Online Adaptive Decision Trees: Pattern Classification and Function Approximation

Article *in* Neural Computation · October 2006

DOI: 10.1162/neco.2006.18.9.2062 · Source: PubMed

1 author:

Jayanta Basak

106 PUBLICATIONS **902** CITATIONS

SEE PROFILE

LETTER Communicated by Jerome Friedman

Online Adaptive Decision Trees: Pattern Classification and Function Approximation

Jayanta Basak
basakjayanta@yahoo.com,bjayanta@in.ibm.com
IBM India Research Lab, Indian Institute of Technology, New Delhi 110016, India

Recently we have shown that decision trees can be trained in the online adaptive (OADT) mode (Basak, 2004), leading to better generalization score. OADTs were bottlenecked by the fact that they are able to handle only two-class classification tasks with a given structure. In this article, we provide an architecture based on OADT, ExOADT, which can handle multiclass classification tasks and is able to perform function approximation. ExOADT is structurally similar to OADT extended with a regression layer. We also show that ExOADT is capable not only of adapting the local decision hyperplanes in the nonterminal nodes but also has the potential of smoothly changing the structure of the tree depending on the data samples. We provide the learning rules based on steepest gradient descent for the new model ExOADT. Experimentally we demonstrate the effectiveness of ExOADT in the pattern classification and function approximation tasks. Finally, we briefly discuss the relationship of ExOADT with other classification models.

1 Introduction

Decision trees (Duda, Hart, & Stork, 2001; Durkin, 1992; Fayyad & Irani, 1992; Friedman, 1991; Breiman, Friedman, Olshen, & Stone, 1983; Quinlan, 1993, 1996; Brodley & Utgoff, 1995), well known for their simplicity and interpretability, usually perform hard splits of data sets, and if a mistake is committed in splitting the data set at a node, it cannot be corrected further down the tree. Attempts have been made to address this problem (Friedman, Kohavi, & Yun, 1996); in general, they employ various kinds of look-ahead operators to estimate the potential gain in the objective after more than one recursive split of the data set. Moreover, a decision tree is always constructed in batch mode; the branching decision at each node of the tree is induced based on the entire set of training data. Variations have been proposed for efficient construction mechanisms to aleviate this difficulty (Utgoff, Berkman, & Clouse, 1997; Mehta, Agrawal, & Rissanen, 1996). Online algorithms (Kalai & Vempala, n.d.; Albers, 1996), 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; therefore, even if a node is faulty (or commits 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 adaptivity. 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.

In a recent article (Basak, 2004), we showed that decision trees with a specified depth can be trained in the online adaptive mode (OADT), leading to a better generalization score as compared to conventional decision trees and neural networks. 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 the hierarchical mixture of experts (Jordan & Jacobs, 1993, 1994), where the 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 the hierarchical mixture of experts, 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. Training of OADT was performed by considering the representation of class by an aggregate of leaf node activations, and, correspondingly, the hyperplanes of the intermediate (nonterminal) nodes were updated to reduce an aggregated average error.

OADTs also proved to be more efficient in terms of classification accuracy when compared to the neural networks and the hierarchical mixture of experts. However, the OADTs are seriously bottlenecked by two factors. First, in our previous discussion (Basak, 2004), we outlined how to represent a two-class classification problem using the leaf nodes of an OADT in a top-down manner. However, it was not clear how to represent a multiclass classification problem using a single such network. Second, the OADTs were designed to address the classification task only, and no issues were discussed to address the problem of function approximation.

In this letter, we present an extension of OADT (we call it ExOADT) to handle both multiclass pattern classification and the function approximation problems. Various attempts have been made to enhance decision trees by constructing oblique decision trees (Murthy, Kasif, & Salzberg, 1994) and hybridizing the decision trees with other computational paradigms such as neural networks, support vector machines, and fuzzy set theoretic concepts for the purpose of classification (Bennett, Wu, & Auslender, 1998;

Strömberg, Zrida, & Isaksson, 1991; Golea & Marchand, 1990; Boz, 2000; Janickow, 1998; Suárez & Lutsko, 1999). In almost all of these approaches, the basic formalism of decision trees is maintained in the sense that the data are recursively partitioned locally at each nonterminal node, and finally the class labels are assigned at the leaf nodes. Thus, the overall construction of the decision tree is data driven, and it reduces the bias at the cost of variance. Model-based construction of decision trees (Geman & Jedynak, 2001) has also been attempted in order to optimize the overall structure. Deviations from these data-driven construction of decision trees are the hierarchical mixture of experts (HME) and online adaptive decision tree (OADT). In both cases, instead of local splitting criteria at each nonterminal node, a global objective measure is minimized. In the case of HME, the decisions are hierarchically agglomerated in a bottom-up fashion, and the learning rules based on expectation-maximization (EM) are used. In the case of OADT, decision is made in a top-down manner, and steepest descent rule is used.

In most of the decision tree construction approaches and their applications (Chien, Huang, & Chen, 2002; Cho, Kim, & Kim, 2002; Riley, 1989; Salzberg, Delcher, Fasman, & Henderson, 1998; Yang & Pedersen, 1997; Zamir & Etzioni, 1998), the decision tree–like network is modeled for classification tasks. Decision trees are employed for function approximation mostly in the context of reinforcement learning for value function estimation (Uther & Veloso, 1998; Pyeatt & Howe, 1998; Wang & Dietterich, 1999) where the input space is divided into different regions and at each subspace a local regression is performed. One of the most elegant approaches in this line is treeNet (Friedman, 2001; Friedman, Hastie, & Tibshirani, 1998), where additive expansions of the predicted functions are performed by tree boosting where piecewise constant approximation is performed at a finer granularity. A large number of studies are available in the literature for function approximation based on neural networks, radial basis functions, and additive splines (Girosi, Jones, & Poggio, 1995; Buhmann, 1990; Broomhead & Lowe, 1988; Grimson, 1982; Moody & Darken, 1989; Poggio & Girosi, 1990a, 1990b; Powell, 1987; Poggio, Torre, & Koch, 1985), which are mostly based on regularization theory (Tikhonov & Arsenin, 1977). A comprehensive discussion on these approaches can be found in Girosi et al. (1995). Recently, the function approximation task has been dealt with in the framework of structural risk minimization, namely, support vector regression (Vapnik, 1998; Vapnik, Golowich, & Smola, 1997; Smola & Schölkopf, 1998a, 1998b; Gunn, 1998; Burges, 1998).

In this letter, we show that the extension of the online adaptive decision tree (ExOADT) is able not only to handle the muticlass classification problem but also to provide a robust smooth estimate of the desired functions. ExOADT not only adapts the local hyperplanes in the nonterminal nodes in the online mode, but is also able to smoothly change the structure depending on the data set. The rest of the letter is organized as follows. In section 2,

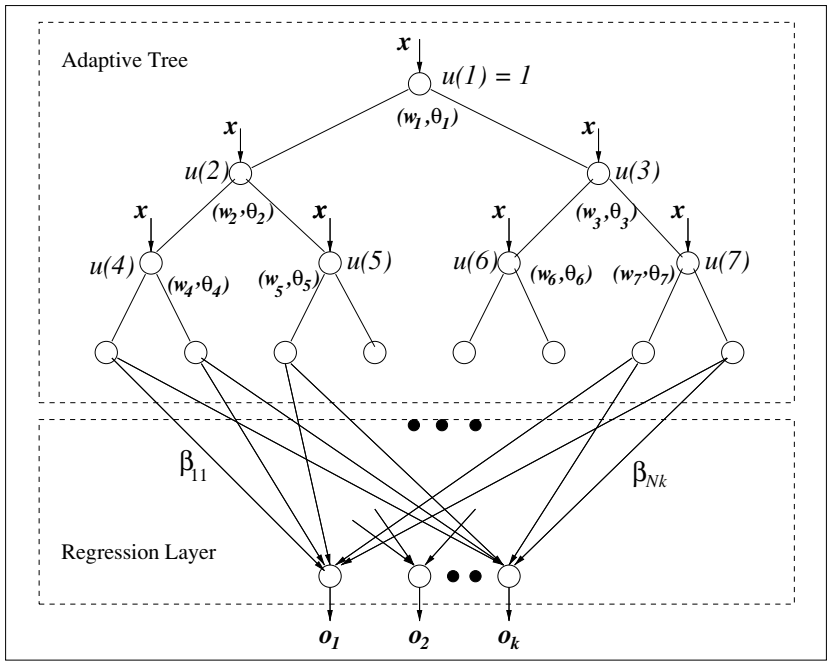

Figure 1: Structure of an extended OADT network. The network consists of two parts: one adaptive decision tree and a regression layer. Each intermediate node of the adaptive decision tree accepts the input pattern vector $\mathbf{x}$ as input, and it activates its two child nodes differentially depending on the embedded sigmoidal function defined by its parameters $(\mathbf{w}, \theta)$. The root node activation $u(1)$ is always unity. The leaf nodes of the adaptive tree are connected to the output layer through an adaptive connection matrix β. The output of the output layer is represented by $o_1, o_2, \ldots, o_k$ respectively.

we describe the model and the associated learning rules for pattern classification and function approximation. We then demonstrate the effectiveness of the model for performing the task of function approximation and pattern classification on certain real-life data sets in section 3. We discuss various aspects of the model in section 4 conclude in section 5.

2 Description of the Model

The structure of the ExOADT is shown in Figure 1. It consists of a complete binary tree of certain depth l with the leaf nodes connected to the output nodes via adaptive weight matrix β. Each nonterminal node i has a local decision hyperplane characterized by a vector $(\mathbf{w}_i, \theta_i)$ with $\|\mathbf{w}_i\| = 1$ such that $(\mathbf{w}_i, \theta_i)$ together has n free variables for an n-dimensional input space.

2.1 Activation of the Nodes. As a convention, we number the nodes in ExOADT in the breadth-first order with the root node numbered as 1; this convention is followed throughout this letter. The activation of two child nodes $2i$ and $2i+1$ of the node i is given as

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

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

where $\mathbf{x}$ is the input pattern, u_i represents the activation of node i, and $f(.)$ represents the local soft partitioning function attributed by the parameter vector $(\mathbf{w}_i, \theta_i)$. We choose $f(.)$ as a sigmoidal function. The activation of the root node is set to unity. Thus, the activation of a leaf node p can be represented as

$$u_p = \prod_{i \in P_p} f\left(lr(i, p)(\mathbf{w}_i . \mathbf{x} + \theta_i)\right). \tag{2.3}$$

Here P_p is the path from the root node to the leaf node p, and $lr(.)$ is an indicator function about whether the leaf node p is on the right or left path of node i such that

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

Considering a sigmoidal activation function $f(.)$, Basak (2004) shows that the sum of the activation of all leaf nodes is always unity provided that the root node has unit activation.

