

# A Connectionist Approach to Generating Oblique Decision Trees

Rudy Setiono and Huan Liu

Department of Information Systems and Computer Science

National University of Singapore

Kent Ridge, Singapore 119260

{rudys,liuh}@iscs.nus.edu.sg

phone: (65) 772-6297/772-6563

## Abstract

Neural networks and decision tree methods are two common approaches to pattern classification. While neural networks can achieve high predictive accuracy rates, the decision boundaries that they form are highly nonlinear and generally difficult to comprehend. The classification process of decision trees, on the other hand, can be readily translated into a set of rules. In this paper, we present a novel algorithm for generating oblique decision trees that capitalizes on the strength of both approaches. Oblique decision trees classify the patterns by testing on linear combinations of the input attributes. As a result, an oblique decision tree is usually much smaller than the univariate tree generated for the same domain.

Our algorithm consists of two components: connectionist and symbolic. A three-layer feedforward neural network is constructed and pruned for the

problem in hand. A decision tree is then built for the hidden unit activation values of the pruned network. An oblique decision tree is obtained by expressing the activation values in terms of the original input attributes. We test our algorithm on a wide range of problems. The oblique decision trees generated by the algorithm preserve the high accuracy of the neural networks, while keeping the explicitness of decision trees. Moreover, they outperform univariate decision trees generated by the symbolic approach and oblique decision trees built by other approaches in accuracy and tree size.

## 1 INTRODUCTION

Various application problems exist. Some are better addressed by the symbolic model; others are better taken care of by the connectionist model. Researchers realize that hybrid approaches can take advantage of both symbolic and connectionist models to attack tough problems [1]. In this work, we address the issue of building Oblique Decision Trees. Decision tree induction [2, 3] is an example of adopting a symbolic model. Decision trees are easy to understand. However, in some complex applications, typical decision trees may have too many branches and too many nodes along one branch. This will hinder the understanding of a decision tree. Researchers identify the one-feature test at a node [4] as one source of this problem. In general, a decision tree generates axis-parallel hyperplanes to partition the data. If oblique hyperplanes are allowed, a more compact decision tree can be constructed. Oblique decision trees are more general than univariate classification trees that make use of the one-feature test only. At each node of an oblique decision tree, a new feature that is a linear combination of the original input features is constructed. By using this new feature, the oblique decision tree effectively partitions the input attribute space by hyperplanes that are not neces-

sarily axis-parallel. Many researchers [4, 5, 6, 7] tried to find a hyperplane at each node of the tree by incorporating various methods such as simulated annealing, randomization, regression, and linear programming. They effectively employed a hybrid approach to this oblique decision tree generation problem.

There is also an example of using neural networks to extract nonlinear feature at each node of the classification tree [8]. It shares some commonalities with the methods that build oblique decision trees [4, 5, 6]: (1) they follow the same procedure how a decision tree is constructed; and (2) they are hybrid methods. The typical procedure of decision tree induction is a recursive one. It (1) starts at the root, (2) finds a feature to split the data, (3) stops if the split data are sufficiently pure [2], otherwise starts at the root of a subtree (i.e., starts recursion at (1)).

We use the neural network approach as an example to elaborate the process of generating nonlinear decision trees. Instead of finding  $a$  feature to test at a node, a small neural network is built at each node. Basically, a network that can best partition the data (as measured by an impurity criterion) is searched. This network then splits the data and the recursive procedure continues. For every node generated, a network is built. In other words, the number of networks built is as many as the internal nodes generated. This finding also applies to the methods employing simulated annealing, randomization, and regression to find oblique hyperplanes. We notice that finding new features is indeed a complicated process.

Oblique and nonlinear decision trees are wonderful tools, but the difficulties in finding the right combinations of input attributes to be used as new features have deterred their wide application. We need to find a way that can simplify the tree building process and avoid the pitfalls other methods may experience. We propose a novel approach. Instead of finding a new feature at each node using a neural

network or a randomized algorithm, we apply a neural network to the data before the tree is built. The basic idea is simple. The data is first used to train and prune a network. After a network is obtained, its hidden unit activation values are used to construct a univariate decision tree. Then each test at the nodes of the tree is replaced by a linear combination of initial attributes. The replacement can be easily worked out based on the weights and connections in the first layer (input units to hidden units) of the network.

The structure of this paper is as follows. Section II describes the details of our proposed connectionist-symbolic method. The connectionist component of the algorithm consists of a network construction algorithm and a network pruning algorithm. The symbolic component is C4.5 [10], a popular algorithm which builds decision trees. Section III presents the experimental results from our proposed algorithm. Comparisons are made between our algorithm and two other methods that build oblique decision trees (CART [2] and OC1 [6]). Section IV concludes the paper.

