Online Adaptive Decision Trees

Article *in* Neural Computation · October 2004

DOI: 10.1162/0899766041336396 · Source: PubMed

1 author:

Jayanta Basak

106 PUBLICATIONS **902** CITATIONS

LETTER Communicated by Jerome Friedman

Online Adaptive Decision Trees

Jayanta Basak
bjayanta@in.ibm.com
IBM India Research Lab, Indian Institute of Technology, New Delhi–110048, India

Decision trees and neural networks are widely used tools for pattern classification. Decision trees provide highly localized representation, whereas neural networks provide a distributed but compact representation of the decision space. Decision trees cannot be induced in the online mode, and they are not adaptive to changing environment, whereas neural networks are inherently capable of online learning and adpativity. Here we provide a classification scheme called online adaptive decision trees (OADT), which is a tree-structured network like the decision trees and capable of online learning like neural networks. A new objective measure is derived for supervised learning with OADT. Experimental results validate the effectiveness of the proposed classification scheme. Also, with certain real-life data sets, we find that OADT performs better than two widely used models: the hierarchical mixture of experts and multilayer perceptron.

1 Introduction

Two major tools for classification (Duda & Hart, 1973; Jain, Duin, & Mao, 2000) are neural networks (Haykin, 1999) and decision trees (Breiman, Friedman, Olshen, & Stone, 1983; Quinlan, 1993). The first, especially feedforward networks, embeds a distributed representation of the inherent decision space, whereas the decision tree always makes a localized representation of the decision. The decision tree (Duda, Hart, & Stork, 2001; Durkin, 1992; Fayyad & Irani, 1992; Friedman, 1991; Breiman et al., 1983; Quinlan, 1993, 1996; Brodley & Utgoff, 1995) is a widely used tool for classification in various domains, such as text mining (Yang & Pedersen, 1997), speech (Riley, 1989; Chien, Huang, & Chen, 2002), bio-informatics (Salzberg, Delcher, Fasman, & Henderson, 1998), web intelligence (Zamir & Etzioni, 1998; Cho, Kim, & Kim, 2002), and many other fields that need to handle large data sets. Decision trees are in general constructed by recursively partitioning the training data set into subsets such that finally certain objective error (such as impurity) over the entire data is minimized. Although the very nature of the construction of axis-parallel decision trees always tries to reduce the bias at the cost of variance (it can be a high generalization error), axis-parallel decision trees are quite popular in terms of their interpretability; each class

can be specifically described by a set of rules defined by the path from the root to the leaf nodes of the tree. Variations of decision trees also exist in the literature such as SVDT (Bennett, Wu, & Auslender, 1998) and oblique decision trees (Murthy, Kasif, & Salzberg, 1994), which take into account the combination of more than one attribute in partitioning the data set at any level (at the cost of interpretability). Decision trees usually perform hard splits of the data set, and if a mistake is committed in splitting the data set at a node, then it cannot be corrected further down the tree. In order to take care of this problem, attempts have been made (Friedman, Kohavi, & Yun, 1996) that in general employ various kinds of look-ahead operators to guess the potential gain in the objective after more than one recursive split of the data set. Second, a decision tree is always constructed in the batch mode, that is, the branching decision at each node of the tree is induced based on the entire set of training data. Different variations have been proposed for an efficient construction mechanism (Utgoff, Berkman, & Clouse, 1997; Mehta, Agrawal, & Rissanen, 1996). Online algorithms (Kalai & Vempala, n.d.; Albers, 1996), on the other hand, sometimes prove to be more useful in the case of streaming data, very large data sets, and situations where memory is limited.

Unlike decision trees, neural networks (Haykin, 1999) make decisions based on the activation of all nodes, and therefore even if a node is faulty (or makes mistakes) during training, it does not affect the performance of the network very much. Neural networks also provide a computational framework for online learning and adpativity. In short, a decision tree can be viewed as a token passing system, and a pattern is treated as a token that follows a path in the tree from the root node to a leaf node. Each decision node behaves as a gating channel, depending on the attribute values of the pattern token. Neural networks, on the other hand, resemble the nervous system in the sense of distributed representation. A pattern is viewed as a phenomenon of activation, and the activation propagates through the links and nodes of the network to reach the output layer.

Some attempts have been made to combine the neural networks with decision trees to obtain different classification models. For example, in the neural tree (Strömberg, Zrida, & Isaksson, 1991; Golea & Marchand, 1990), multilayer perceptrons are used at each decision node (intermediate node) of the tree for partitioning the data set. Attempts are also made to construct decision trees (Boz, 2000) from trained neural networks, as well as to construct neural networks (Lee, Jone, & Tsai, 1995) from trained decision trees. However, all of these methods attempt to hybridize the concepts borrowed from both these paradigms and learn locally within each node, although none of them addresses the issues of learning in terms of achieving overall classification performance.

Possibly the most elegant approach in this line is the hierarchical mixture of experts (HME) (Jordan & Jacobs, 1993, 1994) and the associated learning algorithm. In HME, a tree-structured network (similar to a decision tree)

is viewed as a hierarchy of the mixture of experts, and the output of the individual experts is combined selectively through gating networks in a bottom-up manner through the hierarchy such that the root node always represents the final decision. HME provides a general framework for both classification and regression, and a statistical framework, the expectation-maximization (EM) algorithm, has been used for estimating the network parameters in the batch mode. Jordan and Jacobs (1993) demonstrated that HME is a more powerful classification tool when compared with the feed-forward neural networks and decision trees. Online learning algorithms are also derived for HME.

Here we provide a scheme for online classification by preserving the structure of decision trees and the learning paradigm of neural networks. We call our model an online adaptive decision tree (OADT), which can learn in the online mode and can exhibit adaptive behavior (i.e., adapt to a changing environment). Online adaptive decision trees are complete binary trees with a specified depth where a subset of leaf nodes is assigned for each class, in contrast to HME, where root node always represents the class. In the online mode, for each training sample, the tree adapts the decision hyperplane at each intermediate node from the response of the leaf nodes and the assigned class label of the sample. Unlike HME, OADT uses the top-down structure of a decision tree; instead of probabilistically weighing the decisions of the experts and combining them, OADT allocates a data point to a set of leaf nodes (ideally one) representative of a class. Since OADT operates in a top-down manner, it requires a different objective error measure. The decision in this case is collectively represented by a set of leaf nodes. We provide a different objective error measure for OADT and subsequently derive the learning rules based on the steepest gradient descent criteria.

We organize the rest of the letter as follows. In section 2, we show how the leaf nodes should respond to a given sample in OADT for a specified structure of the tree. We then formulate an objective error measure to reflect the discrepency between the collective responses of the leaf nodes and the assigned class label of the training sample. We derive the learning rule by following the gradient descent of the error measure. In section 3, we demonstrate the performance of the OADT on real-life data sets. Finally, we discuss and conclude this letter in sections 4 and 5, respectively.