The activation of the output node k, in the case of pattern classification, is given as

$$o_k = f\left(\sum_{j \in \Omega} \beta_{jk} u_j\right), \tag{2.5}$$

where Ω is the set of leaf nodes, β is the connection matrix between the leaf nodes and the output nodes, and $f(.)$ is a sigmoidal activation function to restrict the output in $[0, 1]$. In the case of k-class classification problem, we have k such output nodes. In the case of function approximation, we have only one output node (considering that there is only one predictor variable),

and the output is simply the weighted sum of the leaf node activations:

$$o = \sum_{j \in \Omega} \beta_j u_j. \tag{2.6}$$

Note that this kind of representation is also performing a piecewise linear approximation of the output function. However, due to soft splitting of the input data in the nonterminal nodes, it is not essential that only one leaf node will be activated at a time. Second, due to soft splitting, activation of the leaf nodes makes a smooth transition in the range $[0, 1]$ governed by the fact that (Basak, 2004)

$$\sum_{j \in \Omega} u_j = 1. \tag{2.7}$$

For each nonterminal node, we select the sigmoidal form of activation as

$$f(v) = \frac{1}{1 + \exp(-m_1 v)}, \tag{2.8}$$

where m_1 decides the steepness of the function and it is the same for all nonterminal nodes. Considering the minimum activation of a maximally activated leaf node to be greater than a certain threshold, we derived (Basak, 2004) that

$$m_1 \geq \frac{1}{\delta} \log l/\epsilon, \tag{2.9}$$

where δ is a margin between a point (pattern) and the closest decision hyperplane, and ϵ is a constant such that the minimum activation of a maximally activated node should be no less that $1 - \epsilon$. For example, if we choose $\epsilon = 0.1$ and $\delta = 1$, then $m_1 = \log(10l)$. Note that the parameter m_1 can be learned as part of the supervised training. However, we fix m_1 in such a way that the maximally activated leaf node gets an activation as close to unity as possible with a margin ϵ. This is done in analogy with the classical decision tree where only one leaf node is assigned to a given pattern. For the sigmoidal functions of the output nodes for classification, the steepness parameter m_2 is different from that of the nonterminal nodes, and we always select $m_2 = 1$. The parameter m_2 can also be learned along with β during the supervised training. However, since the output o_k of the kth output node can be expressed as

$$o_k = \frac{1}{1 + \exp(-m_2 \sum_{j \in \Omega} \beta_{jk} u_j)}, \tag{2.10}$$

the parameter m_2 acts as a scaling factor to β. Since we do not impose any restriction on the maximum or minimum value of β (unlike $\mathbf{w}$, where we always have $\|\mathbf{w}\| = 1$), we need to fix m_2 at some value. Equation 2.10 reveals that we can fix m_2 at any value, and β will change accordingly. Here we selected $m_2 = 1$.

2.2 Learning Rule for Pattern Classification. For pattern classification, we define the loss function as the squared error based on the assumption that the model noise introduced on the predictor gaussian in nature (Amari, 1967, 1998) such that

$$E(\mathbf{x}) = \frac{1}{2}\sum_k (y_k(\mathbf{x}) - o_k(\mathbf{x}))^2 \tag{2.11}$$

where $y_k \in \{0, 1\}$ depending on whether the pattern belongs to the kth class. Subject to steepest descent, the parameters are changed as

$$\mathbf{\Delta w}_i = -\eta_{opt}\frac{\partial E}{\partial \mathbf{w}_i} \tag{2.12}$$

$$\Delta\theta_i = -\eta_{opt}\frac{\partial E}{\partial \theta_i} \tag{2.13}$$

$$\mathbf{\Delta}\beta_k = -\eta_{opt}\frac{\partial E}{\partial \beta_k}, \tag{2.14}$$

where η_{opt} is the step size of the steepest descent using line search (Friedman, 2001):

$$\eta_{opt} = \text{argmin}_\eta (E(\mathbf{x}; \mathbf{w} + \mathbf{\Delta w}, \theta + \Delta\theta, \beta + \mathbf{\Delta}\beta). \tag{2.15}$$

Considering that each $\mathbf{w}_i$ has $n - 1$ free variables in an n-dimensional space subject to $\|\mathbf{w}_i\| = 1$, we have

$$\mathbf{\Delta w}_i = -\eta_{opt}\frac{\partial E}{\partial \mathbf{w}_i}(\mathbf{I} - \mathbf{w}_i\mathbf{w}_i^{'}). \tag{2.16}$$

Evaluating the partials, we get

$$\mathbf{\Delta w}_i = \eta_{opt} m_1 q_i (\mathbf{I} - \mathbf{w}_i\mathbf{w}_i^{'})\mathbf{x} \tag{2.17}$$

$$\Delta\theta_i = \lambda\eta_{opt} m_1 q_i \tag{2.18}$$

and

$$\mathbf{\Delta}\beta_j = \eta_{opt} m_2 e_j o_j (1 - o_j)\mathbf{z}. \tag{2.19}$$

The parameter λ is a constant to differentiate the learning rates of $\mathbf{w}$ and θ. Normally the parameter θ depends on the range of the distribution of input attribute values. In our algorithm, we normalize all input attributes in $[-1, 1]$ so that we choose $\lambda = 1$. The error in each output node j is e_j, that is, $e_j = y_j - o_j$. The error at each intermediate (nonterminal) node i is q_i, which is given as

$$q_i = \sum_{j \in \Omega} \delta_j u_j (1 - v_{ij}) lr(i, j), \tag{2.20}$$

where δ is the backpropagated error such that

$$\delta_j = \sum_{k \in O} \beta_{jk} e_k, \tag{2.21}$$

where O is the set of output nodes. Each nonterminal node i locally computes v_{ij}, which is given as

$$v_{ij} = f((\mathbf{w}_i^{'} \mathbf{x} + \theta_i) lr(i, j)). \tag{2.22}$$

The activation of the leaf nodes is represented by the vector $\mathbf{z}$, that is, $\mathbf{z} = [u_j | j \in \Omega]$.

Equations 2.17 and 2.18 reveal that we first compute the error δ backpropagated to the leaf nodes from the output nodes and then compute the local changes in each nonterminal node. Note that the values of q can be computed in each nonterminal node independent of the other nonterminal nodes. Equations 2.17 and 2.18 show that the parameters of a nonterminal node i are updated based on the error only in those leaf nodes that are on either the left path or the right path of i. If a leaf node j is on neither the left nor the right path of node i, then δ_j does not affect the parameter changes in i. Therefore, the root node parameters are affected by all the leaf nodes; that is, the parameters in the root node start capturing the gross behavior of the data in terms of linear separability. As we go down the tree, only the data points in a subspace affect the parameters of the nonterminal nodes. Equation 2.19 is the adaptive updating of the parameters β to obtain the linear regression except the factor $o_j(1 - o_j)$, which is a nonlinear transformation due to the presence of sigmoidal activation.

In order to perform the line search on η to get η_{opt} such that $\eta_{opt} = \text{argmin}_{\eta}(E(\mathbf{x}) + \Delta E(\mathbf{x}))$, we let the first-order approximation of $\Delta E = -E$. Algebraic manipulation (see the appendix) yields

$$\eta_{opt} = \frac{E}{m_2^2 \|\mathbf{z}\|^2 \sum_{k \in O} e_k^2 o_k^2 (1 - o_k)^2 + m_1^2 \sum_{i \notin \Omega} q_i^2 (\lambda + \|\mathbf{x}\|^2 - (\mathbf{w}_i^{'} \mathbf{x})^2)}. \tag{2.23}$$

At a near-optimal point, if E becomes very small, then the denominator in equation 2.23 becomes very small, and the behavior will become unstable. In order to stabilize the behavior of the network, we make

$$\eta_{opt} = \frac{E}{1 + m_2^2 \|\mathbf{z}\|^2 \sum_{k \in O} e_k^2 o_k^2 (1 - o_k)^2 + m_1^2 \sum_{i \notin \Omega} q_i^2 (\lambda + \|\mathbf{x}\|^2 - (\mathbf{w}_i' \mathbf{x})^2)}. \tag{2.24}$$

2.3 Learning Rule for Function Approximation. The learning rule for function approximation is the same as that for classification except that we use linear instead of sigmoidal units in the output. Since there is no sigmoidal function at the output node, we denote the steepness parameter of the nonterminal node transfer function by m. We consider a squared error loss at the output

$$E = \frac{1}{2}(y - o)^2, \tag{2.25}$$

where $o = \sum_{j \in \Omega} \beta_j u_j$. Following steepest descent, we get the learning rules as

$$\Delta \mathbf{w}_i = \eta_{opt} m (y - o) r_i (\mathbf{I} - \mathbf{w}_i \mathbf{w}_i') \mathbf{x} \tag{2.26}$$

$$\Delta \theta_i = \lambda \eta_{opt} m (y - o) r_i \tag{2.27}$$

and

$$\Delta \beta = \eta_{opt} (y - o) \mathbf{z} \tag{2.28}$$

where

$$r_i = \sum_{j \in \Omega} \beta_j u_j (1 - v_{ij}) lr(i, j), \tag{2.29}$$

the activation of leaf nodes $\mathbf{z} = [u_j | j \in \Omega]$, and the error at the output node $e = (y - o)$.

The learning rate parameter η_{opt} is obtained by performing a line search so that $\eta_{opt} = \text{argmin}_\eta |y - (o + \Delta o)|$. Letting the first-order approximation of $\Delta o = -e$, we get

$$\eta_{opt} = \frac{1}{\sum_{j \in \Omega} u_j^2 + m^2 \sum_{i \notin \Omega} r_i^2 (\lambda + \|\mathbf{x}\|^2 - (\mathbf{w}_i' \mathbf{x})^2)}. \tag{2.30}$$

In order to stabilize the learning behavior at a near-optimal point, we modify the learning rate as

$$\eta_{opt} = \frac{1}{1 + \sum_{j\in\Omega} u_j^2 + m^2 \sum_{i\notin\Omega} r_i^2(\lambda + \|\mathbf{x}\|^2 - (\mathbf{w}_i'\mathbf{x})^2)}. \tag{2.31}$$

One point that may be noted here is that the learning rate depends on the error $e = (y - o)$. Since the output is not necessarily bounded in $[0, 1]$ for the function approximation task, the step size can be large depending on the value of the predictor variable. However, we always restrict the input in $[-1, 1]$ so that the value of θ is also restricted in that space. Therefore, if the step size becomes large for greedy approximation of the steepest descent, then θ can become very large, and the network may not converge. In order to address this problem, we always confine the predictor variable in $[-1, 1]$, and after the network is trained, we rescale the output accordingly.

3 Experimental Results

We experimented with both synthetic and real-life data sets for classification and function approximation. In this section and later in the letter, we demonstrate the performance of ExOADT and also compare it with other classification and function approximation paradigms.

3.1 Protocol. We trained ExOADT in the online mode. The entire batch of samples is repeatedly presented in the online mode to ExOADT, and we refer to the number of times the batch of samples presented to an OADT as the number of epochs. If the size of the data set is large (i.e., the data density is high), then it is observed that ExOADT takes fewer epochs to converge, and for relatively smaller data sets, ExOADT takes a larger number of epochs. On average, we found that OADT converges near its local optimal solution within 50 epochs, although the required number of epochs increases with the depth of the tree. Subsequently, we report all results for 200 epochs.