## **2 NN-DT: A CONNECTIONIST-SYMBOLIC METHOD**

A standard three-layer feedforward neural network is the connectionist component of our method. The number of units in the hidden layer is automatically determined by a network construction algorithm. A pruning algorithm is then applied to the constructed network to remove redundant and irrelevant network connections. Input units that have all their connections to the hidden layer removed correspond to irrelevant data attributes. Hence, input attributes not needed for classification are identified and eliminated by network pruning.

The concept learned by a trained network is embedded in the hidden layer of the network. The hidden unit activation values of the patterns are computed as the hyperbolic tangent of the weighted inputs. By having a sufficiently large number

of units, arbitrary decision boundaries can be formed by the network. As many classification problems require highly nonlinear decision boundaries, this attractive feature of neural networks leads to their wide applications. On the other hand, the complex decision boundaries may deter more practitioners of machine learning from applying the connectionist techniques in applications where explanation for the classifications of the data is required.

Our method, NN-DT (Neural Network and Decision Tree) attempts to strike a balance between the need to be able to form complex decision boundary for a problem and the need to be able to articulate the classification process. NN-DT achieves this balance by generating a decision tree for the activation values of a trained network. Hence, starting from the root node and proceeding down the tree, we have at each node the test

if  $AV < \zeta$ , then left branch,

otherwise right branch,

where AV is the activation value and  $\zeta$  is a threshold determined by the decision tree learning algorithm.

Let  $x_1, x_2, \dots, x_n$  denote the  $n$  components of the input vector  $x$  and  $w_{ij}$  denote the weight for the connection from input unit  $i$  to hidden unit  $j$ . Let  $a_j$  be the threshold of the  $j$ -th hidden unit. The activation value for input  $x$  at hidden unit  $j$  is  $AV_j$ , which is computed as

$$AV_j = \tanh \left( \sum_{i=1}^n x_i w_{ij} + a_j \right)$$

Since the hyperbolic tangent function is a one to one function, the test at a node of the decision tree can be replaced by its equivalent test

if  $(\sum_{i=1}^n x_i w_{ij} + a_j) < \tanh^{-1}(\zeta)$ , then left branch,

otherwise right branch.

Note that the condition of the test is now a linear combination of the inputs with the weights determined by the neural network. By applying a symbolic learning algorithm to the hidden unit activations of the trained network, we effectively obtain a classifier that divides the input attribute space by hyperplanes. Only the activations of the input patterns that have been correctly classified by the network are used to build a decision tree. Hence, the accuracy of the decision tree is at least as high as that of the network on the training dataset. On the testing dataset, the results of our extensive experiment show that there is no significant difference between the accuracy rates of the networks and the decision trees generated using the networks' hidden unit activations.

While the predictive accuracy rates of the decision trees built by NN-DT are comparable to those of the networks, the decisions made by the trees are easier to interpret. One may argue that a decision tree that tests on a linear combination of the features at each of its node is not as interpretable as a tree that tests only on a single attribute, however, the number of nodes of the former is usually much smaller than the latter. By testing on the contribution of the attributes as a group, more effective splitting of the data can be achieved. This inevitably leads to a reduction in the number of nodes needed to classify the patterns. For many applications, the dramatic reduction in the tree size makes it possible to elucidate the partitioning of the data with more ease.

The steps of our algorithm are as follows:

**Algorithm NN-DT:**

*Input:* A classification problem with  $n$  training examples.

*Output:* An oblique decision tree.

1. Construct a feedforward neural network with a single hidden layer.
2. Prune unnecessary connections and units from the network.

3. Apply a symbolic method to build a decision tree for the hidden unit activations of all patterns that have been correctly classified by the network.
4. Replace all conditions of the tests for node splitting in the decision tree by their linear equivalents.

#### *A. A neural network construction algorithm*

It is known that a network having a single hidden layer is capable of approximating any decision boundary, in general however, it is not known how many units in the hidden layer are needed. Often, the number of hidden units is determined by trial and error. Several rounds of experiments are needed if the number of hidden units is to be determined by the trial and error approach. If the network has too many hidden units, it may overfit the data and result in poor generalization. On the other hand, a network with too few hidden units may not be able to achieve the required accuracy rate. Algorithms that automatically build feedforward neural networks have been proposed by many researchers to address the problem of finding a suitable number of hidden units. The dynamic node creation (DNC) method proposed by Ash [11] is one such algorithm. It creates feedforward neural networks by sequentially adding hidden units to the hidden layer.

Our neural network construction algorithm MLNNCA [12] is similar to Ash's the dynamic node creation algorithm. The difference between DNC and MLNNCA lies in the error function that is to be minimized during network construction. DNC minimizes the usual sum of squared errors, while MLNNCA minimizes the cross-entropy error function. Both algorithm start with a single hidden layer network consisting of a single hidden unit and find a set of optimal weights for this network. If the network with these weights does not achieve the required accuracy rate, then one hidden unit is added to the network and the network is retrained. The process is repeated until a network that correctly classifies all the input patterns or meets

some other prespecified stopping criteria has been constructed. The outline of our network construction algorithm is given below.