2 Description of the OADT

An OADT is a complete binary tree of depth l where l can be specified beforehand. A tree has a set of intermediate nodes D, $|D| = 2^l - 1$ and a set of leaf nodes L where $L = 2^l$, such that the total number of nodes in the tree is $2^{l+1} - 1$. Each intermediate node i in the tree stores a decision hyperplane in the form $(\mathbf{w}_i, \theta_i)$, a vector lr (the form of lr is explained later in equation 2.4), and a sigmoidal activation function $f(.)$. The weight vector $\mathbf{w}_i$ is n-dimensional for an n-dimensional input and constrained as $\|\mathbf{w}_i\| = 1$

for all i such that $(\mathbf{w}_i, \theta_i)$ together have n free variables. As a convention, we number the nodes in OADT in the breadth-first order starting from the root node numbered as 1, and this convention is followed throughout this article.

2.1 Activation of OADT. As input, each intermediate node can receive activation from its immediate parent and the input pattern $\mathbf{x}$. The root node does not have any parent, and it can receive only the input pattern. As output, each intermediate node produces two outputs—one for the left child and the other for the right child. According to our convention of numbering the nodes, node i has two children numbered as $2i$ and $2i + 1$ where node $2i$ is the left child of node i and node $2i + 1$ is the right child. If $u(i)$ denotes the activation received by node i from its parent, then the activation values received by the left and right children of node i are given as

$$u(2i) = u(i).f(\mathbf{w}_i'\mathbf{x} + \theta_i) \tag{2.1}$$

and

$$u(2i + 1) = u(i).f(-(\mathbf{w}_i'\mathbf{x} + \theta_i)). \tag{2.2}$$

In other words, the node i takes the product of the activation value received from its parent and the activation produced by itself and sends it to the child nodes. The activation values produced by the node i itself are different for two different child nodes. The value of $u(1)$ for the root node is considered to be unity.

The leaf nodes do not see the input pattern $\mathbf{x}$ and reproduce the activation received from its parent. Thus, from equations 2.1 and 2.2, the activation value of a leaf node p is given as

$$u(p) = \prod_{n=1}^{l} f\left((-1)^{mod(\lfloor \frac{p}{2^{n-1}} \rfloor, \lfloor \frac{p}{2^n} \rfloor)}(\mathbf{w}_{\lfloor \frac{p}{2^n} \rfloor}.\mathbf{x} + \theta_{\frac{p}{2^n}})\right), \tag{2.3}$$

that is, the activation of a leaf node is decided by the product of activations produced by the intermediate nodes on the path from the root node to the concerned leaf node. We define a function $lr(i, p)$ to represent whether the leaf node p is on the left path or the right path of an intermediate node i, such that

$$lr(i, p) = \begin{cases} 1 & \text{if } p \text{ is on the left path of } i \\ -1 & \text{if } p \text{ is on the right path of } i \\ 0 & \text{otherwise} \end{cases}. \tag{2.4}$$

Using the function $lr(.)$, the activation of a leaf node p can be represented as

$$u(p) = \prod_{i \in P_p} f\left(lr(i, p)(\mathbf{w}_i.\mathbf{x} + \theta_i)\right), \tag{2.5}$$

where P_p is the path from the root node to the leaf node p. The function $lr(.)$ is stored in the form of a vector at each intermediate node i (the respective $lr(i,:)$).

In OADT, each intermediate node can observe the input pattern, which is unlike the conventional neural networks. In the feedforward networks, usually there exist some hidden nodes that can observe only the input provided by other nodes but cannot observe the actual input pattern. During the supervised training phase of OADT, it is difficult to decide which leaf node should receive the highest activation. However, it is observed that the total activation of the leaf nodes is always constant if the activation function is sigmoid or of similar nature.

Claim 1. *The total activation of the leaf nodes is constant if the activation function of each intermediate node is sigmoidal in nature.*

Proof. Let us assume one form of the sigmoid function as

$$f(v;m) = \frac{1}{1+\exp(-mv)}. \tag{2.6}$$

Then

$$f(v;m) + f(-v;m) = 1. \tag{2.7}$$

This property in general holds true for any kind of symmetric sigmoid functions (including the S-function used in the literature of fuzzy set theory). For any child node p, the activation is given as

$$u(p) = u(\lfloor p/2 \rfloor).f((-1)^{mod(p,\lfloor p/2 \rfloor)}(\mathbf{w}_{\lfloor p/2 \rfloor}.\mathbf{x} + \theta_{\lfloor p/2 \rfloor})). \tag{2.8}$$

Thus, for any two leaf nodes p and $p+1$ having a common parent node $p/2$, we have

$$u(p) + u(p+1) = u(p/2). \tag{2.9}$$

Thus, the total activation of the leaf nodes is

$$\sum_{p=2^{l}}^{p=2^{l+1}-1} u(p) = \sum_{q=2^{l-1}}^{q=2^{l}-1} u(q). \tag{2.10}$$

By similar reasoning,

$$\sum_{q=2^{l-1}}^{q=2^{l}-1} u(q) = \sum_{r=2^{l-2}}^{r=2^{l-1}-1} u(r). \tag{2.11}$$

As we go up the tree until we reach the root node, the sum of the activation becomes equal to $u(1)$. Since we consider that $u(1) = 1$ by default, the total activation of the leaf nodes is

$$\sum_{p} u(p) = 1. \tag{2.12}$$

Therefore, decreasing the output of one leaf node will automatically enforce the increase in the activation of the other nodes given that the activation of the root node is constant.

2.2 Activation Function. The activation of each intermediate node is given as

$$u(p) = \prod_{n=1}^{l} f\left((-1)^{mod(\lfloor \frac{p}{2^{n-1}} \rfloor, \lfloor \frac{p}{2^n} \rfloor)}(\mathbf{w}_{\lfloor \frac{p}{2^n} \rfloor}.\mathbf{x} + \theta_{\frac{p}{2^n}})\right), \tag{2.13}$$

and the activation function $f(.)$ is

$$f(v) = \frac{1}{1 + \exp(-mv)}. \tag{2.14}$$

The value of m is to be selected in such a way that a leaf node representing the true class should be able to produce an activation value greater than a certain threshold, that is,

$$u(p) \geq 1 - \epsilon, \tag{2.15}$$

where p is the activated leaf node representing the true class. Note that if the condition in equation 2.15 holds then the total activation of the rest of the leaf nodes is less than ϵ by claim 1. For example, ϵ can be chosen as 0.1.

Let us further assume that

$$\min_i |\mathbf{w}_i'\mathbf{x} + \theta| = \delta, \tag{2.16}$$

that is, δ is a margin between a pattern and the closest decision hyperplane. In that case, the minimum activation of a maximally activated leaf node representing the true class is

$$u_p(\mathbf{x}) \leq \left(\frac{1}{1 + \exp(-m\delta)}\right)^l, \tag{2.17}$$

considering that the pattern $\mathbf{x}$ satisfies all conditions governed by all hyperplanes $\mathbf{w}_i$ on the path from root to the leaf node p,

$$(\mathbf{w}_i'\mathbf{x} + \theta) lr(i, p) > 0. \tag{2.18}$$

In that case, from equation 2.15,

$$\frac{1}{(1+\exp(-m\delta))^l} \geq 1-\epsilon, \tag{2.19}$$

such that

$$l\log(1+\exp(-m\delta)) \leq -\log(1-\epsilon). \tag{2.20}$$

Assuming $\exp(-m\delta)$ and ϵ to be small, we have

$$l\exp(-m\delta) \leq \epsilon, \tag{2.21}$$

that is,

$$m \geq \frac{1}{\delta}\log(l/\epsilon). \tag{2.22}$$

For example, if we choose $\epsilon = 0.1$ and $\delta = 1$, then $m = \log(10l)$.