The performance of ExOADT depends on the depth (l) of the tree and the parameter m. We have chosen the parameters ϵ and δ as 0.1 and 1, respectively, that is, $m_1 = \log(10l)$. Since m_1 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 choose the steepness parameter of the output node transfer function $m_2 = 1$ for all experiments in pattern classification. We initialize each $\mathbf{w}_i$ randomly with the constraint that $\|\mathbf{w}_i\| = 1$. We always initialize each $\theta_i = 0$. For the regression layer, we initialize each β_{ik} as $\beta_{ik} = 1/N$ for every i and k where $N = |\Omega|$ is the number of leaf nodes.

We normalize all input patterns such that any component of **x** lies in the range $[-1, +1]$. We normalize each component of **x** (say, x_i) as

$$\hat{x}_i = \frac{2x_i - (x_i^{\max} + x_i^{\min})}{x_i^{\max} - x_i^{\min}} \tag{3.1}$$

where $x_i^{\max}$ and $x_i^{\min}$ are the maximum and minimum values of x_i over all observations, respectively. Note that in this normalization, we do not take the outliers into account. Data outliers can badly influence the variables in such normalization. Instead of linear scaling, using certain nonlinear scaling or certain more robust scaling such as that based on interquartile range may improve the performance of our model. However, we stick to the simple linear scaling mainly for two reasons. First, data outliers can be handled separately. We did linear scaling to fit the data in our model, not for data cleansing. We therefore preserve the exact structure of data distribution for each variable. Second, we compared our model with several existing classifiers. In order to investigate the performance of other classifiers, we used certain standard software packages as described in the next section. Since these packages use linear scaling for normalization, we used them for a fair comparison.

3.2 Results of Classification. We first demonstrate the effectiveness of ExOADT in the classification task. We have considered the data sets available in the UCI machine learning repository (Blake & Merz, 1998). Table 1 summarizes the data set description. We considered the data sets from the UCI machine learning repository (Blake & Merz, 1998) mostly based on the fact that the data should not contain any missing variable and consist of only numeric attributes. We modified the Ecoli data originally used in Nakai and Kanehisa (1991) and later in Horton and Nakai (1996) for predicting protein localization sites. The original data set consists of eight different classes out of which three classes—outer membrane lipoprotein, inner membrane lipoprotein, and inner membrane cleavable signal sequence—have only five, two, and two instances, respectively. Since we report the 10-fold cross-validation score, we omitted samples from these three classes and report the results for the rest of the five different classes. Note that in the original work (Nakai & Kanehisa, 1991), the results are not cross-validated either.

We performed 10-fold cross validation on each data set. Table 2 summarizes the performance of ExOADT on these data sets. We also provide a comparison of the results with the hierarchical mixture of experts (HME) (Jordan & Jacobs, 1993, 1994), C4.5 (Quinlan, 1993, 1996), SVM (Vapnik, 1998; Smola & Schölkopf, 1998a; Gunn, 1998), k-nearest neighbor algorithm (Duda et al., 2001), naive Bayes classifier, bagging (Breiman, 1996), and AdaBoost (Schapire & Singer, 1999) algorithm. We implemented the hierarchical mixture of experts by using the MATLAB toolbox as provided in

Table 1: Description of the Pattern Classification Data Sets Obtained from the UCI Machine Learning Repository (Blake & Merz, 1998).

Data Set	Number of Instances	Number of Features	Number of Classes
Indian Diabetes (Pima)	768	8	2
Diagnostic Breast Cancer (Wdbc)	569	30	2
Prognostic Breast Cancer (Wpbc)	198	32	2
Liver Disease (Bupa)	345	6	2
Flower (Iris)	150	4	3
Bacteria (Ecoli)	326	7	5
AI (Monks1)	556	6	2
AI (Monks2)	601	6	2
AI (Monks3)	554	6	2

Murphy (2001, 2003). We used the implementation of WEKA (Garner, 1995; Witten & Frank, 2000) in studying the performance of C4.5, k-nearest neighbor, naive Bayes classifier, bagging, and AdaBoost. To implement SVM, we used the publicly available code in Canu, Grandvalet, and Rakotomamonjy (2003) which is also referred to in Cristianini and Shawe-Taylor (2000). Note that WEKA also provides an implementation of SVM, particularly the sequential minimal optimization (SMO) (Platt, 1998). However, the SMO algorithm as provided in WEKA cannot handle more than two class labels. We therefore implemented the multiclass SVM using the software provided in Canu et al. (2003).

We also graphically depict the classification results in Figure 2. Figure 2 illustrates the performances of the best-performing ExOADT and the best-performing HME. Note that depicting the results of the best-performing model structure is a bit overoptimistic. However, in model selection, usually the best-performing model is selected based on the cross-validation performance. In the domain of neural networks also, the results of the best-performing neural network structure are usually reported. In this context, we must mention that we implemented multilayer perceptron (MLP) with backpropagation learning using the WEKA tool. We studied the performance of MLP for various structures and observed that for certain data sets, MLP provides very poor performance. For example, with the liver disease data set (Bupa), 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 one hidden layer and 15 hidden nodes. Possibly an increase in the number of hidden layers may increase the score.

Table 2: Ten-Fold Cross-Validation Performance of ExOADT on the Classification Data Sets as in Table 1 for Different Depths from Depth = 3 to Depth = 7.

				SVM (Gaussian Kernel)		
Data Set	C4.5	k-NN	NaiveBayes	(size = 1)	(size = 5)	—
Pima	74.09 Bagging (C4.5): 74.87	70.44 Bagging (k-NN): 73.44	75.78 Bagging (Naive Bayes): 75.91	66.93 AdaBoost (C4.5): 71.35	73.30 AdaBoost (k-NN): 70.44	 AdaBoost (Naive Bayes): 76.56
Hierarchical mixture of experts	Height = 3: 71.09	Height = 4: 69.92	Height = 5: 70.03	Height = 6:68.60	Height = 7: 68.75	Height = 8: 68.98
ExOADT	Depth = 3: 73.85	Depth = 4: 77.49	Depth = 5: 76.97	Depth = 6: 75.80	Depth = 7: 77.10	—
Wdbc	92.44 Bagging (C4.5): 94.73	96.13 Bagging (k-NN): 96.49	93.15 Bagging (Naive Bayes): 93.15	96.48 AdaBoost (C4.5): 95.96	95.80 AdaBoost (k-NN): 96.13	 AdaBoost (Naive Bayes): 95.96
Hierarchical mixture of experts	Height = 3: 92.80	Height = 4: 91.05	Height = 5: 93.86	Height = 6: 93.50	Height = 7: 93.50	Height = 8: 92.97
ExOADT	Depth = 3: 96.65	Depth = 4: 97.18	Depth = 5: 97.18	Depth = 6: 97.53	Depth = 7: 97	—
Wpbc	76.77 Bagging (C4.5): 80.81	72.22 Bagging (k-NN): 71.72	66.67 Bagging (Naive Bayes): 66.67	76.70 AdaBoost (C4.5): 75.76	72.79 AdaBoost (k-NN): 72.22	 AdaBoost (Naive Bayes): 73.74
Hierarchical mixture of experts	Height = 3: 68.67	Height = 4: 74.78	Height = 5: 68.22	Height = 6: 70.17	Height = 7: 72.61	Height = 8: 66.72
ExOADT	Depth = 3: 77.82	Depth = 4: 78.26	Depth = 5: 79.26	Depth = 6: 80.76	Depth = 7: 81.82	—

Table 2: Continued

Data Set	C4.5	k-NN	NaiveBayes	SVM (Gaussian Kernel) (size = 1)	SVM (Gaussian Kernel) (size = 5)	—
Bupa	66.38 Bagging (C4.5): 72.46	64.35 Bagging (k-NN): 61.45:	55.94 Bagging (Naive Bayes): 55.36	60.62 AdaBoost (C4.5): 67.54	67.84 AdaBoost (k-NN): 64.35	AdaBoost (Naive Bayes): 65.51
Hierarchical mixture of experts	Height = 3: 65.29	Height = 4: 66.19	Height = 5: 62.24	Height = 6: 62.19	Height = 7: 63.48	Height = 8: 63.90
ExOADT	Depth = 3: 64.62	Depth = 4: 66.76	Depth = 5: 63.52	Depth = 6: 62.31	Depth = 7: 62.36	—
Iris	95.33 Bagging (C4.5): 94.67	96 Bagging (k-NN): 96	96 Bagging (Naive Bayes): 96	91.33 AdaBoost (C4.5): 94	90.67 AdaBoost (k-NN): 96	AdaBoost (Naive Bayes): 94.67
Hierarchical mixture of experts	Height = 3: 83.33	Height = 4: 81.33	Height = 5: 86.67	Height = 6: 84	Height = 7: 84.67	Height = 8: 87.33
ExOADT	Depth = 3: 95.33	Depth = 4: 96	Depth = 5: 96	Depth = 6: 96.67	Depth = 7: 96.67	—
Ecoli	84.66 Bagging (C4.5): 83.44	82.82 Bagging (k-NN): 83.13	88.04 Bagging (Naive Bayes): 87.42	74.18 AdaBoost (C4.5): 85.58	80.95 AdaBoost (k-NN): 82.82	AdaBoost (Naive Bayes): 88.04
Hierarchical mixture of experts	Height = 3	Height = 4	Height = 5	Height = 6	Height = 7	Height = 8
	—	—	—	—	—	—
ExOADT	Depth = 3: 78.51	Depth = 4: 81.29	Depth = 5: 76.01	Depth = 6: 76.61	Depth = 7: 79.97	—

Table 2: Continued

Monks1	89.93 Bagging (C4.5): 100	88.49 Bagging (*k*-NN): 87.23	66.55 Bagging (Naive Bayes): 66.19	86.97 AdaBoost (C4.5): 91.37	98.52 AdaBoost (*k*-NN): 88.49	 AdaBoost (Naive Bayes): 65.83
Hierarchical mixture of experts	Height = 3: 96.96	Height = 4: 95.54	Height = 5: 98.39	Height = 6: 98.20	Height = 7: 98.39	Height = 8: 98
ExOADT	Depth = 3: 90.11	Depth = 4: 96.76	Depth = 5: 94.77	Depth = 6: 98.36	Depth = 7: 98.03	—
Monks2	92.85 Bagging (C4.5): 98.67	89.52 Bagging (*k*-NN): 89.52	65.39 Bagging (Naive Bayes): 65.06	94.86 AdaBoost (C4.5): 99.67	95.67 AdaBoost (*k*-NN): 89.52	 AdaBoost (Naive Bayes): 61.40
Hierarchical mixture of experts	Height = 3: 76.20	Height = 4: 76.36	Height = 5: 77.05	Height = 6: 80.20	Height = 7: 81.02	Height = 8: 80.54
ExOADT	Depth = 3: 72.26	Depth = 4: 78.05	Depth = 5: 80.07	Depth = 6: 79.10	Depth = 7: 82.72	—
Monks3	98.92 Bagging (C4.5): 98.92	78.70 Bagging (*k*-NN): 83.75	91.52 Bagging (Naive Bayes): 91.52	95.66 AdaBoost (C4.5): 97.83	93.68 AdaBoost (*k*-NN): 78.70	 AdaBoost (Naive Bayes): 94.95
Hierarchical mixture of experts	Height = 3: 91.52	Height = 4: 92.62	Height = 5: 92.40	Height = 6: 91.70	Height = 7: 92.07	Height = 8: 92.01
ExOADT	Depth = 3: 95.83	Depth = 4: 96.19	Depth = 5: 96.73	Depth = 6: 97.82	Depth = 7: 96.57	—

Notes: We compare the performance with other classifiers C4.5, *k*-NN, naiveBayes, SVM, bagging, boosting, and hierarchical mixture of experts. C4.5 (J4.8), *k*-NN, naiveBayes, bagging, and boosting implementations are available in the WEKA software package (Garner, 1995; Witten & Frank, 2000). In WEKA, the value of *k* in the case of the *k*-NN classifier is selected by the leave-one-out rule. We implemented SVM with gaussian kernels using the MATLAB code available in Canu et al. (2003) and Cristianini and Shawe-Taylor (2000), and the HME has been implemented using the toolbox (Murphy, 2001, 2003).

