuracies for Each Class

	CAP(f_1)		CAP(f_2)		PER(f_1)		PER(f_2)	
class	#nodes	%corr	#nodes	%corr	#nodes	%corr	#nodes	%corr
0	108	95.24	93	96.54	9	91.77	2	96.21
1	87	98.27	80	98.45	12	97.32	3	98.53
2	226	87.13	167	91.94	18	83.30	5	93.22
3	247	88.88	212	90.90	17	85.24	6	92.72
4	183	89.99	139	90.20	14	83.92	5	92.65
5	251	86.07	193	90.25	15	78.67	3	92.39
6	121	91.09	110	89.72	12	87.07	4	93.14
7	185	89.95	161	90.34	16	87.98	5	93.10
8	288	83.01	234	84.57	22	79.39	6	89.36
9	279	84.98	239	87.44	19	83.55	5	91.22
all	1975	89.57	1628	91.11	154	86.01	44	93.32

terminal nodes needed to represent each class and the corresponding classification accuracy on TE.

It is interesting to see that despite the differences in tree sizes, the two tree-growing algorithms do not differ by large in classification accuracy. An inefficient growing algorithm may cause duplication of similar structures along different branches, whose effect is unpredictable and is dependent on the class distributions in the feature space.

Yet the degradation in accuracy as a class is distributed to more terminal nodes is far more obvious within the same tree. For instance, classes 0 and 1 are consistently more likely to be correct than other classes, and there are consistently fewer terminal nodes representing 0's and 1's in each of the trees. Recall that each of these trees classifies correctly all training points in TR, the poor performance for some classes on TE is no doubt a result of poor generalization. Generalization is understandably worse when more terminal nodes, i.e., more tailored hyperplanes are used to fit a class.

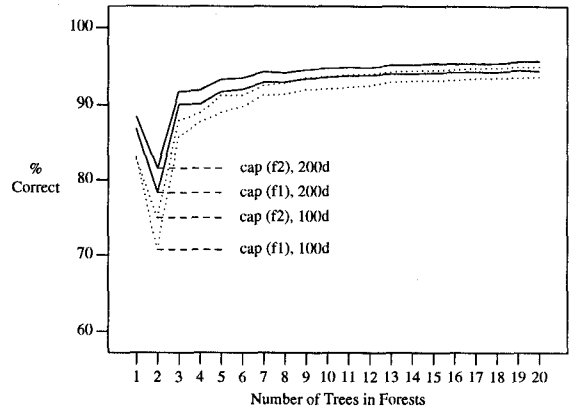

Figure 1: Classification Accuracy (% correct) of Forests Constructed by Central Axis Projection (in 100- and 200- dimensional random subspaces)

Results with multiple trees

We now see how the problem of poor generalization can be overcome by the use of multiple trees. Again we tested the idea with both feature vectors f_1 and f_2 and both tree-growing methods. The results are shown in Figures 1 and 2.

We show the changes in overall classification correct rate as new trees were added to the classifier. Each of the additional trees was constructed in a random feature subspace that had not been used. The subspaces were restricted to 100 or 200 dimensions in the experiments, and the resultant differences in classification accuracy are clear from the figures.

Since training time is substantially shorter with central axis projection, we could easily afford to let the algorithm continue until 20 trees were created. With perceptron training we stopped the algorithm at 10 trees. Note that with sufficient resources the algorithm could have continued to create many more trees, and the stopping points we had chosen were arbitrary.

In each graph the dip in the correct rate when two

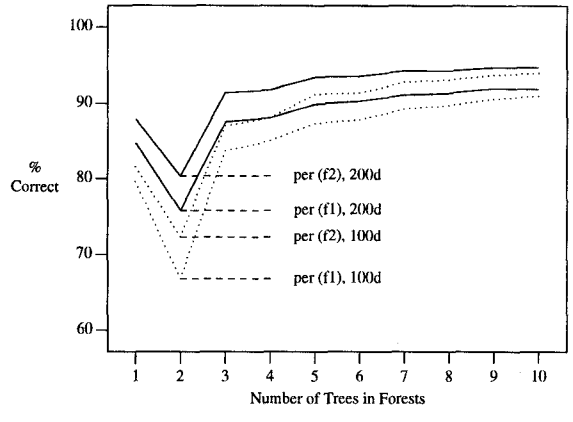

Figure 2: Classification Accuracy (% correct) of Forests Constructed by Perceptron Training Algorithm (in 100- and 200- dimensional random subspaces)

trees are used is due to ambiguities of the combined decisions (two classes sharing the same $g_c(x)$ value and are both rejected). This can be avoided if we adjust the probability estimates using a secondary training set.

From the graphs it is very clear that accuracy increases with the addition of new trees, and the increase is nearly monotonic. The trend is the same despite many differences in the details of the runs – different feature vectors, different numbers of subspace dimensions, and different tree-growing algorithms. The increase slows down as the forests grow, but there has not been any sign that there exists any upper limit below the highest possible. In one test we continued the run until 40 trees were created, and the increase in correct rate still did not stop. This is in sharp contrast with many other classifier designs, where an increase in classifier complexity almost always leads to overtraining. We have shown a way to increase classifier complexity (similarly, its capacity) without trading off generalization accuracy.