2.3 Loss Function. For a two-class problem, we partition the leaf nodes into two categories Ω_1 and Ω_2, corresponding to two different classes, C_1 and C_2. We assign the odd-numbered leaf nodes to Ω_1 and the even-numbered leaf nodes to Ω_2 such that for a trained OADT, it is desired that

$$\begin{aligned} (\mathbf{x} \in C_1) \ \textit{and} \ (p \ \textit{is activated}) \quad &\Rightarrow \quad p \in \Omega_1 \\ (\mathbf{x} \in C_2) \ \textit{and} \ (p \ \textit{is activated}) \quad &\Rightarrow \quad p \in \Omega_2. \end{aligned} \tag{2.23}$$

For a supervised learning machine with a single output, if $g(\mathbf{x})$ is the output for an underlying model $y(\mathbf{x})$, then $y(\mathbf{x})$ can be described as (Amari, 1998)

$$y(\mathbf{x}) = g(\mathbf{x};\mathbf{W}) + \varepsilon(\mathbf{x}), \tag{2.24}$$

where $\mathbf{W}$ is a matrix defining the set of parameters and ε is a gaussian noise such that

$$p(\varepsilon) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left[-\frac{1}{2\sigma^2}(y-g(\mathbf{x};\mathbf{W}))^2\right]. \tag{2.25}$$

The effective loss function derived from the likelihood measure can thus be expressed as

$$E = \frac{1}{2}\sum_{\mathbf{x}}(y-g(\mathbf{x};\mathbf{W}))^2. \tag{2.26}$$

However, in OADT, we do not have any underlying function analogous to $y(\mathbf{x})$ for each leaf node since it is not known a priori which leaf node will be activated for a given input. In the framework of decision trees, it is difficult to assign any desired activation to any particular leaf node. For a given pattern $\mathbf{x}$, we can say that one node from a group of nodes should be activated.

We define an indicator function,

$$y(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in C_1 \\ -1 & \mathbf{x} \in C_2 \end{cases} . \tag{2.27}$$

We model the noise as

$$p(\varepsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2\sigma^2} \left((1-y) \sum_{j \in \Omega_1} u_j(\mathbf{x}; \mathbf{W}) + (1+y) \sum_{j \in \Omega_2} u_j(\mathbf{x}; \mathbf{W}) \right)^2 \right], \tag{2.28}$$

where Ω_1 and Ω_2 are the set of leaf nodes assigned to class 1 and class 2, respectively. Here, $\mathbf{W}$ represents all weight vectors corresponding to all intermediate nodes of OADT. Equation 2.28 tells us that for any sample $\mathbf{x} \in C_1$, if any node from Ω_2 is activated or vice versa, then there is an error.

Considering that $\sum_j u_j(x; \mathbf{W}) = 1$, the objective error measure can be expressed as

$$E = \frac{1}{2} \sum_{\mathbf{x}} \left[y(\mathbf{x}) - \sum_{j \in \Omega_1} u_j(\mathbf{x}; \mathbf{W}) - \sum_{j \in \Omega_2} u_j(\mathbf{x}; \mathbf{W}) \right]^2 \tag{2.29}$$

since $y^2(\mathbf{x}) = 1$ for all $\mathbf{x}$. Considering that $\sum_{j \in \Omega_1} u_j(\mathbf{x}; \mathbf{W}) + \sum_{j \in \Omega_2} u_j(\mathbf{x}; \mathbf{W}) = 1$, the objective error measure is

$$E = \frac{1}{2} \left[\sum_{j \notin \Omega_c} u_j(\mathbf{x}; \mathbf{W}) \right]^2 , \tag{2.30}$$

where Ω_c is the set of leaf nodes assigned for class c.

2.4 Learning Rule. Following steepest gradient descent, we have

$$\begin{aligned}\Delta \mathbf{w}_i &= -\eta \left[\sum_{j \notin \Omega_c} u_j(\mathbf{x}; \mathbf{W})\right] \sum_{j \notin \Omega_c} \frac{\partial u_j(\mathbf{x}; \mathbf{W})}{\partial \mathbf{w}_i} \\ \Delta \theta_i &= -\eta \left[\sum_{j \notin \Omega_c} u_j(\mathbf{x}; \mathbf{W})\right] \sum_{j \notin \Omega_c} \frac{\partial u_j(\mathbf{x}; \mathbf{W})}{\partial \theta_i},\end{aligned} \tag{2.31}$$

where Ω_c is either Ω_1 or Ω_2 depending on sample belonging to Class 2 or Class 1 respectively.

The activation of a leaf node j can in general be expressed as $u_j(\mathbf{x}; \mathbf{W}) = v_{1j} \cdots v_{ij} \cdots v_{\lfloor j/2 \rfloor j}$ where $1, \ldots, i, \ldots, \lfloor j/2 \rfloor$ are the nodes on the path from the root to the leaf node j, and

$$v_{ij} = f((\mathbf{w}_i' \mathbf{x} + \theta_i) lr(i, j)) \tag{2.32}$$

is the contribution of node i in the activation of the leaf node j. Therefore, from equation 2.31,

$$\begin{aligned}\Delta \mathbf{w}_i &= -\eta m \left[\sum_{j \notin \Omega_c} u_j(\mathbf{x}; \mathbf{W})\right] \left[\sum_{j \notin \Omega_c} u_j(\mathbf{x}; \mathbf{W})(1 - v_{ij}) lr(i, j)\right] \mathbf{x} \\ \Delta \theta_i &= -\eta m \left[\sum_{j \notin \Omega_c} u_j(\mathbf{x}; \mathbf{W})\right] \left[\sum_{j \notin \Omega_c} u_j(\mathbf{x}; \mathbf{W})(1 - v_{ij}) lr(i, j)\right].\end{aligned} \tag{2.33}$$

Since we impose that $\|\mathbf{w}_i\| = 1$ for all i, we have $\mathbf{w}_i' \Delta \mathbf{w}_i = 0$ for a small η, that is, $\Delta \mathbf{w}$ is normal to $\mathbf{w}$. Further, from equation 2.33, we observe that $\Delta \mathbf{w}$ changes in the direction of $\mathbf{x}$ for steepest descent. Taking the projection of $\mathbf{x}$ on to the normal hyperplane of $\mathbf{w}$, we get

$$\Delta \mathbf{w} \propto (\mathbf{I} - \mathbf{w}\mathbf{w}')\mathbf{x}. \tag{2.34}$$

