Classification Trees with Neural Network Feature Extraction

Heng Guo† and Saul B. Gelfand††

† CIC Corporation, 275 Shoreline Dr., Redwood Shores, CA 94065
††School of Electrical Engineering, Purdue Univ., W. Lafayette, IN 47907

Abstract

We propose the idea of using small multilayer nets at the decision nodes of a binary classification tree to extract nonlinear features. This approach exploits the power of tree classifiers to use appropriate local features at the different levels and nodes of the tree. The nets are trained and the tree is grown using a gradient-type learning algorithm in conjunction with a heuristic class aggregation algorithm. The proposed method improves on standard classification tree design methods [1] in that it generally produces trees with lower error rates and fewer nodes. It also provides a structured approach to neural network classifier design which reduces the problem associated with training large unstructured nets, and transfers the problem of selecting the size of the net to the simpler problem of finding a right-sized tree.

1. Introduction

Recently, some interesting connections and comparisons between classification trees and neural networks have been investigated [2], [3], [4]. In this paper we investigate how to combine classification trees and neural networks in an effective manner.

A binary classification tree is shown in Figure 1.1. The tree classifies an input pattern vector $\mathbf{x}$ through a chain of binary decisions. Starting at the root node and proceeding down the tree, tests of the form $f(\mathbf{x}) < \theta$ are conducted to determine whether the pattern goes to the left or right descendent. Each such test is called a splitting rule, and $f(\mathbf{x})$ and θ are the associated feature and threshold, respectively. The pattern is assigned the class label $\hat{c}$ of the terminal node it lands in.

Given a data set consisting of pattern vectors and class labels, most approaches to classification tree design determine the splitting rules in a stepwise top-down fashion. The family of features from which the splitting rule is selected is known to have a very significant effect on performance. In the literature two different families of features have typically been used: the coordinates of the pattern vector and linear combinations of the coordinates of the pattern vector [1], [5], [6].

Difficult pattern recognition problems with complex decision boundaries may require nonlinear features. Such features would further reduce the myopic effect of the stepwise splitting procedure. In this paper we present a method for extracting a nonlinear feature at each decision node using a small multilayer perceptron.

The paper is organized as follows. In Section 2 we describe necessary notation and briefly review some details of the CART (Classification and Regression Trees) method for constructing classification trees [1]. The CART method will serve as a baseline for development of and comparison with the proposed method. In Section 3 we discuss how to incorporate a small multilayer perceptron at each decision node of a classification tree to extract a nonlinear feature. Here we examine how to select and train the nets, and grow and prune the tree. In Section 4 we present experimental results on a waveform recognition problem and a handwritten character recognition problem. More details of the proposed method and experimental results may be found in [10].

2. Classification Tree Notation and the CART Method

In this section we describe necessary notation and briefly outline the CART method for constructing classification trees [1]. We start by giving some notation for binary trees.

We denote by T a binary tree. We also write $t \in T$ to denote a node t of the tree T, and $S \leq T$ to denote a

0-8186-2855-3/92 $3.00 ©1992 IEEE

pruned subtree S of the tree T (a pruned subtree is a subtree which shares the same root node as the original tree). Let root(T) be the root node of T, and $\tilde{T}$ be the terminal nodes of T. For each nonterminal node $t \in T - \tilde{T}$, let t_L and t_R be the left and right child nodes of t, respectively.

We associate a decision rule with a tree (which we then call a classification tree) by specifying a splitting rule at each nonterminal node and a class label at each terminal node, as described in Section 1. Furthermore, if we assign a class label to each node (terminal and nonterminal alike) then every pruned subtree is a classification tree. We next discuss the two phases of classification tree construction in CART, i.e., tree growing and tree pruning.