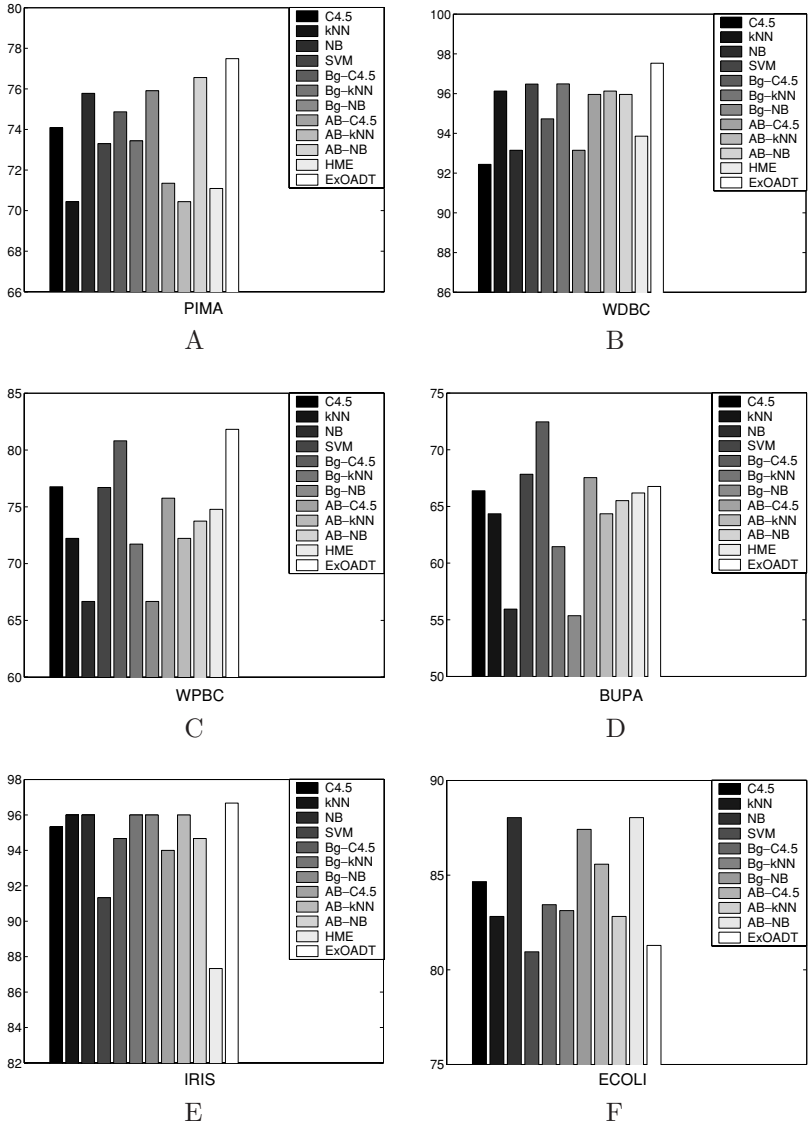

Figure 2: Pictorial illustration of the performance of different classifiers. NB = naive Bayes classifier. Bg = bagging. AB = Adaboost ensemble classifiers. For HME and ExOADT, the best results are depicted.

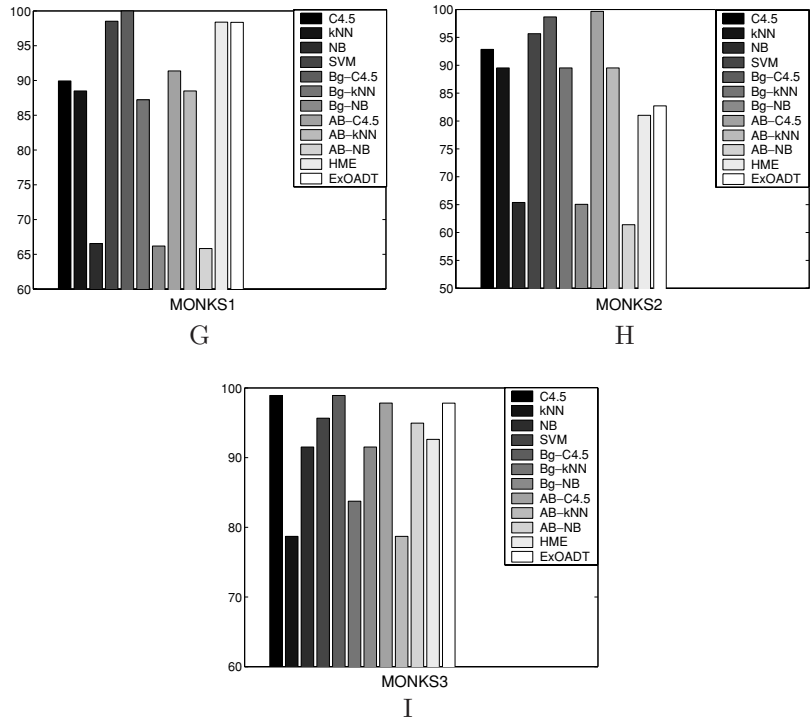

Figure 2: Continued.

In general, HME performs better than MLP in most of the cases in terms of classification score, which is also reported in Jordan and Jacobs (1993). We therefore do not report the classification score obtained with MLP separately.

We observe that for the Pima Indians Diabetes data set, ExOADT (of depth = 7) outperforms all classifiers, including bagging and AdaBoost. It is interesting to observe a similar set of results in the case of breast cancer data (both the diagnostic and the prognostic breast cancers). However, we observe that in the case of liver disease, ExOADT performs poorly as compared to bagging. However, its performance is still better than the hierarchical mixture of experts. Again in the case of Iris data set, we observe that ExOADT performs better than its counterparts, including the bagging and AdaBoost. However, in the case of Ecoli data, ExOADT cannot perform as well as the others. One of the most interesting aspects is that for this task of protein site localization, the naive Bayes performs the best—even better than the bagged and boosted trees. Here we did not report the results of hierarchical mixture of experts since it could not properly converge

using the EM algorithm. In the artificial domain of the Monks problem, we find that bagged and boosted trees perform very well—better than the others. However, we observe that the ExOADT performs better than other classifiers including SVM if we leave out the bagging and boosting.

Overall, we observe that the ExOADT has the potential to be established as a best classifier if we compare it with single decision trees, hierarchical mixture of experts, and even SVM. However, we see that the multiclassifier systems like bagging and boosting have the capacity to outperform ExOADT in many situations. It may be noted here that the classification and function approximation performance of a model also depends on the data set, as pointed out by Friedman (2001). Evidently without any theoretical establishment, it is difficult to claim that ExOADT is the best single classifier (bagging and boosting combine the decision of more than one classifier); however, we claim that on the data sets that we used, ExOADT performs best.

3.3 Results on Function Approximation. We now demonstrate the effectiveness of ExOADT in function approximation tasks. As mentioned before, we normalize the input data in the range $[-1, 1]$. Since the learning rate parameter is always multiplied by the output error $e = (y - o)$, the learning step size can be large for large predictor output. This in turn results in a larger step size for the changes in θ of the nonterminal nodes, and the network search space can shoot out of the normalized input space—the network may not converge as a consequence. We therefore normalize the predictor variable also in the range $[-1, 1]$, and the output produced by ExOADT is rescaled accordingly to compare with the original predictor variable values.

First, we demonstrate the behavior of ExOADT in approximating synthetic functions. We considered a one-dimensional function as shown in Figure 3. The characteristic of the function is such that it is often difficult to approximate it using the standard multilayer perceptron due to the presence of two very different-sized humps. An MLP network often takes a very high learning time to fine-tune the smaller hump. Figure 3A illustrates the results obtained by ExOADT with different depths from depth $= 3$ to depth $= 6$ under the noiseless condition. Note that in the one-dimensional case every nonterminal node has $w = 1$ and according to the learning rules (see equation 2.17), w is not updated; only the value of θ is updated. We observe that initially, the complexity of the predicted function increases with depth; however, after a certain depth, the complexity of the predicted function does not increase, and the network is able to produce a smooth function. This is in contrast to the conventional multilayer neural networks, where, without any imposed regularization (Haykin, 1999), the network often overfits with the increase in the number of hidden nodes.

We then added noise to the predictor variable; making a random shift in a range $[-\Delta, \Delta]$. Figures 3B, 3C, and 3D illustrate the performance of

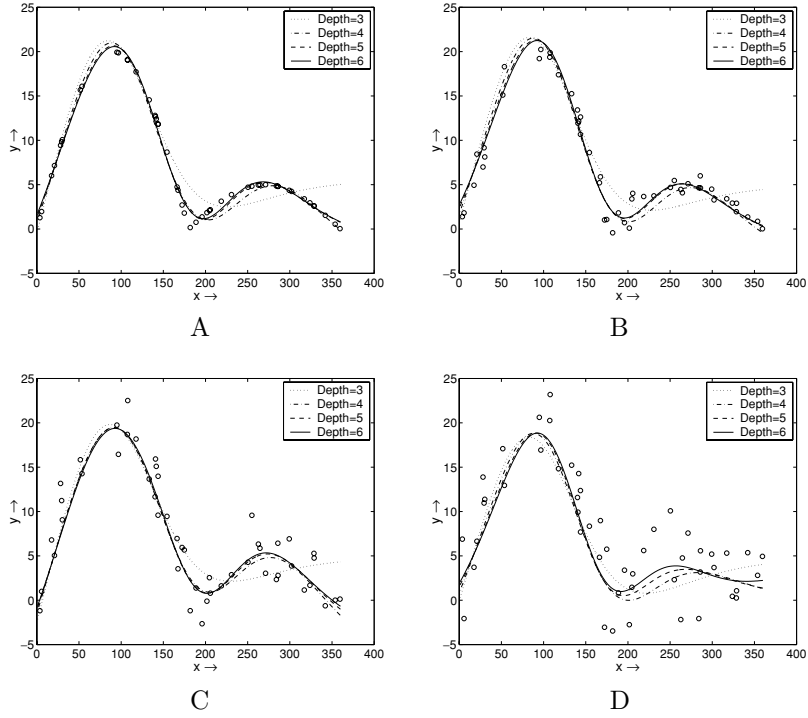

Figure 3: Approximation of a one-dimensional function using ExOADT for different depths from depth = 3 to depth = 6. (A) A noiseless situation. (B–D) The approximated curve obtained noise levels 1, 2, and 3, respectively.

the ExOADT for different noise levels with $\Delta = 1$, 2, and 3, respectively. We observe that even with a high degree of noise, the ExOADT is able to find a smooth function without doing the overfitting, although the depth is increased to $l = 6$. There is an embedded regularization in ExOADT since the summation of the activations of the leaf nodes is always unity.