The modified learning rule is therefore given as

$$\begin{aligned}\Delta \mathbf{w}_i &= -\eta m U_c z_{ic} (\mathbf{I} - \mathbf{w}_i \mathbf{w}_i') \mathbf{x} \\ \Delta \theta_i &= -\eta m U_c u l_{ic},\end{aligned} \tag{2.35}$$

where

$$U_c = \sum_{j \notin \Omega_c} u_j(\mathbf{x}; \mathbf{W}) \tag{2.36}$$

and

$$z_{ic} = \sum_{j \notin \Omega_c} u_j(\mathbf{x}; \mathbf{W})(1 - v_{ij}(\mathbf{x}))lr(i, j). \tag{2.37}$$

We select η to ensure steepest descent such that the residual error with first-order approximation goes to zero, that is,

$$\sum_{j \notin \Omega_c} \Delta u_j(\mathbf{x}; \mathbf{W}) = -U_c(\mathbf{x}). \tag{2.38}$$

Expanding the first-order approximation of Δu_j, from equation 2.35,

$$\Delta u_j = -\eta m^2 \sum_{i \in D} [U_c z_{ic} u_j(\mathbf{x}; \mathbf{W}) \times (1 - v_{ij}(\mathbf{x}))lr(i, j)(1 + \|\mathbf{x}\|^2 - (\mathbf{w}'\mathbf{x})^2)], \tag{2.39}$$

where D is the set of intermediate nodes. Therefore,

$$\sum_j \Delta u_j = -\eta m^2 U_c \sum_{i \in D} z_{ic}^2 (1 + \|\mathbf{x}\|^2 - (\mathbf{w}'\mathbf{x})^2). \tag{2.40}$$

From equation 2.38,

$$\eta = \frac{1}{m^2 \sum_{i \in D} z_{ic}^2 (1 + \|\mathbf{x}\|^2 - (\mathbf{w}'\mathbf{x})^2)}. \tag{2.41}$$

In this analysis, we assumed the same rate for updating $\mathbf{w}$ and θ, although $\mathbf{w}$ is updated with an additional constraint of $\|\mathbf{w}\| = 1$. In order to make it more flexible, we can have different learning rates as η and $\lambda\eta$ for updating $\mathbf{w}$ and θ, respectively. The overall learning rule is therefore given as

$$\begin{aligned} \Delta \mathbf{w}_i &= -\alpha U_c z_{ic} (\mathbf{I} - \mathbf{w}_i \mathbf{w}_i') \mathbf{x} \\ \Delta \theta_i &= -\lambda \alpha U_c z_{ic}, \end{aligned} \tag{2.42}$$

where

$$\alpha = \frac{1}{m \sum_{k \in D} z_{kc}^2 (\lambda + \|\mathbf{x}\|^2 - (\mathbf{w}'\mathbf{x})^2)}. \tag{2.43}$$

It can be noted here that near the optimal point in the parameter space, the value for z_{kc} becomes very small, and as a result, the steepest descent can take relatively larger steps. In order to take care of this problem, we define the denominator of the learning rule as

$$\alpha = \frac{1}{m \left(1 + \sum_{k \in D} z_{kc}^2 (\lambda + \|\mathbf{x}\|^2 - (\mathbf{w}'\mathbf{x})^2)\right)}. \tag{2.44}$$

It is interesting to observe the behavior of the learning rule from equation 2.42. The changes in the weights are always modulated by the total undesired response of the leaf nodes, that is, U_c, which is the total activation of the leaf nodes not corresponding to class c (i.e., nodes not belonging to Ω_c) for a sample coming from class c equation 2.36. The factor z_{ic} denotes the contribution of the leaf nodes that are descendants of the node i and do not belong to Ω_c (see equation 2.37). If a leaf node j not corresponding to class c (see equation 2.37) is on the right path of node i, then j contributes negatively to z, and in effect it is a positive contribution in the learning rule in equation 2.42. This is due to the fact that if the activation of the left child of a node increases, then that of the right child always decreases (claim 1), and vice versa. Also looking at the expression for z_{ic}, equation 2.37, it appears that z is modulated by $(1 - v)$, where v is the contribution of that node in the resultant activation of the leaf nodes. In other words, if a node is highly active, then it is penalized more or the weight parameters are adjusted by a relatively larger step than if the node is less active, that is, a highly active intermediate node is taken to be more responsible for producing the erroneous responses in its descendants if the descendants are found to be erroneous in their activities. Observing the learning rule in equation 2.42, we find that every intermediate node can adjust its parameter values regardless of other nodes so long as it can observe the activation values in the leaf nodes. This is unlike the class of backpropagation learning algorithms as used in the feedforward networks where there is an inherent dependency of the lower-layer nodes on the higher-layer nodes for learning due to error propagation. However, in OADT learning (see equations 2.42 and 2.44), since z_{ic} represents the accumulated effect of all descendant leaf nodes of node i, it ensures the accumulation of activities from a larger number of leaf nodes as i goes up the hierarchy. Thus, the coarse level changes in the parameter values occur at the higher layers (the root node and the nodes closer to the root), and the finer details are captured at the lower layers as it comes closer to the leaf nodes.

3 Simulation and Experimental Results

We simulated OADT in MATLAB 5.3 on Pentium III machine. We experimented with both synthetic and real-life data sets, and here we demonstrate the performance of OADT in both the cases.

3.1 Protocols. We trained OADT in the online mode. The entire batch of samples is repeatedly presented in the online mode to OADT, and we call the number of times the batch of samples presented to an OADT the number of epochs. If the size of the data set is large (i.e., data density is high), then it is observed that OADT takes fewer epochs to converge, and for relatively smaller size data sets, OADT takes a larger number of epochs. On average, we found that OADT converges near its local optimal solution

within 10 to 20 epochs, although the required number of epochs increases with the depth of the tree. We report all results for 200 epochs.

The performance of OADT depends on the depth (l) of the tree and the parameter m. We have chosen the parameters ϵ and δ (see equation 2.22) as 0.1 and 1, respectively, that is, $m = \log(10l)$. Since m is determined by l in our setting, the performance is solely dependent on l. We report the results for different depths of the tree. We normalize all input patterns such that any component of $\mathbf{x}$ lies in the range $[-1, +1]$. Note that it is not always required to normalize the input within this specified range. However in that case, the parameter λ (see equation 2.42) needs to be chosen properly, although we observed that the performance is not very sensitive to variations in the choice of λ. We have chosen λ as unity in the case of normalized input values.

In order to test the performance of OADT with the test samples, we provide an input pattern $\mathbf{x}$ and obtain the activation of all leaf nodes. We find the class label corresponding to the maximally activated leaf node and assign the same class label to the test sample. In all experiments, we assigned the odd-numbered leaf nodes to class 1 (Ω_1 is the set of odd-numbered leaf nodes) and even-numbered leaf nodes to class 2.