## Maximum Likelihood Neural Network Construction Algorithm (MLNNCA)

1. Let  $H$  be the initial number of hidden units in the network. Set the all initial weights in the network randomly.
2. Find a point that minimizes the error function:

$$E(w, v) = - \left( \sum_{i=1}^k \sum_{p=1}^C t_p^i \log S_p^i + (1 - t_p^i) \log(1 - S_p^i) \right), \quad (1)$$

where

$k$  is the number of patterns.

$C$  is the number of output units.

$S_p^i$  is the predicted output for input  $x^i$  at output unit  $p$ ,

$$S_p^i = \sigma \left( \sum_{j=1}^H \tanh \left( (x^i)^T w^j \right) v_p^j \right)$$

$x^i$  is an  $n$ -dimensional input pattern,  $i = 1, 2, \dots, k$ .

$w^j$  is an  $n$ -dimensional vector of weights for the arcs connecting the input layer and the  $j$ -th hidden unit,  $j = 1, 2, \dots, h$ .  $(x^i)^T w^j$  is the scalar product of the two  $n$ -dimensional vectors.

$v^j$  is a  $C$ -dimensional vector of weights for the arcs connecting the  $j$ -th hidden unit and the output layer. The weight of the connection from the  $j$ -th hidden unit to the  $p$ -th output unit is denoted by  $v_p^j$ .

$\sigma(x)$  is the sigmoid function,  $1/(1 + \exp(-x))$ .

$t_p^i$  is the target output for input  $x_i$  at output unit  $p$ .

3. If the set of weights that minimizes the error function (1) results in a network that meets a prespecified stopping condition, then stop.

4. Add one unit to the hidden layer and select initial weights for the arcs connecting this new node with the input units and the output unit. Set  $H = H + 1$  and go to Step 2.

The available patterns are divided into 2 sets, one set of data for constructing the network and the second set for cross-validation. If the addition of a new hidden unit does not increase the accuracy of the network on the cross-validation set, MLNNCA is terminated. The previous setting of the network, i.e., the network with  $H - 1$  hidden units is the output of the algorithm.

The connections between the new hidden unit and the input/output units are assigned initial weights according to the formulation described in our earlier work [12]. We will not repeat the details here, but it is worth noting that with this formulation, the cross-entropy error function (1) is guaranteed to decrease with the addition of a new hidden unit.

### *B. N2P2F: A network pruning algorithm*

Methods for removing individual weights from a network usually augment a penalty term to the network error function (Hertz, et al. 1991). By adding a penalty term to the error function, the relevant and irrelevant network connections can be distinguished by the magnitudes of their weights or by other measures of saliency when the training process has been completed. The saliency measure of a connection gives an indication on the expected increase in the error function after that connection is eliminated from the network. In the pruning methods Optimal Brain Damage [13] and Optimal Brain Surgeon [14], the saliency of each connection is computed using a second order approximation of the error function near a local minimum. If the saliency of a connection is below a certain threshold, then the connection is removed from the network. If the increase in the error function is larger than a predetermined acceptable error increase, the network must to be

retrained

The algorithm N2P2F for neural network pruning outlined below was recently developed by Setiono [15]. Neural Network Pruning with Penalty Function (N2P2F) first trains a fully connected network to find a set of weights that minimizes the augmented error function:

$$\tilde{E}(w, v) = E(w, v) + P(w, v), \quad (2)$$

where the penalty is

$$P(w, v) = \epsilon_1 \sum_{j=1}^h \left( \sum_{\ell=1}^n \frac{\beta(w_\ell^j)^2}{1 + \beta(w_\ell^j)^2} + \sum_{p=1}^C \frac{\beta(v_p^j)^2}{1 + \beta(v_p^j)^2} \right) + \epsilon_2 \sum_{j=1}^h \left( \sum_{\ell=1}^n (w_\ell^j)^2 + \sum_{p=1}^C (v_p^j)^2 \right), \quad (3)$$

$\beta, \epsilon_1, \epsilon_2$  are positive penalty parameters. The error term  $E(w, v)$  is the cross-entropy error function (1).

#### Algorithm N2P2F: Neural Network Pruning with Penalty Function

1. Let  $\eta > 0$  be a threshold value that determines if a weight can be removed.
2. Pick a fully connected network and train this network to minimize the error function (2) such that a prespecified accuracy level is met. Let  $(w, v)$  be the weights of this network.
3. For each connection from input unit  $\ell$  to hidden unit  $j$ ,  $w_\ell^j$  in the network, if

$$\max_p |v_p^j w_\ell^j| \leq \eta, \quad (4)$$

then remove  $w_\ell^j$  from the network.

4. For each connection from hidden unit  $j$  to output unit  $p$ ,  $v_p^j$  in the network, if

$$|v_p^j| \leq \eta, \quad (5)$$

then remove  $v_p^j$  from the network.