2.1 Tree Growing

In the tree growing phase of CART a large tree T is grown by recursively finding splitting rules (features and thresholds) until all terminal nodes have pure or nearly pure class membership or cannot be split further. The feature and threshold at a node are determined by minimizing a splitting criterion. Let N be the number of training samples, N(t) be the number of training samples which land in node t, $N_j(t)$ be the number of training samples which land in node t and belong to class j, and M be the number of classes. Define

$$p(t) = \frac{N(t)}{N}, \quad p_L(t) = \frac{p(t_L)}{p(t)}, \quad p_R(t) = \frac{p(t_R)}{p(t)},$$

and

$$p(j \mid t) = \frac{N_j(t)}{N(t)}.$$

Observe that p(t) is the probability that a randomly selected training sample lands in node t, $p_L(t)$ ($p_R(t)$) is the conditional probability that the training samples lands in node t_L (t_R) given it lands in node t, and $p(j \mid t)$ is the conditional probability that the training sample belongs to class j given it lands in node t. In CART the splitting criterion is based on a node impurity function such as the Gini criterion

$$i(t) = \sum_j \sum_{i \neq j} p(i \mid t)p(j \mid t)$$

For a given node impurity function, i(t), define the decrease in the node impurity $\Delta i(f,\theta,t)$ due to a split at node t with feature f and threshold θ by

$$\Delta i(f,\theta,t) = i(t) - i(t_L)p_L(t) - i(t_R)p_R(t). \qquad (2.1)$$

The best feature f* and threshold θ^* at node t are obtained by maximizing the decrease in node impurity

$$\Delta i(f^*,\theta^*,t) = \max_{f \in F,\theta} \Delta i(f,\theta,t)$$

where F is the allowed feature set (F could be allowed to depend on the node t). The allowed feature set F in CART is either the coordinate functions $f(\mathbf{x}) = x_i$ or the linear functions $f(\mathbf{x}) = a_1x_1 + \cdots + a_px_p$.

2.2 Tree Pruning

In the tree pruning phase of CART the large tree T is pruned back to avoid overfitting the training data. A pruned subtree is selected by minimizing an error rate estimate over a parametric family of pruned subtrees. The family of pruned subtrees is generated as follows. Suppose each node in T is assigned a class label based on majority vote. Then the resubstitution estimate of the probability of error for a pruned subtree classifier S can be expressed as

$$R(S) = \sum_{t \in \tilde{S}} R(t),$$

where

$$R(t) = r(t)p(t),$$

$$r(t) = 1 - \max_j p(j \mid t),$$

and p(t), $p(j \mid t)$ are defined as above. Now define the error-complexity of the pruned subtree classifier S by

$$R_\alpha(S) = R(S) + \alpha|\tilde{S}|$$

where $\alpha \geq 0$ and $|\tilde{S}|$ is the number of terminal nodes in S. The desired family of pruned subtrees $T(\alpha)$, $\alpha \geq 0$ is obtained by minimizing the error-complexity criterion

$$R_\alpha(T(\alpha)) = \min_{S \leq T} R_\alpha(S)$$

for each fixed value of α. Note that T(0) = T and $T(\alpha)$ = root(T) for α large enough. An algorithm for generating the $T(\alpha)$, $\alpha \geq 0$, is given in [1, p. 294].

From the family of pruned subtrees $T(\alpha)$, $\alpha \geq 0$, a pruned subtree $T(\alpha^*)$ is selected by minimizing an honest estimate of the probability of error

$$\hat{R}(T(\alpha^*)) = \min_\alpha \hat{R}(T(\alpha)).$$

CART can use several methods to obtain honest estimates of the error rate. The simplest approach is to use an independent test set. More sophisticated approaches which make more efficient use of the training data use cross validation or bootstrap estimates. See [1], [9] for details.

3. Tree Classifiers with Neural Network Feature Extraction

In this section we discuss the idea of using a small multilayer perceptron at each node of the classification tree to extract a nonlinear feature. The general flow of our algorithm for constructing a classification tree with a multilayer perceptron at each decision node is similar to the CART algorithm described in Section 2 and consists of tree growing and tree pruning phases.