3.2 Real-Life Data. We experimented with different real-life data sets available in the UCI machine learning repository (Merz & Murphy, 1996). We computed the 10-fold cross-validation score for each data set. The experimental protocols for OADT are exactly as described in section 3.1. Table 1 demonstrates the performance of OADT. As a comparison, we report the classification performance obtained with the HME for each data set. We implemented the HME using the software available on the web (Martin, n.d.), which is also cited in the Bayes net toolbox (Murphy, 2001, 2003). We used 100 epochs for HME with the EM algorithm, although it converges within 30 epochs for every data set. Table 1 demonstrates that OADT performs better (sometimes marginally) than HME. We also implemented a multilayer perceptron (MLP) network with different number of hidden nodes using the WEKA software package (Garner, 1995; Witten & Frank, 2000) in Java. We observed that both HME and OADT outperform MLP in terms of classification score for each data set. Jordan and Jacobs (1993) reported that HME performs better than MLP in most cases in terms of classification score. Also, for the data sets having highly nonlinear structure (e.g., liver disease data), MLP often gets stuck to local minima if the parameters (such as learning rate and the momentum factor) are not properly tuned. With different experiments, we obtained a maximum score of 57.97% for this data set using an MLP with 1 hidden layer and 15 hidden nodes. Possibly an increase in the number of hidden layers may increase the score. We therefore do not report the classification score obtained with MLP separately. In addition to comparing with HME using the EM algorithm, we compare the performance of OADT with certain batch-mode algorithms, including decision tree, k-nearest neighbor, naive Bayes, support vector machine, and bagged

Table 1: Ten-Fold Cross-Validation Scores of OADT on Four Real-Life Data Sets for Three Depths of 3, 4, and 5.

Data Description and Model		Data Set			
		Breast Cancer (Diagnostic)	Breast Cancer (Prognostic)	Liver Disease (Bupa)	Diabetes (Pima)
Number of features		30	32	6	8
Number of instances		569	198	345	768
HME	$Depth = 3$	92.98	73.11	63.42	69.78
	$Depth = 4$	92.44	70.17	66.47	68.87
	$Depth = 5$	93.15	74.56	65.81	69.78
	$Depth = 6$	94.90	71.56	59.14	69.00
	$Depth = 7$	94.55	68.72	64.86	71.21
OADT	$Depth = 3$	97.37	77.78	66.67	76.69
	$Depth = 4$	97.19	76.89	66.24	75.39
	$Depth = 5$	96.49	77.78	64.86	74.35
C4.5	Score	92.44	76.77	66.38	74.09
	$\|D\|$	25	19	51	43
k-NN		96.13	72.22	64.35	70.44
SVM		97.19	78.28	70.43	76.69
Naive Bayes		93.15	66.67	55.94	75.78
Bagging C4.5		94.73	80.81	72.46	74.87
AdaBoost C4.5		95.96	75.76	67.54	71.35

Notes: The data sets are available in the UCI machine learning repository (Merz & Murphy, 1996). A comparison with HME is also provided for different depths. Comparison with different batch-mode algorithms are also shown. $|D|$ denotes the number of decision nodes generated by the decision tree C4.5. For k-NN classifier, the value of k is set automatically by the leave-one-out principle. SVM uses the cubic polynomial kernels. The classifiers k-NN, SVM, naive Bayes, C4.5 (J4.8), and bagged and boosted trees are available in the WEKA software package.

and boosted decision trees. We implemented all these classification tools using the WEKA software package (Garner, 1995; Witten & Frank, 2000) in Java. In the case of the decision tree, we used C4.5, which is available as J4.8 in the WEKA package. In the case of k-NN algorithms, the value of k is automatically set by the leave-one-out principle. In the case of SVM, we chose cubic polynomial kernels. We observe that OADT outperforms the first three classification algorithms in terms of classification score. However, SVM performs best for two data sets and the bagged trees perform best for two other data sets as compared to all other algorithms. In this context, it may be mentioned that OADT learns strictly in the online mode, and it performs better than its online counterparts. Overall, in the paradigm of online learning, we find that OADT performs reasonably well.

3.3 Synthetic Data. It is well known that parity problems are difficult to deal with in the standard feedforward neural networks. Various investigations (Lavretsky, 2000; Hohil, Liu, & Smith, 1999) are available in the

Table 2: Performance of OADT for n-Parity Problem with Different Amounts of Gaussian Noise.

Task		Recognition score (%) of OADT of Depth					Bayes Score
Dimension	SD	2	3	4	5	6	(Theoretical)
2-parity	0.2	97.40	97.90	98.10	98.40	98.10	98.77
	0.3	88.30	88.20	89.50	89.30	88.70	90.90
	0.4	77.90	76.40	79.90	81.30	81.30	81.10
3-parity	0.2	82.40	96.30	97.00	95.90	96.30	98.16
	0.3	70.40	82.10	83.30	81.50	84.40	86.99
	0.4	57.40	67.20	75.00	74.50	71.80	74.53
4-parity	0.2	56.55	71.43	84.23	89.78	96.33	97.56
	0.3	55.16	65.77	79.07	78.67	83.04	83.45
	0.4	53.27	61.90	62.20	66.57	68.75	69.35
5-parity	0.2	62.30	65.73	74.60	83.17	87.70	96.97
	0.3	50.60	56.55	62.40	67.24	71.37	80.26
	0.4	52.92	57.06	58.97	61.59	64.11	65.26

Notes: The dimensionality n varies from 2 to 5, and the standard deviation of gaussian noise varies from 0.2 to 0.4. It is observed that the OADT performance gracefully degrades with noise.

literature that address solving N-parity problems in connectionist frameworks and provide the design of different architectures to deal with this class of problems. In this section, we show that OADT is also able to handle the gaussian parity problems and provide reasonable classification score. Table 2 demonstrates the performance of OADT for gaussian parity problems where we generated 1000 data points in a unit hypercube and perturbed them with gaussian noise. We show the classification results for different sizes of the gaussian kernels along with the corresponding theoretical upper bound (Bayes risk in the appendix). We find that in most cases, OADT performs close to the theoretical optimum. In the case of higher dimension, the performance falls short of the theoretical bound due to the decrease in the data density. For example, in the case of the 5-parity problem, there are 32 corners of the hypercube, and we have only 31 points (approximately) on an average at each corner. Decision trees are usually not used for such problems because it is difficult to classify such patterns using decision trees without smart look-ahead operators; however, decision trees prove to be very suitable in real-life scenario for their interpretability.

4 Discussion

4.1 Issues of Online Learning. We have used the steepest gradient descent algorithm for training OADT, and adjusted the learning rate parameter by observing the error that needs to be reduced at each step. In the updating rule, since we rely on only the steepest descent, we do not explicitly incor-

porate any factor to explicitly distinguish the differening priors; rather, the weights are updated depending on the frequency of different class labels. In order to incorporate the probability models, we need to compute an inverse Hessian matrix as used in the online version of the hierarchical mixture of experts (Jordan & Jacobs, 1993, 1994), where the inverse covariance matrix (Ljung & Söderström, 1986) is computed, and in the natural gradient descent algorithm (Amari, 1998; Amari, Park, & Fukumizu, 2000), where the inverse Fisher information matrix is computed. Since it is difficult to derive the inverse of a matrix incorporating the statistical structure in the online mode, a stochastic approximation is made using a decay parameter. However, in this class of algorithms, we need to adjust two learning rate parameters as described in Amari et al. (2000). In our learning algorithm, we do not require any explicit learning rate parameter to be defined by the user. Only the depth of the tree needs to be specified before training. Nonetheless, the performance of OADT can possibly be improved by considering smarter gradient learning algorithms, which constitutes a scope for further study.