5. If no weight satisfies condition 4 or condition 5, then for each  $w_\ell^j$  in the network, compute

$$\omega_\ell^j = \max_p |v_p^j w_\ell^j|.$$

Remove  $w_\ell^j$  with the smallest  $\omega_\ell^j$ .

6. Retrain the network. If classification rate of the network falls below the specified level, then stop and use the previous setting of network weights. Otherwise, go to Step 3.

In steps 3 and 4, N2P2F removes all the connections of the network whose magnitudes satisfy the conditions (4) or (5). The justification for these steps of the algorithm are described in [15]. In step 5, we remove a network connection from an input unit to a hidden unit  $w_\ell^j$  based on the values of its products with the weight of the connections from hidden unit  $j$  to all output units  $p = 1, 2, \dots, C$ . The connection with the smallest maximum product is selected for removal. After one or more connections are removed, the network is retrained in Step 6.

When constructing a neural network, we divide the available patterns into training dataset and cross validation dataset. The same two sets can be used to decide whether or not the pruning algorithm should be terminated. In our implementation, we keep the highest accuracy of the network on the cross validation dataset. If the removal of a connection in causes the accuracy of the network on this dataset to drop by more than 1 % from its best value, we stop the algorithm.

### *C. Generating decision rules*

The aim of neural network pruning is to obtain a network with only the relevant attributes and connections among its units. By removing connections from a network, overfitting of the data can be reduced, hence generalization ability of the network can be improved [16, 17]. In general, however, it is still difficult to

interpret the classification of even a small network with a few connections in a meaningful fashion.

In order to interpret the network classification, we apply C4.5 [10] using the network hidden activations as input. C4.5 is an enhanced version of ID3 [3]. At each decision node, ID3 uses information theory to select a single feature of the data that gives the largest information gain. An illustration of how information gain is computed as well as the description of the ID3 algorithm can be found in Quinlan’s work [18] and in [19]. Unlike ID3, C4.5 can handle continuous input attributes. C4.5 also generates a set of classification rules by tracing a path from the root of the decision tree to each leaf. The rules are then pruned by removing unnecessary rule conditions.

By making use of C4.5 as a component of the system, we can exploit the best features of the connectionist and symbolic learning algorithms. Neural networks are known to be excellent classifiers for a wide range of applications, while the rules generated by C4.5 are clearly more comprehensible than the collection of weights of a trained network.

### **3 EXPERIMENTS**

In this section, we present the experimental results from our proposed NN-DT algorithm. The datasets that we have selected to test the algorithm are publicly available and they can be obtained via ftp from the UC Irvine data repository [20]. We conducted our experiments to see if the following claims can be substantiated:

- The accuracy of the method is at least as good as its two individual components, neural network and C4.5.
- The tree generated by the method is smaller than the tree generated by C4.5.

- The accuracy and tree size of our proposed method compare favorably with those of other methods that generate oblique decision trees.

For comparison purpose, two other methods that generate oblique decision trees are tested. These methods are CART [2] and OC1 [6]:

1. CART splits each node in the decision tree as to maximize the purity of the resulting subsets. A node with patterns that belong only to one class has the highest purity. Nodes with patterns from different classes has a nonzero Gini diversity index. A split is induced by the hyperplane  $\sum_i a_i x_i \leq c$ , where  $x_i$  is the normalized attributes,  $a_i$  the coefficients that determine the orientation of the hyperplane, and  $c$  is a threshold. The values of  $a_i$  and  $c$  are fine-tuned by perturbing their values to decrease the impurity of the split.
2. OC1 first finds the best axis-parallel split at a node, it then looks for a better split by searching for oblique hyperplanes in the attribute space. Oblique hyperplanes with lower impurity than the best axis-parallel split are obtained by randomly perturbing the current hyperplane to a new location.

The 14 datasets selected for the experiment are summarized in Table 1. For nominal attributes, their values are binarized. Doing so is not biased against C4.5 since researchers [21] have observed that this would improve decision trees' accuracy. In order to reduce bias in computing the accuracy and tree size, we perform ten fold cross validation for each dataset:

1. Randomly divide the available patterns into 10 disjoint subsets, each consisting of approximately 10 % of the patterns.
2. Use the patterns in the first 8 subsets as the training dataset, the patterns in the 9th subset as the cross validation set, and those in the last subset as the testing set.

Name	#Data	#A	Type
1. Monk1	432	17	binary
2. Monk2	432	17	binary
3. Monk3	432	17	binary
4. TicTacToe	958	27	binary
5. Vote	300	48	binary
6. BreastCancer	699	9	continuous
7. Bupa	345	6	continuous
8. Ionosphere	351	34	continuous
9. Iris	150	4	continuous
10. Pima-diabetes	768	8	continuous
11. Sonar	208	60	continuous
12. Australian	690	14	mixed
13. HeartDisease	297	13	mixed
14. Housing	506	13	mixed

Table 1: Dataset Summary. #Data - data size, Type - attribute type, and #A - number of attributes.