3.1 Tree Growing

The particular type of neural net used to extract a nonlinear feature at each decision node is multilayer perceptron [7]. Let $\mathbf{x}$ denote the net input vector and y denote the net output. The feature $y = f(\mathbf{x})$ generated by such a multilayer perceptron at decision node t is used in the following manner. If $y < 0$ then $\mathbf{x}$ is directed to node t_L; if $y \geq 0$ then $\mathbf{x}$ is directed to node t_R. Note that we set the splitting rule threshold $\theta = 0$ here because any splitting rule with $\theta \neq 0$ is equivalent to some splitting rule with $\theta = 0$ and an appropriate choice of the output node threshold.

We choose to use the backpropagation algorithm [8] as a basis for training the multilayer perceptrons. Let $(\mathbf{x}(n),d(n))$, $n = 1,...,N$, denote a training sequence of input vectors and desired outputs, and let $y(n) = f(\mathbf{x}(n))$. Backpropagation is a stochastic gradient algorithm for minimizing the least squares error criterion

$$E = \frac{1}{N}\sum_{n=1}^{N}(d(n) - y(n))^2$$

over the weights and threshold parameters of the net. Now let $(\mathbf{x}(n),c(n))$, $n = 1,...,N$, denote the training sample of pattern vectors and their class labels at a given node. The backpropagation algorithm can be used to train a single output multilayer perceptron to discriminate between two classes or disjoint groups of classes by assigning the desired output d corresponding to a class label c as follows:

$$d = -1 \text{ if } c \in C_L$$

$$= +1 \text{ if } c \in C_R,$$

where C_L, C_R is a partition of the classes into two groups. In the case of three or more classes we need, of course, to select an appropriate partition.

Our algorithm for training the multilayer perceptron at each node of the binary decision tree proceeds as follows. The training involves two nested optimization problems. In the inner optimization problem the backpropagation algorithm is used to minimize the above least-squares criterion over the weight and threshold parameters of the net for a given pair of aggregate classes and so find a good split between that pair of aggregate classes. In the outer optimization problem a heuristic search is used to minimize a Gini impurity criterion over the pairs of aggregate classes and so find a good pair of aggregate classes and hence a good overall split.

Training Algorithm for Multilayer Neural Network at Decision Node

Let t be a decision node and $C = \{\omega_1, \ldots, \omega_M\}$ be the set of M classes at node t.

Step 1 Select an initial partition C_L, C_R of C.

Step 2 Train the net at node t with the backpropagation algorithm to separate C_L and C_R. Suppose $f(\bullet)$ is the input-output mapping of the trained net. For each training sample $\mathbf{x}$, direct $\mathbf{x}$ to t_L if $f(\mathbf{x}) \leq 0$ and to t_R if $f(\mathbf{x}) > 0$. Compute the decrease in the Gini impurity criterion as defined in (2.1) and denote it by Δi_0.

Step 3 For $m = 1,...,M$ form the partition $C_L(m), C_R(m)$ of C by changing the assignment of class ω_m in the partition C_L, C_R; train the net with the backpropagation algorithm to separate $C_L(m)$ and $C_R(m)$; and compute the decrease in the Gini impurity criterion as defined in (2.1) and denote it by Δi_m.

Step 4 Let $\Delta i_{m^*} = \max_{1 \leq m \leq M} \Delta i_m$. If $\Delta i_{m^*} \leq \Delta i_0$ then exit. Otherwise set $C_L = C_L(m^*)$ and $C_R = C_R(m^*)$ and go to *Step 2*.

Some further implementation details on selecting the size of the nets, training the nets and growing the tree are discussed in Section 4.