During the training of OADT, we normalized the data set so that the maximum and minimum values of each component of the input patterns are within the range of $[-1, 1]$ (see section 3.1). We did the normalization in order to select $\lambda = 1$ (see equation 2.42). However, in the case of the streaming data set, it may not be possible to know the maximum and minimum values that the attributes can take, and thus sometimes normalization may not be possible, although an educated guess can work in this case. However, in general, λ can be selected greater than unity also in order to enforce that the step size for parameter θ is larger than that for $\mathbf{w}$, although it may be difficult to derive any theoretical guideline for selecting θ. In order to avoid this situation for a general case, the learning in equation 2.42 can be performed in two steps, analogous to the learning as presented in Basak (2001). The weight vector $\mathbf{w}$ can be updated first and then θ can be updated based on the residual error. However, in the case of the two-step learning algorithm, the responses for the leaf nodes are to be computed twice for the same pattern. The two-stage learning algorithm can be formulated as follows. Update $\mathbf{w}$ without changing θ based on the response of the leaf nodes as

$$\Delta \mathbf{w}_i = -\frac{U_c z_{ic} (\mathbf{I} - \mathbf{w}_i \mathbf{w}_i') \mathbf{x}}{m \left(1 + \sum_{k \in D} z_{kc}^2 (\|\mathbf{x}\|^2 - (\mathbf{w}'\mathbf{x})^2)\right)}. \tag{4.1}$$

After updating $\mathbf{w}$ for all intermediate nodes (which can be performed simultaneously), recompute the activation of the leaf nodes for the same pattern $\mathbf{x}$. From the recomputed activation values of the leaf nodes, update θ as

$$\Delta \theta_i = -\frac{U_c z_{ic}}{m(1 + \sum_{k \in D} z_{kc}^2)}. \tag{4.2}$$

Note that in the two-stage training algorithm for OADT, the requirement for selecting the parameter λ, equation 2.42, is not required.

By the inherent nature of OADT, it can adapt to a changing environment since we do not freeze the learning during the training process. Our learning algorithm does not depend on any decreasing learning rate parameter, and therefore the changing characteristics of a data set are reflected in the changes in the parameters of the intermediate nodes. This fact enables OADT to behave adaptively so long as the depth of the tree does not demand any changes.

We have used an OADT of fixed depth for training; however, it can be grown by observing the performance. If the performance of the tree (such as the decrease in error rate or some other measure) is not satisfactory with a certain depth, then it can be increased by creating child nodes to the existing leaf nodes without disturbing the learned weight vectors. The process is analogous to growing neural networks, that is, adding hidden nodes to a feedforward network. However, we have not yet studied the performance of a growing OADT. It constitutes an area for future study. Similarly, for obtaining better generalization performance, it may be necessary to prune the tree selectively, which is also an area for further study. The depth of an OADT approximately depends on the number of convex polytopes that can describe the class structure. Analogous to model selection techniques in neural networks, the model selection for OADT can be investigated in the future.

4.2 Behavior of the Model. In Figure 1, we illustrate the different decision regions generated by OADT while trained on a two-class problem. For multidimensional patterns, it is difficult to visualize the regions formed, and therefore we show the behavior of OADT for only two attributes. We generated data from a mixture of bivariate gaussian distribution such that the classification task is highly nonlinear. In Figure 1, we observe the regions that are obtained from OADT after 50 iterations for different depths from one to six. With the increase in the depth of OADT, the complexity of the regions increases. However, we observe that after a certain depth, the complexity of the regions does not change significantly. It indicates an interesting fact that for this particular problem, the complexity of the decision region has a saturating tendency after a certain depth of OADT, which shows that even if the depth of OADT is increased beyond the desired value, OADT does not overfit, which may provide good generalization capability. We have illustrated the decision space for only two attributes; however, it requires further investigation to understand whether this saturating tendency is true in general.

Conventional decision tree learning algorithms are more elegant in inducing axis-parallel trees (Breiman et al., 1983; Quinlan, 1993) where at each decision node, a decision is induced based on the value of a particular attribute. An OADT by its very nature induces oblique decision trees, since

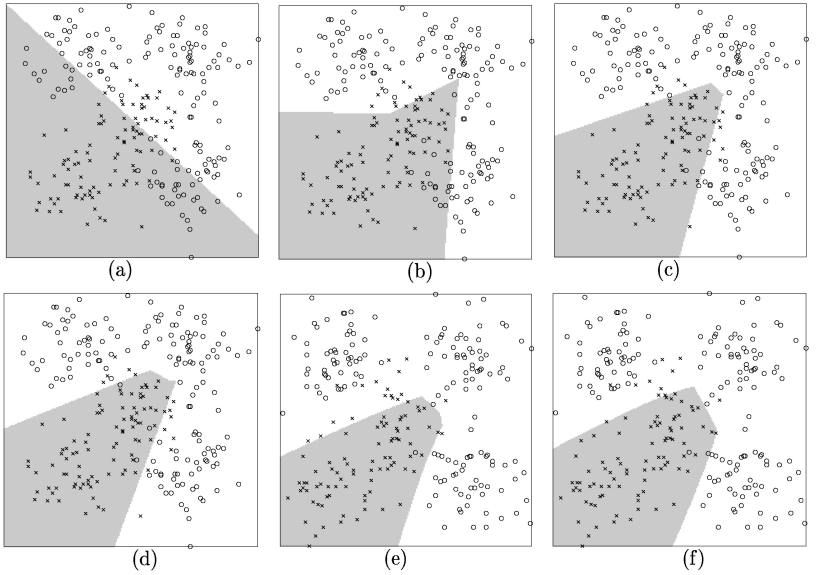

Figure 1: The regions obtained by OADT for (a) depth = 1, (b) depth = 2, (c) depth = 3, (d) depth = 4, (e) depth = 5, and (f) depth = 6. Two different class labels are marked as x and o.

if the steepness of the activation function (m) of the intermediate nodes is increased, then the OADT starts behaving like an oblique decision tree, and in the limit of $m \to \infty$, it is exactly an oblique decision tree. Thus, OADT can be viewed as interpretable as the oblique decision trees are. However, in comparison to the existing axis-parallel decision trees (such as C4.5), OADT falls short in terms of interpretability.

In feedforward networks, usually there are hidden nodes, which cannot observe the input and the output. The weight updating for each hidden node depends on the error propagated from the output layer. In the case of OADT, each intermediate node can directly compute the change in weight from the response of the leaf nodes. Thus, the weight updating of each intermediate node can be made independently and simultaneously with other intermediate nodes, unlike feedforward networks, where there is an inherent dependency on the error propagation through layers.

The memory required to store an OADT is $O(d * (n + 1))$, where d is the number of intermediate nodes and n is the dimension of the input space. In the case of a two-layer perceptron with one output node, the required storage is $O(h * (n + 1) + h + 1)$, where h is the number of hidden nodes, considering that all hidden nodes are connected to all input nodes. Since $d = 2^{l-1}$, where l is the depth of OADT, the storage required for OADT

can be more than a feedforward network if $d > h$. The hyperplane set by one intermediate node in OADT interacts only with that created by the descendants of that node in forming the decision regions. In feedforward networks, all decision hyperplanes formed by the hidden nodes can interact with each other. Thus feedforward networks can in general provide a more compact representation of the decision region as compared to OADT.