We demonstrate the behavior of ExOADT in approximating two-dimensional functions. We generated four gaussian humps on the four corners of a two-dimensional plane, as shown in Figure 4. We illustrate the function approximated by ExOADT for different depths and different numbers of training samples. We observe a similar behavior as in the case of one-dimensional function that as the depth increases initially, the complexity of the estimated function increases, and it fine-tunes the training data. However, after a certain depth, the function complexity does not increase, and as a result it does not overfit the data (see Figures 4A, 4B, and 4C). We then experimented by introducing noise in the predictor variable. We added

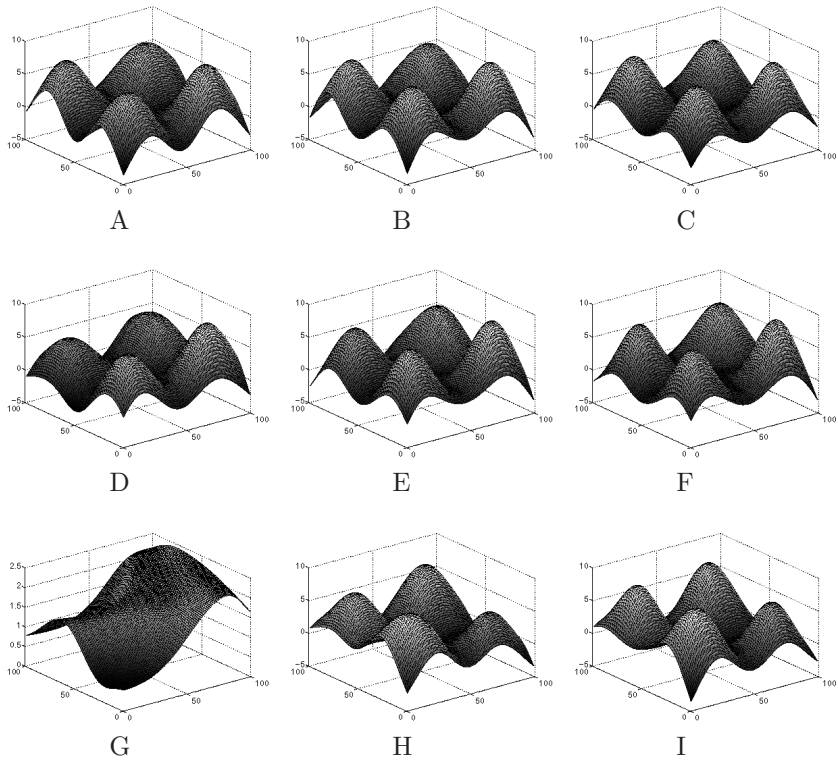

Figure 4: Approximation of a two-dimensional function for different depths of ExOADT. (A–C) The function interpolated from only 300 data points by ExOADT of depth = 5, 6, and 7, respectively. (D–F) The same function interpolated from 300 noisy data points with noise level unity for depth = 5, 6, and 7, respectively. (G–I) The functions interpolated for 1000 noisy data points with noise level = 3 for depth = 5, 6, and 7, respectively.

noise in the same way as in the case of the one-dimensional function: we randomly shifted the predictor value in the range $[-\Delta, \Delta]$. Figures 4D, 4E, and 4F illustrate the results on noisy data obtained by ExOADT of depths from 5 to 7 for 300 sample points with $\Delta = 1$. We then experimented with $\Delta = 3$ and observed that ExOADT fails to interpolate the function within 200 epochs. We then considered 1000 randomly generated nosiy points with noise level $\Delta = 3$ and approximated the functions with depths from 5 to 7. Figures 4G, 4H, and 4I illustrate the respective results. We observe that in the highly noisy situation, an ExOADT of depth = 5 fails to interpolate the function properly, although it is able to do so with depths 6 and 7.

Table 3: Description of the Function Approximation Data Sets Obtained from the Bilkent University Data Repository (Guvenir & Uysal, 2000).

Data Set	Number of Instances	Number of Variables	Maximum Predicted Value	Minimum Predicted Value
Baseball (BB)	337	16	6100	109
Housing (HH)	506	13	50	5
Kinematics (KI)	8192	8	1.458521	0.040165
Elevator (EV)	8752	18	0.078	0.012
Ailerons (AL)	7154	40	−0.0003	−0.0035
Friedman (FR)	40,768	10	30.522	−1.228

3.4 Function Approximation Results on Real-Life Data. We now demonstrate the effectiveness of ExOADT in approximating functions in real-life data sets. We collected the data sets for function approximation from the publicly available web site of the Bilkent University data repository (Guvenir & Uysal, 2000). We have chosen six data sets mostly considering the fact that the data should not contain any categorical variable and there should be no missing values. Table 3 summarizes the data sets in terms of the number of variables and the number of samples. Let us provide a brief description as in Guvenir and Uysal (2000). The baseball data set contains the performance of the players in terms of batting average, number of runs, number of hits, number of doubles, and others and their salary in terms of thousands of dollars. The data were collected in 1992 (the original source description is provided in Guvenir & Uysal, 2000). The task is to find a correspondence between the performance of the players and their salaries. The Ailerons data set addresses a control problem: flying an F16 aircraft. The attributes describe the status of the aircraft, and the goal is to predict the control action on the ailerons of the aircraft. The Elevator data set is similar to the Ailerons, and obtained from the task of controlling an F16 aircraft. However, attributes here are different from the Ailerons domain, and the goal (predictor) variable is related to an action taken on the elevators of the aircraft (Guvenir & Uysal, 2000). The Boston Housing data originally appeared in Harrison and Rubinfeld (1978), and are available in the StatLib library maintained by Carnegie Mellon University and UCI (Blake & Merz, 1998, Guvenir & Uysal, 2000). The task is to predict housing prices in the Boston area depending on such factors as pollution, accessibility, crime rate, and structure. The Kinematics data set is concerned with the forward kinematics of an eight-link robot arm. The Friedman data set is an artificial data set originally used by Friedman (1991) in approximating functions with MARS (multivariate adaptive regression splines).

We divided the data randomly in different proportions. First, we considered 25% data for training and the rest, 75%, for testing. We then considered

50% for both training and testing, and finally considered 75% for training and 25% for testing. We report the results in terms of the absolute difference of the actual predictor value and the output of the network for both the training and test sets:

$$\text{Absolute error} = \mathcal{E}_{\mathbf{x}}|y(\mathbf{x}) - o(\mathbf{x})|, \tag{3.2}$$

where $\mathcal{E}$ stands for the expected value. We also report the normalized error measure as provided by Friedman (2001), which is given as

$$\text{Normalized error} = \frac{\mathcal{E}_{\mathbf{x}}|y(\mathbf{x}) - o(\mathbf{x})|}{\mathcal{E}_{\mathbf{x}}|y(\mathbf{x}) - median_{\mathbf{x}}\, y(\mathbf{x})|}. \tag{3.3}$$

The normalized error is the average absolute error normlized by the optimum constant error (Friedman, 2001).

We report the average absolute error and the normalized error for both the training data set and the test data set for different sizes of the training data in Table 4. We compare our results with the performance of support vector regression (Vapnik et al., 1997; Smola & Schölkopf, 1998b; Gunn, 1998). We approximated the functions in SVR using the ϵ-insensitive loss function (Vapnik et al., 1997) with gaussian kernels. We implemented the algorithm in Matlab using the software available in Canu et al. (2003), which is also referred to in Cristianini and Shawe-Taylor (2000). We also pictorially depict the performance of both ExOADT and SVR in Figure 5.

We observed that the performance of SVR is dependent on the choice of ϵ and the gaussian kernel size. For the normalized data set, we have chosen $\epsilon = 0.01$ and report the results for different sizes of the gaussian kernels. The results show that SVR is much better than ExOADT in terms of the training error. However, if we compare the test errors, we observe that they are more or less comparable, although SVR has an edge over ExOADT. We report here the best results that are obtained with SVR by fine-tuning the parameters. We report the results for ExOADT with $\epsilon = 0.1$ and $\delta = 1$ in all the cases. We did not investigate whether fine-tuning the parameters of ExOADT would improve the results. We did not report the results of SVR for the Kinematics data set with 75% training data and the other larger data sets since we could not get results for these data sets using SVR due to the memory limitation. Note that due to the memory requirements, SVR is bottlenecked in terms of scalability, which is not true for ExOADT since it operates in the online mode without needing to store the data points. We demonstrate this by reporting the results of ExOADT for certain larger data sets: Elevator, Ailerons, and the Friedman (as used in MARS) data sets.

Table 4: Performance of ExOADT on the Function Approximation Data Sets as in Table 3, for Different Depths from 3 to 7.

Data Set	Train Size	Error Type	Error						
				ExOADT					
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7	—
BB	25%	Absolute	Train	158.07	131.64	119.42	121.02	119.65	
			Test	661.98	595.15	601.37	584.85	553.59	
		Friedman	Train	0.1656	0.1379	0.1251	0.1268	0.1254	
			Test	0.6936	0.6236	0.6301	0.6128	0.5800	
				SVR (Gaussian kernel)					
				size = 0.5	size = 1	size = 2	size = 3	size = 4	size = 5
		Absolute	Train	56.7471	57.0302	68.4514	119.8526	172.1374	209.5573
			Test	684.1490	576.9421	593.5051	546.8188	519.8181	504.6622
		Friedman	Train	0.05946	0.05976	0.07172	0.1256	0.1804	0.2196
			Test	0.7168	0.6045	0.6219	0.5730	0.5447	0.5288
				ExOADT					
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7	—
	50%	Absolute	Train	328.80	219.18	191.07	218.13	211.01	
			Test	517.48	680.67	580.77	678.56	729.81	
		Friedman	Train	0.3445	0.2297	0.2002	0.2286	0.2211	
			Test	0.5422	0.7132	0.6085	0.7110	0.7647	
				SVR (Gaussian kernel)					
				size = 0.5	size = 1	size = 2	size = 3	size = 4	size = 5
		Absolute	Train	56.5042	55.5422	129.6090	212.3789	262.3273	298.2583
			Test	647.7100	572.5080	601.8826	550.6553	514.1745	498.5944
		Friedman	Train	0.0592	0.0582	0.1358	0.2225	0.2749	0.3125
			Test	0.6787	0.5999	0.6364	0.5770	0.5387	0.5224