3. Apply NN-DT as follows:

- (a) Construct a network using the algorithm MLNNCA for the training dataset. Stop MLNNCA if adding one more hidden unit does not improve the accuracy rate of the network on the cross validation set.
- (b) Retrain the constructed network with the penalty-augmented error function (2).
- (c) Apply the algorithm N2P2F to prune the network. Terminate N2P2F

if the accuracy of the network on the cross validation set drops by more than 1% from its best predictive rate.

- (d) Report the accuracy of the pruned network on the testing set.
  - (e) Apply C4.5 to build a decision tree using the activation values of the patterns in the training and cross validation datasets that have been correctly classified by the pruned network. Report the accuracy of the tree on the testing dataset and the size of the decision tree.
4. Build a decision tree using C4.5/OC1/CART using the combined training set and cross validation set. Report the accuracy on the testing dataset and the size of the decision tree.
  5. Repeat from Step 2 with different selection of training/cross validation/testing datasets until each subset has been used as the testing dataset once.

C4.5, OC1 and CART were run using their default parameter settings. We started our neural network construction algorithm with an initial  $H = 1$  hidden unit. A number of parameters in the penalty term of the error function ( 3) influence the performance of the pruning algorithm. For some problems, larger values of the parameters  $\epsilon_1, \epsilon_2$  and  $\beta$  may result in more network connections removed at the same time in Steps 3 and 4 of N2P2F. As a consequence, the total number of epochs needed to retrain the network can be substantially reduced. For some other problems, however, retraining a newly constructed network with the augmented error function having large penalty parameters can adversely affect the accuracy of the network. For all the experimental results that we report here, we have set the values of the parameters as follows:  $\epsilon_1 = 1, \epsilon_2 = 10^{-4}$ , and  $\beta = 10$ . The value of the threshold for removing network connections (cf. Conditions 4 and 5) is  $\eta = 0.40$ .

Both the network construction algorithm and the network pruning algorithm make use of a variant of the family of quasi-Newton optimization algorithms, the BFGS method, to minimize the error function. The BFGS method, having a superlinear convergence rate, has been shown by many researchers to be much faster than the standard backpropagation method for neural network training. An outline of the BFGS method can be found in [22].

We show the results in Tables 2 and 3. The P-values that are the results of two-tailed  $t$ -tests are also included in the tables. They are calculated for NN, C4.5, OC1, and CART against NN-DT using the results from ten-fold cross-validation run of each algorithm. We want to check whether the difference of the averages between every pair is statistically significant. A small P-value leads us to reject the null hypothesis that there is no difference in the two averages being compared. In this case, a further check will show which method is better by examining the two average values.

In the first column of Table 2 the average accuracy rates for NN-DT are shown. In the second column of the table, the average accuracy rates of the pruned neural networks (NN) are shown. The next three columns show the average accuracy of C4.5, OC1, and CART, respectively. In Table 3, the average tree sizes are shown for the 4 methods that generate decision trees: NN-DT, C4.5, OC1 and CART. A tree size reflects the number of nodes in the tree.

The summaries of the average accuracy rates and tree sizes are shown in Figures 1 and 2. An average from NN-DT is compared against the corresponding average from each of the other methods and the two values are plotted as point  $(x, y)$ , where  $y$  is the average from NN-DT's and  $x$  is that from one of the other methods. In Figure 1, points that are above the 45-degree diagonal line indicate that NN-DT outperforms the other method. As we can see from this figure, for the majority of datasets, NN-DT does perform better than other methods except NN. The

Name	NN-DT	NN	C4.5	OC1	CART
Monk1 (P-value)	100	100 -	100 -	100 -	100 -
Monk2 (P-value)	100	100 -	93.7 <b>(0.0025)</b>	99.77 (0.3434)	99.31 (0.3434)
Monk3 (P-value)	100	100 -	100 -	100 -	97.22 <b>(0.0133)</b>
TicTacToe (P-value)	98.0	97.71 (0.1801)	93.8 <b>(0.0001)</b>	91.75 <b>(0.0034)</b>	86.33 <b>(0.0001)</b>
Vote (P-value)	96.3	96.33 (0.7583)	97.3 (0.1944)	93.10 <b>(0.0422)</b>	95.17 (0.4073)
B-Cancer (P-value)	96.1	96.0 (0.7583)	95.3 (0.2423)	94.99 (0.4033)	94.71 (0.2777)
Bupa (P-value)	68.8	68.18 (0.5565)	61.1 <b>(0.0475)</b>	66.09 (0.5289)	65.22 (0.3443)
Ionosphere (P-value)	88.1	88.34 (0.5986)	90.0 (0.4249)	88.32 (0.9376)	86.61 (0.6827)
Iris (P-value)	99.3	91.67 (0.1682)	94.0 <b>(0.0867)</b>	96.00 <b>(0.0611)</b>	94.00 <b>(0.0342)</b>
Pima (P-value)	76.4	76.32 (0.8996)	70.9 <b>(0.0094)</b>	72.40 <b>(0.0362)</b>	72.40 <b>(0.0362)</b>
Sonar (P-value)	86.1	86.55 (0.3521)	85.5 (0.8519)	85.58 (0.9524)	87.02 (0.8482)
Australian (P-value)	86.7	85.52 0.3955	84.2 (0.2014)	83.04 <b>(0.0922)</b>	86.09 (0.7988)
Heart (P-value)	82.8	83.15 0.7944	72.0 <b>(0.0046)</b>	73.74 <b>(0.0068)</b>	77.78 0.1335
Housing (P-value)	85.8	85.96 0.5912	82.0 <b>(0.0256)</b>	82.41 (0.2625)	80.83 <b>(0.0470)</b>