4.3 Multiclass Input Label. Here we provided an algorithm for learning a two-class problem. Any k-class problem can be handled using $(k-1)$ trained classifiers able to deal with two-class problems. However, it is always desirable to have a single classifier that is able to classify multilabel input data. In the case of k-class problem, we can modify the network into a k-ary tree such that the node i sends an activation g_{il} to its lth child ($l \in \{1, \ldots, k\}$) (in equations 2.1 and 2.2), where

$$g_{il}(\mathbf{x}) = \frac{f(\mathbf{w_{il}}.\mathbf{x} + \theta_{il})}{\sum_{j=1}^{k} f(\mathbf{w_{ij}}.\mathbf{x} + \theta_{ij})}, \tag{4.3}$$

and the activation of the lth child node is $u(i).g_{il}(\mathbf{x})$. The activation is analogous to that of the gating networks used in the framework of mixture of experts (Jordan & Jacobs, 1993, 1994), although the purpose here is to make the activation flow from the root node to the leaf nodes. The rest of the learning algorithm can be derived in exactly the same way as in the case of a two-class problem, where we assign each leaf node i a class index c where $c = mod(i, k)+1$ if the nodes in the tree are numbered in a breadth-first order. However, using such networks requires extra storage space as compared to the two-class problems. In general, if the depth is d, then we require $k^d + 1$ nodes in a k-ary OADT for a k-class problem and $(k-1)(2^d + 1)$ nodes if we employ $k-1$ binary OADTs. Since OADT operates in a top-down manner, it does not have an analogous way of handling multiple class labels with extra output nodes as in the case of HME and the MLP.

5 Conclusions

We presented the OADT, a classification method that maintains the structure of a tree and employs the gradient descent learning algorithm like neural networks for supervised learning in the online mode. OADT is useful in the case of streaming data and limited memory situation where data sets cannot be stored explicitly. OADT is similar to the HME framework in the sense that the latter also employs a tree-structured model. However in OADT, the leaf nodes represent the decision instead of the root node, which is different from the HME framework. Thus, in OADT, the decision region is always explicitly formed by the collective activation of a set of leaf nodes. An analog of the difference between OADT and HME can be drawn from the difference between the divisive and agglomerative clustering in the un-

supervised framework, where in the first case, the patterns are successively partitioned down the hierarchy, whereas in the second case, they are successively agglomerated into larger clusters up the hierarchy. Since in OADT, the decision is represented by the collective activation of the leaf nodes, the objective error measure is different, and therefore new learning learning rules are derived to train OADT in the online mode. We also provided guidelines for selecting the activation function of each node of an OADT for a given depth, and therefore the performance of our model is dependent on only one parameter—the depth of the tree. We also observe that if we increase the depth of OADT, the complexity of the decision region generated by OADT almost saturates after a certain depth, an indication that the model may not suffer much from overfitting even if the depth is increased gradually.

We used steepest gradient descent learning in OADT as opposed to the EM framework in HME, although it is possible to employ gradient descent online learning in the latter framework. We used steepest gradient descent learning in our model, which is analogous to the class of gradient descent algorithms used in feedforward neural networks. In formulating the learning algorithms for OADT, we do not view the activation of the nodes in terms of the posteriors as performed in the HME where multinomial logit probability model (Jordan, 1995) is used to interpret the activation of the nodes in terms of class posteriors. However, analogous to the behavior feedforward neural networks where the activation of the output nodes corresponds to class posteriors (Barron, 1993) after convergence, further investigation can be carried out to interpret the activation of the leaf nodes of OADT in a statistical framework. Also, smarter gradient descent algorithms apart from only the steepest descent can be incorporated in OADT for improved performance. With the real-life data sets, we observe that OADT performs better than HME. However, it requires further exploration to determine if OADT will provide better scores than HME in general. Further exploration in a possible statistical framework may provide better insight into the behavior of OADT and its relationship with HME.

Appendix: Bayes Risk for Parity Problem

For a two-class (C_1 and C_2) symmetric gaussian parity problem, the Bayes risk is given as

$$R = \int_{P(C_2|\mathbf{x}) > P(C_1|\mathbf{x})} p(\mathbf{x}|C_1) d\mathbf{x}, \tag{A.1}$$

where $p(\mathbf{x}|C_1) = \frac{1}{N} \sum_i \mathcal{N}(\mu_i, \Sigma)$ is the conditional distribution with μ representing the corners of a unit n-dimensional hypercube, $\Sigma = \sigma^2 \mathbf{I}$ being the covariance matrix, and $N = 2^{n-1}$ being the number of corners of the hypercube allocated to class C_1.

The conditional distribution of a class is given as

$$p(\mathbf{x}|C_1) = \frac{1}{N}\sum_i \mathcal{N}(\mu_i, \Sigma), \tag{A.2}$$

where μ represents the corners of a unit hypercube such that for class C_1,

$$\text{mod}\left(\sum_j \mu_{ij}, 2\right) = 0 \tag{A.3}$$

for all i, and $\Sigma = \sigma^2\mathbf{I}$ is the covariance matrix. Each attribute is considered to be independent of the other. The Bayes error for the patterns generated by the first kernel $(0, 0, \ldots, 0)$ when the first component of the patterns exceeds $1/2$ is given as

$$\begin{aligned} r = &\frac{1}{N}\int_{1/2}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x_1^2}{2\sigma^2}} dx_1 \\ &\times \int_{-\infty}^{1/2} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x_2^2}{2\sigma^2}} dx_2 \int_{-\infty}^{1/2} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x_3^2}{2\sigma^2}} dx_3 \cdots \end{aligned} \tag{A.4}$$

where $N = 2^{n-1}$, assuming the independence of the components,

$$r = \frac{1}{N}\left(\int_{1/2}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} dx\right)\left(\frac{1}{N}\int_{-\infty}^{1/2} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} dx\right)^{n-1}. \tag{A.5}$$

Computing the integrals,

$$r = \frac{1}{N}\cdot\frac{1}{2^n}(1-K)(1+K)^{n-1}, \tag{A.6}$$

where $K = erf(\frac{1}{2\sqrt{2}\sigma})$. If we consider all possible ways by which one component of a pattern generated from the kernel at $(0, 0, \ldots, 0)$ can exceed the value $1/2$, we get the risk as

$$r_1 = \frac{1}{2N^2}\begin{pmatrix} n \\ 1 \end{pmatrix}(1-K)(1+K)^{n-1}. \tag{A.7}$$

Similarly, if we consider all possible ways by which three components of a pattern generating from the kernel at $(0, 0, \ldots, 0)$ can exceed the value $1/2$,

we get the risk as

$$r_2 = \frac{1}{2N^2}\begin{pmatrix} n \\ 3 \end{pmatrix}(1-K)^3(1+K)^{n-3}, \tag{A.8}$$

and so on.