Table 4: Continued

Data Set	Train Size	Error Type	Error						
						ExOADT			
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7	—
	75%	Absolute	Train	308.77	276.57	253.88	259.19	291.37	
			Test	503.87	555.76	513.63	517.54	577.67	
		Friedman	Train	0.3235	0.2898	0.2660	0.2716	0.3053	
			Test	0.5280	0.5823	0.5382	0.5423	0.6053	
						SVR (gaussian kernel)			
				size = 0.5	size = 1	size = 2	size = 3	size = 4	size = 5
		Absolute	Train	57.3366	59.1674	151.4899	229.6618	289.5370	329.1706
			Test	545.4307	567.1411	532.5809	516.1467	475.0909	448.6760
		Friedman	Train	0.0601	0.0620	0.1587	0.2406	0.3034	0.3449
			Test	0.5715	0.5942	0.5580	0.5408	0.4978	0.4701
						ExOADT			
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7	—
HH	25%	Absolute	Train	1.747	1.537	1.321	1.759	1.319	
			Test	2.706	2.580	2.694	2.916	3.034	
		Friedman	Train	0.2676	0.2353	0.2022	0.2693	0.2020	
			Test	0.4144	0.3951	0.4125	0.4466	0.4646	
						SVR (gaussian kernel)			
				size = 0.5	size = 1	size = 2	size = 3	size = 4	size = 5
		Absolute	Train	0.4231	0.4713	0.9389	1.3615	1.7887	2.0910
			Test	3.5631	3.0910	2.5980	2.5112	2.4927	2.7686
		Friedman	Train	0.0648	0.0722	0.1438	0.2085	0.2739	0.3202
			Test	0.5456	0.4733	0.3978	0.3845	0.3817	0.4239

Table 4: Continued

				ExOADT					
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7	—
HH	50%	Absolute	Train	2.094	1.909	1.716	1.524	1.599	
			Test	2.623	2.521	2.261	2.536	2.257	
		Friedman	Train	0.3206	0.2924	0.2628	0.2334	0.2450	
			Test	0.4017	0.3861	0.3462	0.3883	0.3456	
				SVR (gaussian kernel)					
				size = 0.5	size = 1	size = 2	size = 3	size = 4	size = 5
		Absolute	Train	0.4244	0.6415	1.2046	1.6172	1.8200	2.0522
			Test	2.8217	2.4236	2.1322	2.2677	2.2789	2.4503
		Friedman	Train	0.0649	0.0982	0.1845	0.2476	0.2787	0.3142
			Test	0.4321	0.3711	0.3265	0.3472	0.3489	0.3752
				ExOADT					
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7	—
	75%	Absolute	Train	1.788	1.698	1.570	1.476	1.491	
			Test	2.281	2.049	2.162	2.422	2.367	
		Friedman	Train	0.2737	0.2601	0.2405	0.2260	0.2283	
			Test	0.3494	0.3137	0.3310	0.3708	0.3624	
				SVR (gaussian kernel)					
				size = 0.5	size = 1	size = 2	size = 3	size = 4	size = 5
		Absolute	Train	0.4179	0.6348	1.3013	1.6397	1.8293	1.9801
			Test	2.7586	2.2838	2.0105	2.1420	2.2519	2.3860
		Friedman	Train	0.0640	0.0972	0.1993	0.2511	0.2801	0.3032
			Test	0.4224	0.3497	0.3078	0.3280	0.3448	0.3653

Table 4: Continued

Data Set	Train Size	Error Type	Error						
						ExOADT			
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7	—
KI	25%	Absolute	Train	0.0978	0.0740	0.0592	0.0474	0.0433	
			Test	0.1007	0.0782	0.0684	0.0677	0.0690	
		Friedman	Train	0.4539	0.3435	0.2747	0.2198	0.2010	
			Test	0.4675	0.3630	0.3174	0.3141	0.3203	
						SVR (gaussian kernel)			
				size = 0.5	size = 1	size = 2	size = 3	size = 4	size = 5
		Absolute	Train	0.0134	0.0143	0.0537	0.0843	0.1017	0.1136
			Test	0.0914	0.0769	0.0750	0.0982	0.1129	0.1229
		Friedman	Train	0.0620	0.0664	0.2491	0.3910	0.4721	0.5271
			Test	0.4243	0.3571	0.3482	0.4558	0.5238	0.5703
						ExOADT			
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7	—
	50%	Absolute	Train	0.1008	0.0776	0.0604	0.0536	0.0488	
			Test	0.1052	0.0836	0.0658	0.0619	0.0614	
		Friedman	Train	0.4679	0.3602	0.2803	0.2491	0.2268	
			Test	0.4883	0.3883	0.3055	0.2875	0.2849	
						SVR (gaussian kernel)			
				size = 0.5	size = 1	size = 2	size = 3	size = 4	size = 5
		Absolute	Train	—	—	—	—	—	—
			Test	—	—	—	—	—	—
		Friedman	Train	—	—	—	—	—	—
			Test	—	—	—	—	—	—

Table 4: Continued

				ExOADT				
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7
	75%	Absolute	Train	0.0927	0.0752	0.0631	0.0535	0.0528
			Test	0.0922	0.0763	0.0649	0.0587	0.0590
		Friedman	Train	0.4304	0.3488	0.2927	0.2485	0.2451
			Test	0.4279	0.3540	0.3014	0.2723	0.2739
EV	25%	Absolute	Train	0.0017	0.0017	0.0016	0.0015	0.0015
			Test	0.0017	0.0018	0.0017	0.0017	0.0016
		Friedman	Train	0.4135	0.4067	0.3799	0.3632	0.3557
			Test	0.4266	0.4278	0.4072	0.4029	0.4004
	50%	Absolute	Train	0.0016	0.0015	0.0014	0.0014	0.0014
			Test	0.0017	0.0016	0.0015	0.0015	0.0015
		Friedman	Train	0.4009	0.3632	0.3477	0.3431	0.3409
			Test	0.4174	0.3774	0.3687	0.3666	0.3709
	75%	Absolute	Train	0.0019	0.0018	0.0017	0.0017	0.0015
			Test	0.0020	0.0018	0.0017	0.0017	0.0016
		Friedman	Train	0.4788	0.4375	0.4087	0.4062	0.3726
			Test	0.4871	0.4433	0.4217	0.4183	0.3888
AL	25%	Absolute	Train	0.000117	0.000116	0.000118	0.000115	0.000115
			Test	0.000123	0.000129	0.000128	0.000130	0.000131
		Friedman	Train	0.3861	0.3828	0.3894	0.3795	0.3795
			Test	0.4059	0.4257	0.4224	0.4290	0.4323

Table 4: Continued

Data Set	Train Size	Error Type	Error	ExOADT				
				Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7
	50%	Absolute	Train	0.00121	0.000117	0.000116	0.000115	0.000116
			Test	0.000125	0.000124	0.000126	0.000124	0.000125
		Friedman	Train	0.3993	0.3861	0.3828	0.3795	0.3828
			Test	0.4125	0.4092	0.4158	0.4092	0.4125
	75%	Absolute	Train	0.00124	0.000123	0.000117	0.000120	0.000112
			Test	0.000128	0.000127	0.000124	0.000130	0.000123
		Friedman	Train	0.4092	0.4059	0.3861	0.3960	0.3696
			Test	0.4224	0.4191	0.4092	0.4290	0.4059
FR	25%	Absolute	Train	1.1516	0.9132	0.8544	0.8022	0.7826
			Test	1.1546	0.9267	0.8862	0.8653	0.8718
		Friedman	Train	0.2833	0.2247	0.2102	0.1974	0.1925
			Test	0.2841	0.2280	0.2180	0.2129	0.2145
	50%	Absolute	Train	1.0698	0.8817	0.8428	0.8105	0.8119
			Test	1.0737	0.8938	0.8586	0.8419	0.8542
		Friedman	Train	0.2632	0.2169	0.2074	0.1994	0.1997
			Test	0.2641	0.2199	0.2112	0.2071	0.2102
	75%	Absolute	Train	1.0510	0.9132	0.8653	0.8195	0.7998
			Test	1.0548	0.9131	0.8680	0.8367	0.8308
		Friedman	Train	0.2586	0.2247	0.2129	0.2016	0.1968
			Test	0.2595	0.2246	0.2135	0.2059	0.2043

Notes: The table also demonstrates a comparison with the SVR for certain data sets. We implemented the SVR using the toolbox as available in Canu et al. (2003).

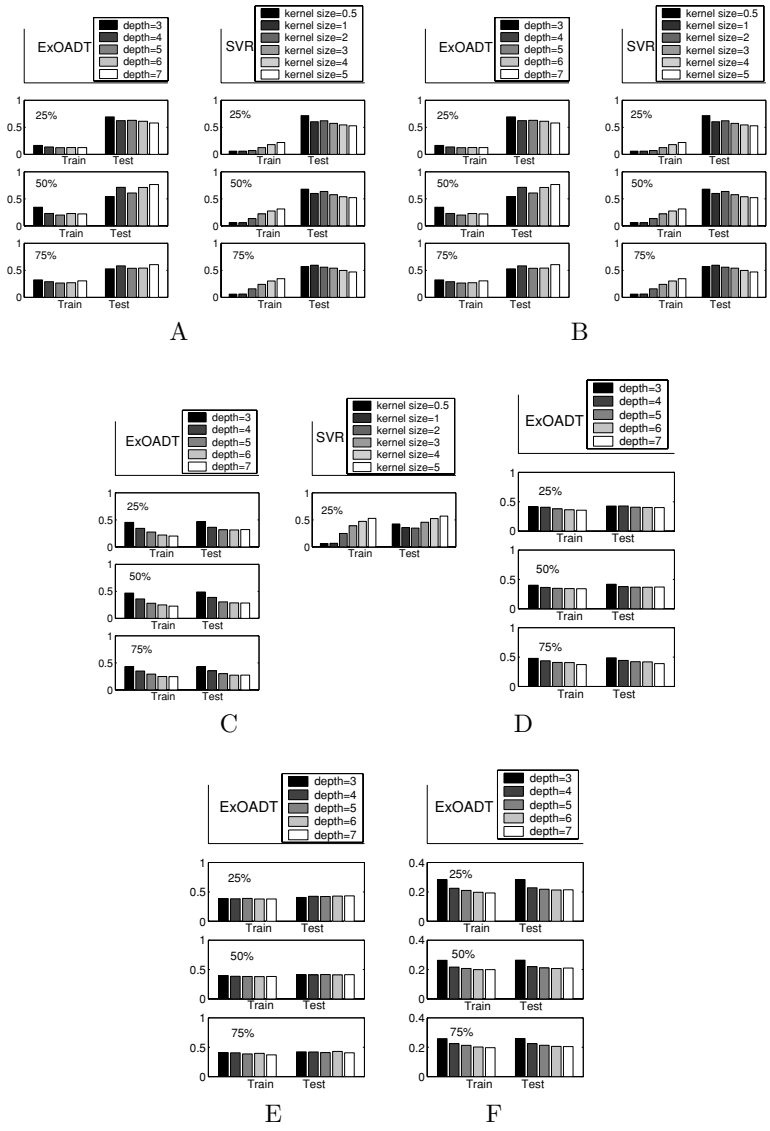

Figure 5: Pictorial illustration of the performance of ExOADT and SVR for function approximation. We provide the results for different depths of ExOADT and different kernel sizes of SVR. (A–F) The results for the Baseball (BB), Housing (HH), Kinematics (KI), Elevator (EV), Ailerons (AI), and Friedman (MARS) data sets, respectively. Each figure illustrates the performance for the training data and test data separately with different training sample sizes of 25%, 50%, and 75% respectively. For data sets Elevator (EV), Ailerons (AI), and Friedman (MARS) (panels (D–F), we have results only for ExOADT; results for SVR could not be obtained due to memory constraints.