Table 2: 10-fold cross validation results - average accuracy (%). P-values in bold mean that results are significantly different from NN-DT's.

Name	NN-DT	C4.5	OC1	CART
Monk1 (P-value)	9.8	14.8 <b>(0.0001)</b>	7.0 <b>(0.0013)</b>	9.4 (0.5594)
Monk2 (P-value)	8.0	87.6 <b>(0.0001)</b>	5.4 (0.1174)	7.4 (0.7418)
Monk3 (P-value)	10.0	15.0 <b>(0.0001)</b>	3.0 <b>(0.0001)</b>	3.0 <b>(0.0001)</b>
TicTacToe (P-value)	5.6	71.8 <b>(0.0001)</b>	17.6 <b>(0.0787)</b>	20.8 <b>(0.0185)</b>
Vote (P-value)	3.0	6.6 <b>(0.0001)</b>	3.2 (0.3434)	3.2 (0.3434)
B-Cancer (P-value)	3.2	22.2 <b>(0.0001)</b>	5.0 (0.1105)	5.8 <b>(0.0533)</b>
Bupa (P-value)	5.8	79.2 <b>(0.0001)</b>	9.2 (0.3961)	18.6 (0.1495)
Ionosphere (P-value)	3.0	24.8 <b>(0.0001)</b>	8.6 <b>(0.0552)</b>	21.4 <b>(0.0005)</b>
Iris (P-value)	5.0	8.2 <b>(0.0011)</b>	5.0 -	5.2 (0.3434)
Pima (P-value)	3.0	122.4 <b>(0.0001)</b>	11.4 <b>(0.0196)</b>	26.2 <b>(0.0722)</b>
Sonar (P-value)	3.0	31.0 <b>(0.0001)</b>	8.2 <b>(0.0056)</b>	16.4 <b>(0.0002)</b>
Australian (P-value)	5.4	68.7 <b>(0.0001)</b>	4.8 (0.8226)	6.2 (0.7571)
Heart (P-value)	3.0	51.9 <b>(0.0001)</b>	9.2 <b>(0.0566)</b>	7.4 <b>(0.0008)</b>
Housing (P-value)	9.6	53.8 <b>(0.0001)</b>	4.2 (0.1510)	8.8 (0.8640)

Table 3: 10-fold cross validation results - tree size (number of nodes including leave nodes). P-values in bold mean that results are significantly different from NN-DT's.

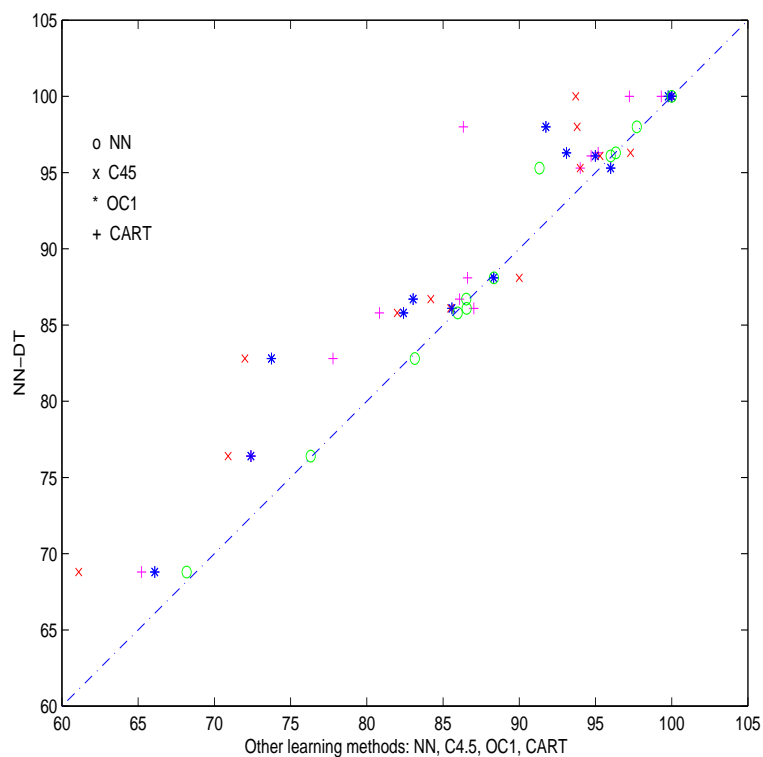


Figure 1: Comparison of accuracy rates of C4.5, OC1, and CART against NN-DT's.