Comparing the results with different feature vectors and different classes, it is very promising that with certain engineering effort (better feature designs, better choices of training data), an excellent classifier can be created. Furthermore, the classifier is arbitrarily improvable to suit accuracy demands and resource limits.

Considering the large number of possible subspaces, our scope of exploration in the experiments has been quite limited. For instance, for implementational convenience we restricted ourselves to combinations of trees in random subspaces of the same number of dimensions, which is by no means necessary. In fact, the choice of a subspace need not be the same for all nonterminal nodes of a tree. The method is also not constrained to any particular tree-growing algorithm, nor even the binary tree structure, and there is still room for exploration with other sophisticated tree-growing algorithms and various tree structures.

6 Conclusions

We have proposed a method for increasing generalization accuracies of decision tree-based classifiers without trading away accuracy on training data. Experiments on handwritten digits proved the validity of the idea and indicated many opportunities for further improvements.

The method we have presented here is another variant of the methodology of stochastic modeling that has been studied in both theory and experimentation recently [1] [7] [9] [10] [11]. The current method is in closest resemblance with the method of multiple LVQ-based classifiers studied in [7]. Both methods involve creating (with certain application of randomization) and combining multiple ways of partitioning the feature space. The decision regions determined by these partitions are *stochastic models*, the subject of study in [1] and [10]. One of the main conclusions from the theory is that the apparent conflict between optimizing training set accuracy and maintaining generalization accuracy is not intrinsic. We have given another piece of empirical evidence of this finding. Our experimentation has also revealed great potential of applying the concepts of stochastic modeling to conventional methods for classifier design.

Acknowledgements

I would like to thank Eugene Kleinberg and Roger Berlind for sharing with me their work on stochastic modeling; Henry Baird, David Ittner, Dz-Mou Jung, George Nagy for stimulating discussions; and Yann le Cun for providing the NIST data.

References

[1] R. Berlind, *An Alternative Method of Stochastic Discrimination with Applications to Pattern Recognition*, Doctoral Dissertation, Department of Mathematics, SUNY at Buffalo, 1994.

[2] L. Bottou, et al., Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition, *Proceedings of the 12th International Conference on Pattern Recognition*, **II**, Jerusalem, Israel, Oct 9-13, 1994, 77-82.

[3] L. Breiman, J.H. Freidmen, R.A. Olsen, C.J. Stone, *Classification and Regression Trees*, Wadsworth, 1984.

[4] L.K. Hansen, P. Salamon, Neural Network Ensembles, *IEEE Transaction of Pattern Analysis and Machine Intelligence*, **PAMI-12**, 10, October 1990, 993-1001.

[5] D. Heath, S. Kasif, S. Salzberg, Induction of Oblique Decision Trees, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, **2**, Chambery, France, Aug 28-Sep 3, 1993, 1002-1007.

[6] T.K. Ho, *A Theory of Multiple Classifier Systems And Its Application to Visual Word Recognition*, Doctoral Dissertation, Department of Computer Science, SUNY at Buffalo, 1992.

[7] T.K. Ho, Recognition of Handwritten Digits by Combining Independent Learning Vector Quantizations, *Proceedings of the Second International Conference on Document Analysis and Recognition*, Tsukuba Science City, Japan, October 20-22, 1993, 818-821.

[8] M.I. Jordan, R.A. Jacobs, Hierarchical Mixtures of Experts and the EM Algorithm, *A.I. Memo* No. 1440, *C.B.C.L. Memo* No. 83, MIT Artificial Intelligence Laboratory, Center for Biological and Computational Learning, and Department of Brain and Cognitive Sciences, August 6, 1993.

[9] E.M. Kleinberg, Stochastic Discrimination, *Annals of Mathematics and Artificial Intelligence*, **1**, 1990, 207-239.

[10] E.M. Kleinberg, An Overtraining-Resistant Stochastic Modeling Method for Pattern Recognition, to appear.

[11] E.M. Kleinberg, T.K. Ho, Pattern Recognition by Stochastic Modeling, *Proceedings of the Third International Workshop on Frontiers in Handwriting Recognition*, Buffalo, May 1993, 175-183.

[12] Y. Park, A Comparison of Neural Net Classifiers and Linear Tree Classifiers: Their Similarities and Differences, *Pattern Recognition*, **27**, 11, 1994, 1493-1503.

[13] J. Schuermann, A Multifont Word Recognition System for Postal Address Reading, *IEEE Transactions on Computers*, **C-27**, 8, Aug 1978, 721-732.

[14] J. Schuermann, W. Doster, A Decision Theoretic Approach to Hierarchical Classifier Design, *Pattern Recognition*, **17**, 3, 1984, 359-369.

[15] I.K. Sethi, G.P.R. Sarvarayudu, Hierarchical Classifier Design Using Mutual Information, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-4**, 4, July 1982, 441-445.