4 Discussion

4.1 Relationship with the Decision Trees. The proposed model ExOADT is essentially an extension to the OADT (Basak, 2004) for multiclass (more than two) classification and function approximation tasks. We train ExOADT using greedy gradient descent rule in the online mode. Since we do not freeze the learning, ExOADT can adapt to changing patterns.

One interesting observation is that an ExOADT of depth l can embed any decision tree of depth, $1 \leq \text{depth} \leq l$. Let us consider the different structures of binary decision trees of depth, depth $\leq l$ that can occur for a k-class problem. Let us consider that every nonterminal node is described by a condition of the form $\mathbf{w}'\mathbf{x} + \theta \leq 0$, where if the $\leq$ condition is satisfied, then the pattern is allocated to the left child; otherwise, it is allocated to the right child. Evidently the axis-parallel decision trees are also embedded in this structure. If we restrict the values of $\beta_{ij} \in \{0, 1\}$ for all $i \in \Omega$ and $j \in O$, then we can realize any binary decision tree of a depth, depth $\leq l$ for a k-class problem. For a two-class classification problem, we illustrate a few examples in Figure 6.

We consider only those decision trees that do not represent any null condition in the decision space; each nonterminal node has exactly two children. Since if any nonterminal node has only one child in either the left or right path, then the other path represents a null space, which is an "unknown" decision. Evidently for a k-class problem there are k^N possible decision trees, where $N = 2^l$ is the number of leaf nodes of a tree of depth $= l$. However, this includes the possibilities where all k classes are not represented. If we consider the number of possible decision trees that can represent exactly k classes, then the problem is different. It is equivalent to finding out the number of ways N balls can be placed in k holes so that each hole gets at least one ball. This is equivalent to the problem in the clustering as to finding out the number of possible ways a set of N points can be partitioned into k clusters so that each cluster consists of at least one point. Let $T(k, N)$ represent the number of possible ways. Then

$$T(k, N) = \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k - i)^N, \tag{4.1}$$

which can be proved by induction. Evidently, $T(k, N) << k^N$ for a large value of N and $k > 1$. Here in the case of ExOADT, even if we restrict $\beta_{ij} \in \{0, 1\}$ for a k-class classification problem, we can generate 2^{kN} different structures—most of which are invalid or ambiguous. For example, these representations also include the possibilities of having multiple class labels assigned to a single leaf node. In short, even if we restrict the values of β in $\{0, 1\}$, the valid search space is less than a fraction of $(\frac{k}{2^k})^N$ of the entire number of possibilities. The fraction $(\frac{k}{2^k})^N$ is much smaller than unity for a

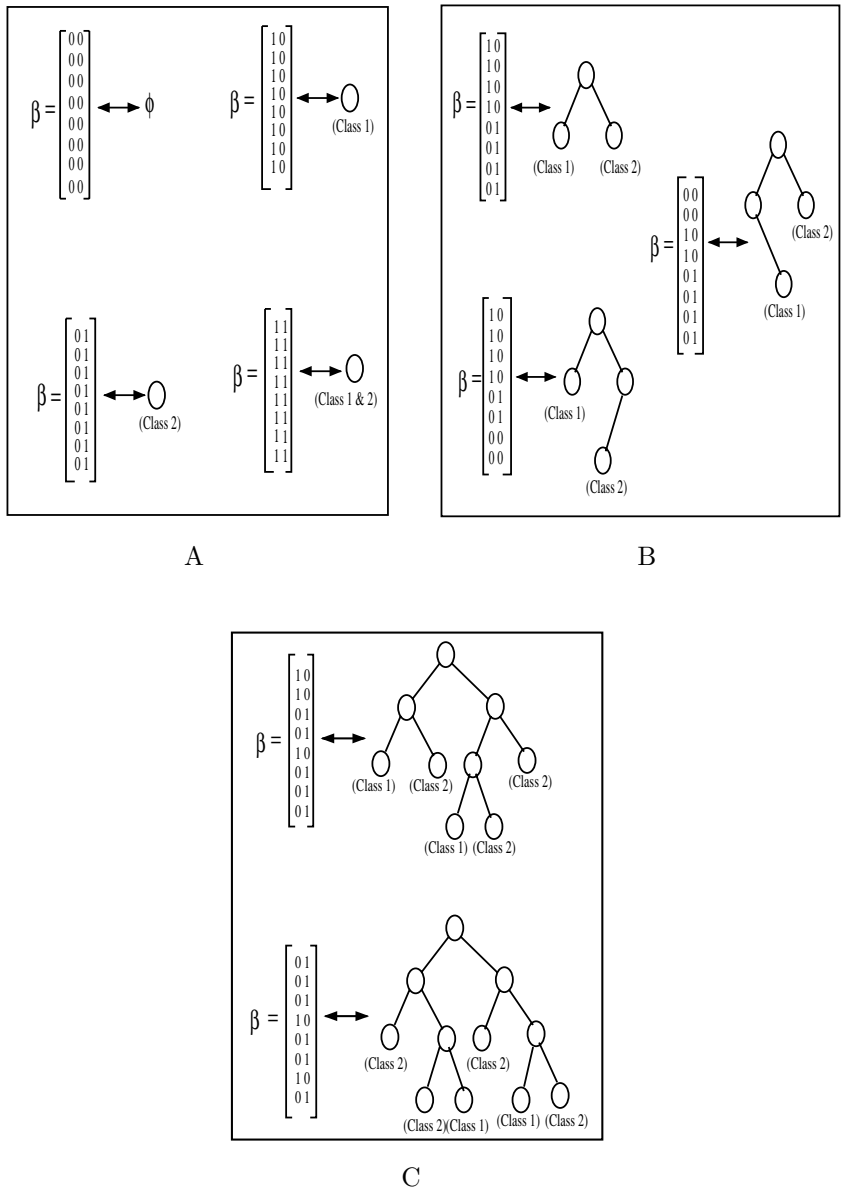

Figure 6: Realization of different decision tree structures for different connection matrix β.

large value of N—the number of leaf nodes. Thus, for a continuous space of β, we have a very narrow search space for obtaining the desired solution, and therefore the steepest gradient descent algorithm works.

Unlike the existing OADT (Basak, 2004), ExOADT provides a flexibility of embedding different decision tree structures in a single framework. In the case of OADT, the structure is fixed for a given depth, and we are allowed to change only the local hyperplanes in each nonterminal node. In the case of ExOADT, we can not only change the local hyperplanes but also, in effect, can smoothly change the structure of the tree.

4.2 Relationship with RBF. In radial basis function networks, normally the joint distribution $P(\mathbf{x}, y)$ is modeled as a sum of kernel densities represented by the hidden nodes (Moody & Darken, 1989). The output is then usually obtained by regression of the predictor variable and the activation of the hidden nodes. In general, gaussian kernels are selected for the hidden nodes. In the case of normalized RBF networks (Moody & Darken, 1989), the function is approximated as (see Girosi et al., 1995)

$$o(\mathbf{x}) = \frac{\sum_{i=1}^{n} c_i G(\mathbf{x} - \mathbf{t_i})}{\sum_{i=1}^{n} G(\mathbf{x} - \mathbf{t_i})}, \tag{4.2}$$

where $G(.)$ are the kernel functions and $\mathbf{t_i}$ are the kernel centers or parameters. In the case of normalized RBF, the space spanned by the individual hidden nodes is no longer elliptical, and the sum of the activations of the hidden nodes is always normalized to unity. Girosi et al. (1995) noted that normalized RBFs have a tight connection with the regularization theory. In the case of ExOADT, the sum of the leaf node activations always goes to unity (or some constant) provided that the root node activation is unity. The leaf nodes in ExOADT in a sense behave similarly to the hidden nodes of the normalized RBFs. However, space spanned by the leaf nodes is not governed by the parametric kernel functions; rather, they are spanned by the decision hyperplanes in the nonterminal nodes from the root to the leaf nodes. The very fact that the summation of the activation of the leaf nodes goes to unity imposes certain inherent regularization in ExOADT similar to the normalized RBFs (Girosi et al., 1995). Instead of the kernel functions, the individual leaf node's activation space defined by the hyperplanes is determined depending on the data distribution.

4.3 Relationship with MLP. In ExOADT, we considered that each nonterminal node has a hyperplane defined by $(\mathbf{w}, \theta)$ where we constrain $\|\mathbf{w}\| = 1$. If we relax this constraint and make $\mathbf{w} = 0$ for all nonterminal nodes except the last but one layer (i.e., parents of the leaf nodes), then ExOADT is structurally the same as a multilayer perceptron with one hidden layer. Since $\mathbf{w} = 0$ for all previous layers, the conditioning of a path

from the root to the leaf becomes independent of $\mathbf{x}$, and therefore the hyperplanes defined in the last layer interact with each other to generate complex decision boundaries like MLP (Haykin, 1999). As we make $\|\mathbf{w}\| = 1$ for all previous layers, it imparts the network (ExOADT) to restrict the decision space confined in certain regions in each path, and these regions defined by the leaf nodes do not interact with each other. This very fact gives ExOADT more efficiency in learning the classification and function approximation tasks as compared to the multilayer perceptron, which is reflected in the results.

4.4 Relationship with HME. The hierarchical mixture of experts is structurally similar to the decision trees except that the decision trees compute the class labels by partitioning the decision space in a top-down manner, whereas the HME computes the class labels by agglomerating the decision space hierarchically upward. ExOADT follows the same concept as in the classical decision trees. However, in HME, the branching factor from child nodes toward the parent nodes is decided by the number of classes. In the case of ExOADT, the branching factor is always two; it always generates a binary tree. An analog of the difference between HME and ExOADT can be drawn from the literature of clustering (unsupervised learning): in agglomerative clustering, the smaller clusters are hierarchically agglomerated upward based on certain criteria (linkage algorithms), whereas in divisive clustering, the data set is iteratively partitioned downward based on certain criteria.