accuracy rates of NN and NN-DT are very similar for most problems as reflected by the many circles lying very near to the diagonal line. C4.5 has better average accuracy than NN-DT on 2 of the 14 datasets. The improvements, however, are not statistically significant.

In Figure 2, eight points are above the diagonal line, while the remaining 20 points are on or below the line. A point under the line corresponds to an NN-DT's tree that is smaller than the one generated by OC1 or CART. The results from C4.5 are not included in the figures, as C4.5 generates trees that have as many as 122.4 nodes in average for one of the dataset.

The conclusions drawn from the experiments are:

*Predictive accuracy:* There is no significant difference in the accuracy of the trees

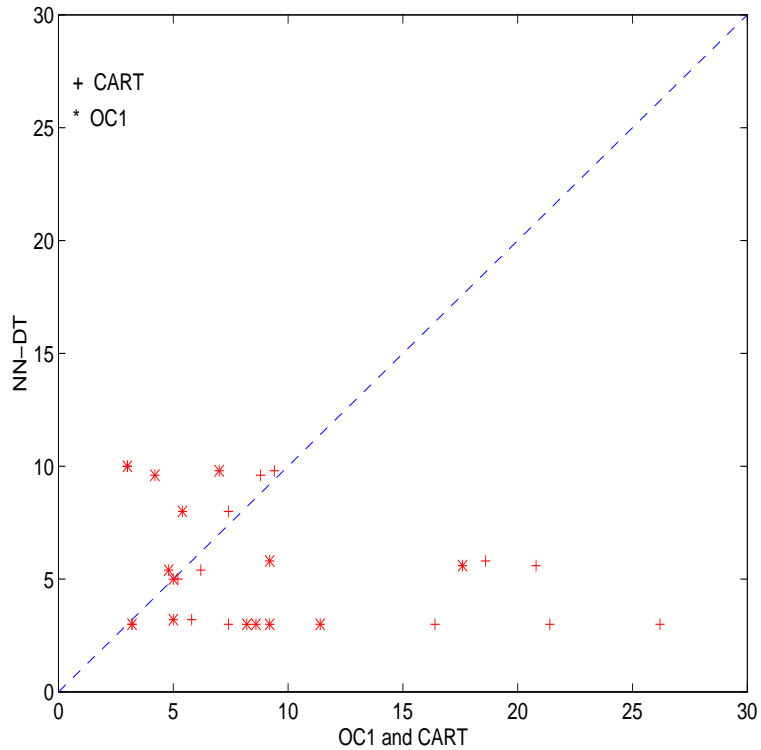


Figure 2: Comparison of tree sizes of OC1 and CART against NN-DT's.

generated by NN-DT and the accuracy of the neural networks whose activation values have been used to generate the trees. The performance of NN-DT is significantly different from that of C4.5 in 7 out of 14 cases. For all 7 cases, the accuracy of NN-DT is higher. When the results of NN-DT are statistically different from those of OC1 and CART, NN-DT also always achieves higher accuracy. The numbers of cases for which NN-DT achieves significantly higher accuracy over OC1 and CART are 6 and 5 out of 14 cases, respectively.

*Tree size:* As expected, compared to the single-feature nodes of the trees generated by C4.5, the number of nodes of NN-DT trees is significantly lower. The reduction of the number of nodes is more pronounced in the case of datasets

with continuous attributes. For the Pima dataset, this reduction is from 122.4 to just 3. For this dataset, the predictive accuracy is also improved significantly by applying NN-DT. The accuracy of C4.5 and NN-DT are 70.9 % and 76.4 %, respectively. In 6 cases, trees created by NN-DT are significantly different from those of OC1's, in only one of the 6 cases, OC1's tree is smaller. In 7 cases, NN-DT's trees are significantly different from those of CART's, in only one of the 7 cases, CART's tree is smaller.

## 4 CONCLUSIONS

We have proposed and implemented a simple method for generating oblique decision trees for pattern classification. The method, NN-DT, makes use of both connectionist and symbolic approaches. For a given problem domain, a three-layer feedforward neural network is constructed and pruned. The hidden unit activations of the pruned network are then used to generate a decision tree. Since the hyperbolic tangent function is used to compute the activation values, the conditions for node splitting in the decision tree involve nonlinear terms that are the hyperbolic tangent of linear combinations of a set of input attributes. The nonlinearity, however, can be easily eliminated by the fact that the hyperbolic tangent function is a one-to-one function. Thus, the algorithm effectively generates an oblique decision tree which partitions the input attribute space by hyperplanes.