3.2 Tree Pruning

The CART tree pruning method (Section 2.2) can be used to prune back the large tree T whether or not it has multilayer perceptrons at its decision nodes for nonlinear feature extraction. Recall that in the CART method a pruned subtree is selected by minimizing an honest error rate estimate over a parametric family of pruned subtrees

$$\hat{R}(T(\alpha^*)) = \min_{\alpha} \hat{R}(T(\alpha))$$

We propose an alternative method where a pruned subtree is selected by minimizing an honest error rate estimate over *all* pruned subtrees

$$\hat{R}(T^*) = \min_{S \leq T} \hat{R}(S)$$

We expect that our pruning methodology will work better than the CART methodology (i.e., produce lower error rates) because the CART method only partially parameterizes the set of pruned subtrees of T. An efficient algorithm for generating T* is given in [9].

The proposed pruning algorithm (see [9]) is essentially a bottom-up algorithm, in that it starts from the terminal nodes and proceeds up the tree, pruning away branches. Only one pass through the tree is required to generate T*. In contrast, the CART pruning algorithm (see [1]) is essentially a top-down algorithm, in that it starts from the root node and proceeds down the tree, pruning away branches. Multiple passes through the tree are required to generate the family $T(\alpha)$, $\alpha \geq 0$, and hence $T(\alpha^*)$.

Recall that the CART tree pruning method requires honest estimates of the error rates $\hat{R}(T(\alpha))$ for $\alpha \geq 0$. These estimates can be obtained from an independent test set or by cross validation or bootstrap. The proposed tree pruning method requires honest estimates of the error rates $\hat{R}(S)$ for $S \leq T$. These estimates can also be obtained from an independent test set, but it is not clear how to obtain them by cross validation or bootstrap, essentially because the family of pruned subtrees we consider is not parameterized. A more sophisticated approach which uses the above pruning methodology but makes more efficient use of the training data can be found in [9].

4. Experimental Results

In this section we discuss some implementation issues for our proposed classification tree with neural network feature extraction (CTNNFE) method, and present two numerical examples.

The CART and CTNNFE tree growing methods were implemented as described in Sections 2 and 3. Tree pruning was implemented with the optimal pruning methodology described in Section 3.2 in all cases. The point here is to isolate the effect of the small multilayer perceptrons.

A number of parameters affect the implementation and performance of our CTNNFE method, which we now discuss.

a) The number of training samples at a decision node decreases as one proceeds down the tree. Smaller nets should be used at decision nodes with fewer training samples as this avoids overfitting the data. To control the size of the net we adopted a scheme where the number of hidden neurons in the multilayer perceptron is chosen such that the number of parameters is not more than half the number of training samples and satisfies a strict upper bound. The same bound was used at all nodes in the tree.

b) The number of sweeps (complete pattern presentations) in the back propagation algorithm must be specified. To control the amount of training we adopted a scheme where training was stopped when the change in the square error from one sweep to the next fell below a threshold. The same threshold was used at all nodes in the tree.

c) The step size in the backpropagation algorithm must also be specified. We used the same step size at all decision nodes in the tree.

4.1 Waveform Recognition

The first example we consider is a synthetic waveform recognition problem from [1]. It is a three-class problem based on the waveforms $h_1(t)$, $h_2(t)$ and $h_3(t)$ shown in Figure 4.1. Each class is a uniform random convex combination of two of these waveforms. The pattern vector is obtained by sampling at 21 points and adding Gaussian noise with zero mean and unit variance. The three classes have equal prior probabilities. According to [1], the Bayes misclassification rate for this problem is approximately .14.

Classification trees were grown and pruned with equal number of independent samples, ranging in size from 250 to 2000 samples. An additional set of 5,000 independent samples was employed to obtain highly accurate estimates of the error rate. For the CTNNFE method the nets at the decision nodes were allowed at most eight hidden neurons, and a step size of .1 and a termination threshold of .01 was used. Since there are only three classes in this problem, class aggregation at decision nodes was performed by exhaustive search.

The numerical results for the different tree design methodologies are presented in Figures 4.2-4.5. Analogous results for a 2 hidden-layer feedforward net are also presented in the figures. The net has 3 output neurons, 5 neurons in the first hidden layer, 20 neurons in the second hidden layer and 21 inputs.