Summing up the factors $r_1, r_2, \ldots,$ we get the risk for patterns generated from the kernel at $(0, 0, \ldots, 0)$ exceeding the value $1/2$ by an odd number of components as

$$r = \frac{1}{2N^2}\left[\begin{pmatrix} n \\ 1 \end{pmatrix}(1-K)(1+K)^{n-1} + \begin{pmatrix} n \\ 3 \end{pmatrix}(1-K)^3(1+K)^{n-3} + \cdots\right]. \tag{A.9}$$

After simplification,

$$r = \frac{1}{2N}(1-K^n). \tag{A.10}$$

For N such kernels, the total Bayes risk is given as

$$R = \frac{1}{2}\left(1 - erf^n\left(\frac{1}{2\sqrt{2}\sigma}\right)\right). \tag{A.11}$$

It can be noted from equation A.11 that with the same variance of the kernel distribution (i.e., the same noise), the Bayes error increases with the increase in dimension.

Acknowledgments

I gratefully acknowledge the anonymous reviewers for their comments, which improved this article considerably.

References

Albers, S. (1996). *Competitive online algorithms* (Tech. Rep. No. BRICS Lecture Series LS-96-2). Aarhus, Denmark: University of Aarhus.

Amari, S. I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, *10*(2), 251–276.

Amari, S. I., Park, H., & Fukumizu, K. (2000). Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, *12*(6), 1399–1409.

Barron, A. R. (1993). Universal approximation bounds for superposition of a sigmoidal function. *IEEE Transactions on Information Theory*, *39*, 930–945.

Basak, J. (2001). Learning Hough transform: A neural network model. *Neural Computation*, *13*, 651–676.

Bennett, K. P., Wu, D., & Auslender, L. (1998). *On support vector decision trees for database marketing* (Tech. Rep. No. RPI Math Report 98-100). Troy, NY: Rensselaer Polytechnic Institute.

Boz, O. (2000). *Converting a trained neural network to a decision tree DecText—decision tree extractor.* Unpublished doctoral dissertation, Lehigh University.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1983). *Classification and regression trees.* New York: Chapman & Hall.

Brodley, C. E., & Utgoff, P. E. (1995). Multivariate decision trees. *Machine Learning*, *19*, 45–77.

Chien, J., Huang, C., & Chen, S. (2002). Compact decision trees with cluster validity for speech recognition. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing* (pp. 873–876). Orlando, FL: IEEE Signal Processing Society.

Cho, Y. H., Kim, J. K., & Kim, S. H. (2002). A personalized recommender system based on web usage mining and decision tree induction. *Expert Systems with Applications*, *23*, 329–342.

Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis.* New York: Wiley.

Duda, R., Hart, P., & Stork, D. (2001). *Pattern classification (2nd ed).* New York: Wiley.

Durkin, J. (1992). Induction via ID3. *AI Expert*, *7*, 48–53.

Fayyad, U. M., & Irani, K. B. (1992). On the handling of continuous-values attributes in decision tree generation. *Machine Learning*, *8*, 87–102.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *Annals of Statistics*, *19*, 1–141.

Friedman, J. H., Kohavi, R., & Yun, Y. (1996). Lazy decision trees. In H. Shrobe & T. Senator (Eds.), *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference* (pp.S 717–724). Menlo Park, CA: AAAI Press.

Garner, S. (1995). Weka: The Waikato environment for knowledge analysis. In *Proc. of the New Zealand Computer Science Research Students Conference* (pp. 57–64). Hamilton, New Zealand: University of Waikato.

Golea, M., & Marchand, M. (1990). A growth algorithm for neural network decision trees. *Europhysics Letters*, *12*, 105–110.

Haykin, S. (1999). *Neural networks: A comprehensive foundation.* Upper Saddle River, NJ: Prentice Hall.

Hohil, M. E., Liu, D., & Smith, S. H. (1999). Solving the N-bit parity problem using neural networks. *Neural Networks*, *12*, 1321–1323.

Jain, A. K., Duin, R. P. W., & Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*, 4–37.

Jordan, M. (1995). *Why the logistic function? A tutorial discussion on probabilities and neural networks.* Available online at: http://citeseer.nj.nec.com/jordan95why.html.

Jordan, M. I., & Jacobs, R. A. (1993). *Hierarchical mixtures of experts and the EM algorithm* (Tech. Rep. No. AI Memo 1440). Cambridge, MA: MIT.

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*, 181–214.

Kalai, A., & Vempala, S. (n.d.). *Efficient algorithms for online decision.* Available online at: http://citeseer.nj.nec.com/585165.html.

Lavretsky, E. (2000). On the exact solution of the parity-N problem using ordered neural networks. *Neural Networks*, *13*, 643–649.

Lee, S.-J., Jone, M.-T., & Tsai, H.-L. (1995). Construction of neural networks from decision trees. *Journal of Information Science and Engineering*, *11*, 391–415.

Ljung, L., & Söderström, T. (1986). *Theory and practice of recursive identification.* Cambridge, MA: MIT Press.

Martin, D. (n.d.). *Hierarchical mixture of experts.* Available online at: http://www.cs.berkeley.edu/~dmartin/software/.

Mehta, M., Agrawal, R., & Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. In P. M. G. Apers, M. Bouzeghoub, & G. Gardarin (Eds.), *Advances in database technology—EDBT'96* (pp. 18–32). Avington, France.

Merz, C. J., & Murphy, P. M. (1996). *UCI repository of machine learning databases* (Tech. Rep.). Irvine: University of California at Irvine.

Murphy, K. (2001). The Bayes net toolbox for Matlab. *Computing Science and Statistics*, *33*, 1–20.

Murphy, K. (2003). *Bayes net toolbox for Matlab.* Available online at: http://www.ai.mit.edu/~murphyk/software/index.html.

Murthy, S. K., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, *2*, 1–32.

Quinlan, J. R. (1993). *Programs for machine learning.* San Francisco: Morgan Kaufmann.

Quinlan, J. R. (1996). Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence*, *4*, 77–90.

Riley, M. D. (1989). Some applications of tree based modeling to speech and language indexing. In *Proc. DARPA Speech and Natural Language Workshop* (pp. 339–352). San Francisco: Morgan Kaufmann.

Salzberg, S., Delcher, A. L., Fasman, K. H., & Henderson, J. (1998). A decision tree system for finding genes in DNA. *Journal of Computational Biology*, *5*, 667–680.

Strömberg, J. E., Zrida, J., & Isaksson, A. (1991). Neural trees—using neural nets in a tree classifier structure. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 137–140). Toronto: IEEE Signal Processing Society.

Utgoff, P. E., Berkman, N. C., & Clouse, J. A. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning*, *29*(1), 5–44.

Witten, I. H., & Frank, E. (2000). *Data mining: Practical machine learning tools and techniques with Java implementations*. San Francisco: Morgan Kaufmann.

Yang, Y., & Pedersen, J. O. (1997). A comparatative study on feature selection in text categorization. In D. H. Fisher (Ed.), *Fourteenth Int. Conference on Machine Learning (ICMl97)* (pp. 412–420). San Francisco: Morgan Kaufmann.

Zamir, O., & Etzioni, O. (1998). Web document clustering: A feasibility demonstration. *SGIR '98: Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval* (pp. 46–54). New York: ACM.

Received August 11, 2003; accepted February 23, 2004.