Our experimental results have been conducted on a wide range of publicly available datasets. These datasets have discrete, continuous or mixed discrete/continuous attributes. The performance of NN-DT compares favorably in accuracy and tree size to those of OC1 and CART, two algorithms that also generate oblique decision trees. When compared to the univariate trees generated by C4.5, the oblique trees generated by NN-DT have far fewer nodes. Compared with the neural networks from which the oblique decision trees are generated by NN-DT, NN-DT yields

comparable accuracy rates. Unlike the neural network's, however, the classification process of a decision tree can be expected to be easier to comprehend.

#### ACKNOWLEDGMENTS

We wish to thank S. Murthy and S. Salzberg for making the code of OC1 available by anonymous ftp. We also thank J. Yao and M. Dash for their help in obtaining the results of OC1 and CART reported in this paper.

## References

- [1] R. Sun, "Hybrid connectionist-symbolic models, a report from the IJCAI 95 workshop on connectionist-symbolic integration," *AI Magazine*, pp. 99-103, Summer 1996.
- [2] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and regression trees*. Belmont, CA: Wadsworth International Group, 1984.
- [3] J.R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 6, no. 1, pp. 81-106, 1986.
- [4] C.E. Brodley and P.E. Utgoff, "Multivariate decision trees," *Machine Learning*, vol. 19, pp. 45-77, 1995.
- [5] D. Heath, S. Kasif, and S. Salzberg, "Learning oblique decision trees," in *Proc. 13th International Joint Conference on Artificial Intelligence*, 1993, pp. 1002-1007.
- [6] S. Murthy, S. Kasif, S. Salzberg, and R. Beigel, "Oc1: Randomized induction of oblique decision trees," in *Proc. of AAAI Conference*, 1993, pp. 322-327.

- [7] O.L. Mangasarian, R. Setiono, and W. Wolberg, “Pattern recognition via linear programming: Theory and application to medical diagnosis,” in *Large-scale Numerical Optimization*, T.F. Coleman and Y. Li, Eds. Philadelphia, PA: SIAM, 1989, pp.22–30.
- [8] H. Guo and S.B. Gelfand, “Classification trees with neural network feature extraction,” *IEEE Trans. Neural Networks*, vol. 3, no. 6, pp. 924–933.
- [9] S.M. Weiss and Kulikowski, *Computer system that learns*. San Mateo, CA: Morgan Kaufmann Publishers, 1991.
- [10] J.R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.
- [11] T. Ash, “Dynamic node creation in backpropagation networks,” *Connection Science*, vol. 1, no. 4, pp. 365–375, 1989.
- [12] R. Setiono, “A neural network construction algorithm which maximizes the likelihood function”, *Connection Science*, vol. 7, no. 2, pp. 147–166, 1995.
- [13] Y. Le Cun, J.S.Denker, and S.A. Solla, “Optimal brain damage,” in *Advances in Neural Information Processing Systems*, 1990, vol. pp. 2, pp. 598–605.
- [14] B. Hassibi and D.G. Stork, “Second order derivatives for network pruning: optimal brain surgeon”, in *Advances in Neural Information Processing Systems*, 1993, vol. 5, pp. 164–171.
- [15] R. Setiono, “A penalty function approach for pruning feedforward neural networks”, *Neural Computation*, vol. 9, no. 1, pp. 185-204, 1997.
- [16] E.D. Karnin, “A simple procedure for pruning back-propagation trained neural network,” *IEEE Transaction on Neural Networks*, vol. 1, no. 2, pp. 239–242, 1990.

- [17] M. Hagiwara, “A simple and effective method for removal of hidden units and weights,” *NeuroComputing*, vol. 6, pp. 207–218, 1994.
- [18] J.R. Quinlan, “Learning efficient classification procedures and their application to chess end games,” in *Machine Learning: an artificial intelligence approach*, R.S. Michalski, J.G. Carbonnel and T.M. Mitchell, Eds. Los Altos, CA: Morgan Kaufmann, 1983.
- [19] K.J. Cios and N. Liu, “A machine learning method for generation of a neural network architecture: a continuous ID3 algorithm,” *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 280–291, 1992.
- [20] P. Murphy and D. Aha, UCI repository of machine learning databases, 1994. FTP from ics.uci.edu in the directory pub/machine-learning-databases.
- [21] J. Shavlik, R. Mooney, and G. Towell, “Symbolic and neural learning algorithms: An experimental comparison,” *Machine Learning*, vol. 6, no. 2, pp. 111–143, 1991.
- [22] R. Setiono and L.C.K. Hui. “Use of quasi-Newton method in a feedforward neural network construction algorithm,” *IEEE Transactions on Neural Networks*, vol. 6, No. 1, pp. 273-277, 1995.