4.2 Handwritten Character Recognition

The second example we consider is a real handwritten character recognition problem. It is a 10 class problem to recognize the numerals 0 - 9. 800 handwritten characters were collected from eight individuals with the same number of samples for each numeral. A digitized binary image of 128x128 pixels is generated for each handwritten numeral. The binary

image is then encoded to form the pattern vector. A sixteen dimensional pattern vector is used for this example, consisting of the row and column sums of the binary image partitioned into eight rows and columns.

Classification trees were constructed identically to the waveform recognition example except that class aggregation at decision nodes was performed using heuristic search as described in Section 3.1. Also, the error rates were estimated through an 8-fold cross validation procedure instead of using a large independent test sample. Each of the eight auxiliary trees was grown with 500 samples, pruned with 200 samples and tested on the remaining 100 samples. The numerical results are given in Table 4.1. Analogous results for a 2-hidden layer feedforward net are also shown in the table. The net has 10 output nodes, 10 nodes in the first hidden layer, 32 nodes in the second hidden layer and 16 inputs. Note that our CTNNFE method is more robust to the selection of the step size μ than is the large multilayer net trained with the backpropagation algorithm.

5. Conclusions

We have developed a new approach to the design of classification trees for pattern recognition. It distinguishes itself from standard classification tree design methods in two major respects: it extracts local nonlinear features at the decision nodes using small multilayer neural networks, and it uses a gradient-type learning algorithm in conjunction with a heuristic class aggregation algorithm for extracting the local features.

The proposed method was evaluated through two applications and was shown to yield superior performance compared to the CART classification tree design method and also a large multilayer net trained with the backpropagation algorithm. Compared to CART the proposed method yielded significantly lower error rates and smaller trees. Compared to the large multilayer net the proposed method yielded comparable error rates and shorter training times, and the training procedure was much more automatic and robust.

Acknowledgement

This research was partially supported by the National Science Foundation through grant No. ECS-8910073.

References

[1] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees,* Wadsworth, Belmont, CA, 1984.

[2] L. Atlas, R. Cole, Y. Muthusamy, A. Lippman, J. Conner, D. Park, M. El-Sharkawi and R.J. Marks II, "A performance comparison of trained multi-layer perceptrons and trained classification trees," *Proc. IEEE,* vol. 78, 1990, pp. 1614-1619.

[3] I.K. Sethi, "Entropy nets: from decision trees to neural networks," *Proc. IEEE,* vol. 78, 1990, pp. 1605-1613.

[4] T.D. Sanger, "A tree-structured adaptive network for function approximation in high dimensional spaces," *IEEE Trans. Neural Networks,* vol. 2, 1991, pp. 285-293.

[5] J. Sklansky and L. Michelotti, "Locally trained piecewise linear classifiers," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 2, 1980, pp. 101-111.

[6] Y. Park and J. Sklansky, "Automated design of linear tree classifiers," *Pattern Recognition,* vol. 23, 1990, pp. 1393-1412.

[7] R.P. Lippman, "An introduction to computing with neural nets," *IEEE ASSP Magazine,* April 1987, pp. 4-22.

[8] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations,* MIT Press, Cambridge, MA, 1986.

[9] S.B. Gelfand, C.S. Ravishankar and E.J. Delp, "An iterative growing and pruning algorithm for classification tree design," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 13, 1991, pp. 163-174.

[10] H. Guo and S.B. Gelfand, "Classification trees with neural network feature extraction," *IEEE Trans. Neural Networks,* to appear.

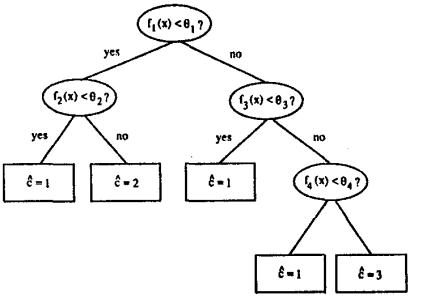

Fig. 1.1. A binary classification tree for a three-class problem

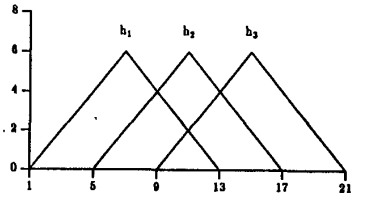

Fig. 4.1. Three basic waveforms

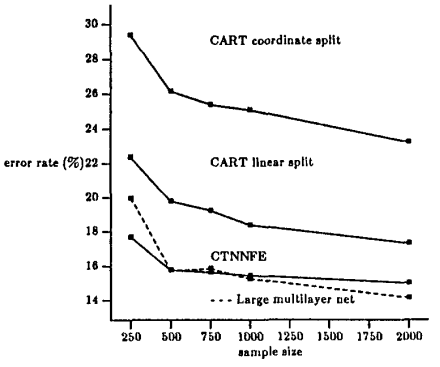

Fig. 4.2. Error rates vs. sample size for CART method, CTNNFE method and large multilayer net.

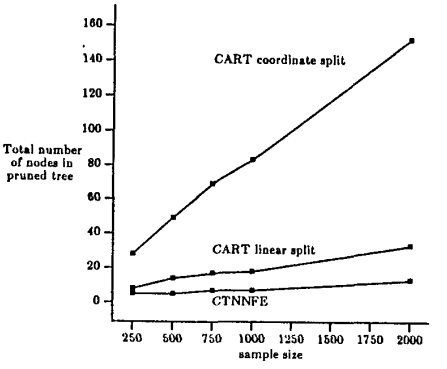

Fig. 4.4. Pruned tree size vs. sample size for CART method and CTNNFE method

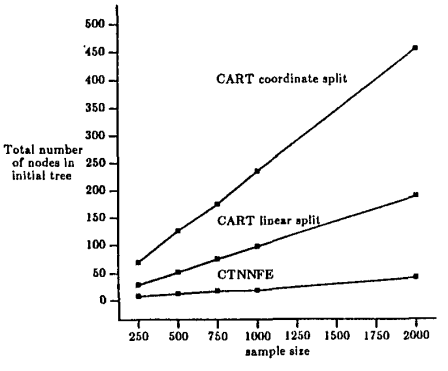

Fig. 4.3. Initial tree size vs. sample size for CART method and CTNNFE method

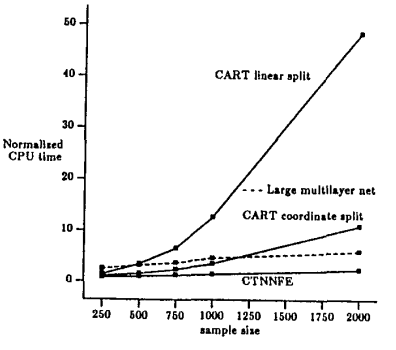

Fig. 4.5. CPU time vs. sample size for CART method, CTNNFE method and large multilayer neural net.

Table 4.1.Numerical results for character recognition example

CART method

Types of features	Error rate	Normalized CPU time	Total number of nodes in initial tree	Total number of nodes in pruned tree
coordinate	28.88%	1	177	74
linear	25.13%	4.4	100	48

CTNNFE method

	Error rate	Normalized CPU time	Total number of nodes in initial tree	Total number of nodes in pruned tree
$\mu = .1$	20.00%	4.1	21	19
$\mu = .2$	19.13%	3.5	25	20
$\mu = .5$	20.00%	3.6	29	23
$\mu = 1$	18.63%	2.8	36	25

Single multilayer net

	Error rate	Normalized CPU time
$\mu = .1$	19.88%	6.7
$\mu = .2$	19.50%	5.8
$\mu = .5$	23.38%	11.2
$\mu = 1$	84.63%*	> 15.0

*The backpropagation algorithm was unstable