4.5 Other Issues. ExOADT inherently provides certain regularization in the approximation of the class labels and the predictor variables in the sense that the sum of activations of all leaf nodes is always unity. However, it is not clear how to select the depth of an ExOADT. One alternative is to create a growing ExOADT where we first start with an ExOADT of depth one. Once the network converges to a local optimum, we increase the depth. For example, let the connection matrix β be $[\beta_1, \beta_2]$ for one class. We can add one more level in the tree and set the connection matrix as $[\beta_1, \beta_1, \beta_2, \beta_2]$ for the same class. Independent of the parameters of the nonterminal nodes in the added level of the tree, this configuration is equivalent to the ExOADT of depth $= 1$. This is due to the fact that the sum of activation of the two sibling leaf nodes is equal to that of their parent. Therefore, the increase in depth does not cause any change in performance intially. We then train the network again with the increased depth: depth $= 2$. After convergence, we again increase the depth by adding one more level to the tree and adjust the connection matrix β in the same way as above. Thus, we create an ExOADT by iteratively increasing its depth. However, we found that the growing ExOADT does not perform better than the normal ExOADT. One reason is that when we create the growing ExOADT, its construction is similar to that of decision trees in the sense that the soft partitioning is more affected by

the local decision. The data set is first linearly partionined by an ExOADT of depth = 1 to obtain an optimum score. As the depth is increased, the network starts with these hyperplanes and then reiterates to obtain the next best solution. As this process continues, the next solution is guided by the starting point of the previous solution every time. On the other hand, if we start with an ExOADT of a certain depth (say, depth = 7), the search for the hyperplanes is activated at each layer from a random initialization. As we mentioned in the case of OADT, ExOADT exhibits performance better than the existing single classifiers on certain data sets, and it learns totally in the online mode. The learning in the online mode relieves a classifier and function approximator of the memory storage required as in the batch mode. Thus, ExOADT can be effectively used for larger data sets as well, trading memory requirements at the expense of time required to learn the class labels. In Tables 5 and 6, we report the CPU time in seconds taken by the ExOADT algorithm for classification and function approximation, respectively, including the total time for initialization and learning in 200 epochs. We report the CPU time for classification in Table 5 with 90% data as the training data and 10% as the test data. We have chosen 90% data for training since the 10-fold cross validation (as we performed) always use 90% data for training in each fold. Table 6 reports the CPU time for function approximation where we used 75% data samples as the training data and 25% as the test data. We can observe that the CPU time increases exponentially with the depth of the tree. We have noted that ExOADT converges well before 200 epochs, although we report all the results for 200 epochs. The CPU time, as reported in Tables 5 and 6, taken by the ExOADT can be reduced if the number of epochs is reduced. It is interesting to observe in Table 6 that the Friedman data set took almost 30 hours for 200 epochs and 75% data for training. However, SVR could not handle even 10% of this data set for training due to the memory constraint.

We have not investigated the pruning of ExOADT as it is often performed in the classical decision trees. In ExOADT, as discussed in section 4 (see Figure 6), the pruning mechanism can be performed by constraining the values of β_{ij}. However, simply constraining β by adding regularization constraint $\lambda\|\beta\|^2$ (λ being a Lagrangian) in the cost function may not serve the purpose, as that will create the null decision space. We can regularize the network by constraining such that the direct siblings of the leaf nodes approach having the same β values for all class labels. A suitable regularization criterior in the space of β still needs to be formulated.

5 Conclusions and Future Scope

We reported a new model for pattern classification and function approximation. This model is similar to an earlier reported model, online adaptive decision tree (OADT; Basak, 2004), extended with a regression layer; we call it ExOADT. OADT was designed only for two-class classification

Table 5: CPU Time Taken by ExOADT of Different Depths for Different Data Sets of Classification.

	CPU Time (seconds)				
Data Set	Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7
Pima	124	241	481	1029	2404
Wpbc	43	87	179	386	919
Wdbc	121	243	494	1060	2551
Bupa	48	93	188	406	1060
Iris	21	40	80	174	456
Monks1	78	151	302	655	1713
Monks2	84	163	327	712	1872
Monks3	79	152	305	660	1715
Ecoli	47	91	181	393	1020

Notes: The CPU time is for 200 epochs, including the time for initialization and learning. A 90% data set has been used for training, and 10% has been used for testing.

Table 6: CPU Time Taken by ExOADT of Different Depths for Different Data Sets of Function Approximation.

	CPU Time (seconds)				
Data Set	Depth = 3	Depth = 4	Depth = 5	Depth = 6	Depth = 7
Baseball (BB)	45	87	184	397	998
Housing (HH)	64	129	264	591	1482
Kinematics (KI)	923	1872	3782	8258	21,387
Elevator (EV)	1200	2419	4938	10,618	26,585
Ailerons (AI)	2127	5000	6477	13,618	32,306
Friedman (MARS)	4722	9449	19,350	45,243	109,138

Notes: The CPU time is for 200 epochs including the time for initialization and learning. Seventy-five percent of the data set has been used for training and the 25% for testing.

tasks. The current model can handle multiclass classification problems and effectively approximate continuous functions. We observe that ExOADT exhibits robust behavior in approximating the functions in the presence of a high amount of noise. We also observe that on certain data sets, ExOADT performs better than all single classifiers such as SVM; however, it falls short of the multiclassifier systems such as bagging and boosting in certain cases. ExOADT is able to learn completely in the online mode, and since we do not freeze the learning, it can adapt to a changing situation. In other words, if the input-output relationship of the data set changes—even if we have a dynamical relationship between the input variables and the output observation—ExOADT can capture the altered mapping between the input

and output, provided the rate of change is much slower than the rate of learning. This adaptive behavior is possible since we need not necessarily freeze the learning of ExOADT. Note that this kind of adaptivity is not possible in the classical decision tree and the ensemble classifiers.

ExOADT does not require any user-defined parameter. The only free parameter of ExOADT is its depth, and we demonstrated that after a certain depth, the performance of ExOADT does not vary to a great extent. One of the most interesting part of ExOADT is that its tree structure and depth can be smoothly changed using the different weights in the output regression layer. In OADT, we have seen that it can change only the local hyperplanes in the nonterminal nodes for a given depth of the tree, and the allocation of the class labels to the leaf nodes is also fixed. The ExOADT not only adds the flexibility of assigning class labels differentially to the leaf nodes but also adds a set of free variables that softly adapt the structure of the tree along with the local hyperplanes in the nonterminal nodes. We briefly discussed the relationship of ExOADT to other classification models such as decision tree, RBF, and multilayer perceptron. We have used only the steepest gradient descent learning; the performance can perhaps be enhanced by using smarter learning algorithms. Moreover, the performance of ExOADT can possibly be improved by having certain regularization on the output regression layer.

Appendix: Derivation of Learning Rate

We choose the optimal learning rate η_{opt} such that $E(\mathbf{x}) + \Delta E(\mathbf{x}) \to 0$ for any new pattern $\mathbf{x}$ according to the first-order approximation (line search), that is, $\Delta E \to -E$. We can approximate ΔE (for small change in E) as

$$\Delta E = \sum_i \left(\frac{\partial E}{\partial \mathbf{w}_i}\right)^T \Delta \mathbf{w}_i + \sum_i \left(\frac{\partial E}{\partial \theta_i}\right) \Delta \theta_i + \sum_k \left(\frac{\partial E}{\partial \beta_k}\right)^T \Delta \beta_k. \quad \text{(A.1)}$$

Evaluating the partials, from equations 2.12 to 2.19, we get

$$\Delta E(\mathbf{x}) = -\eta \left(m_2^2 \|\mathbf{z}\|^2 \sum_{k \in O} e_k^2 o_k^2 (1 - o_k)^2 + m_1^2 \sum_{i \notin \Omega} q_i^2 (\lambda + \|\mathbf{x}\|^2 - (\mathbf{w}_i' \mathbf{x})^2) \right). \quad \text{(A.2)}$$

Since we choose $\eta = \eta_{opt}$ such that $\Delta E(\mathbf{x}) \to -E(\mathbf{x})$, we have

$$\eta_{opt} = \frac{E}{m_2^2 \|\mathbf{z}\|^2 \sum_{k \in O} e_k^2 o_k^2 (1 - o_k)^2 + m_1^2 \sum_{i \notin \Omega} q_i^2 (\lambda + \|\mathbf{x}\|^2 - (\mathbf{w}_i' \mathbf{x})^2)}. \quad \text{(A.3)}$$

References

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

Amari, S. (1967). Theory of adaptive pattern classifiers. *IEEE Trans., EC-16*, 299–307.

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

Basak, J. (2004). Online adaptive decision trees. *Neural Computation, 16*, 1959–1981.

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: Department of Mathematical Sciences, Rensselaer Polytechnic Institute.

Blake, C., & Merz, C. (1998). *UCI repository of machine learning databases.* Available online at http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Boz, O. (2000). *Converting a trained neural network to a decision tree DecText—decision tree extractor.* Unpublished doctoral dissertation, Lehigh University. Available online at *citeseer.ist.psu.edu/boz00converting.html*.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 26*, 123–140.

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.

Broomhead, D. S., & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems, 2*, 321–355.

Buhmann, M. D. (1990). Multivariate cardinal interpolation with radial basis functions. *Constructive Approximation, 6*, 225–255.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery, 2*, 121–167.

Canu, S., Grandvalet, Y., & Rakotomamonjy, A. (2003). *SVM and kernel methods matlab toolbox.* Rouen, France: Perception Systèmes et Information, INSA de Rouen. Available online at http://asi.insa-rouen.fr/%7Earakotom/toolbox/.

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). Piscataway, NJ: IEEE Press.

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.

Cristianini, N., & Shawe-Taylor, J. (2000). *Support vector machines.* Available online at http://www.support-vector.net/software.html.

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. (2001). Greedy function approximation: A gradient boosting machine. *Annals Statistics, 29*, 1189–1232.

Friedman, J. H., Hastie, T., & Tibshirani, R. (1998). *Additive logistic regression: A statistical view of boosting* (Tech. Rep.). Palo Alto, CA: Department of Statistics: Stanford University.

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. 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). Available online at citeseer.nj.nec.com/garner95weka.html.

Geman, D., & Jedynak, B. (2001). Model-based classification trees. *IEEE Trans. Information Theory, 47*, 1075–1082.

Girosi, F., Jones, M., & Poggio, T. (1995). Regularization theory and neural network architectures. *Neural Computation, 7*, 219–269.

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

Grimson, W. E. L. (1982). A computational theory of visual surface interpolation. *Proc. of the Royal Society of London B, 298*, 395–427.

Gunn, S. R. (1998). *Support vector machines for classification and regression* (Tech. Rep. No. http://www.ecs.soton.ac.uk/∼srg/publications/pdf/SVM.pdf). Southampton: University of Southampton: Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Sceince.

Guvenir, H. A., & Uysal, I. (2000). *Bilkent University function approximation repository.* Available online at http://funapp.cs.bilkent.edu.tr/DataSets/.

Harrison, D., & Rubinfeld, D. (1978). Hedonic prices and the demand for clean air. *J. Environ. Economics and Management, 5*, 81–102.

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

Horton, P., & Nakai, K. (1996). A probablistic classification system for predicting the cellular localization sites of proteins. In *Intelligent Systems in Molecular Biology* (pp. 109–115). New York: AAAI Press.

Janickow, C. Z. (1998). Fuzzy decision trees: Issues and methods. *IEEE Trans. Systems, Man, and Cybernetics, 28*, 1–14.

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

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.

Mehta, M., Agrawal, R., & Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. In P. Apers, M. Bouzeghoub, & G. Gardarln (Eds.) *Advances in database technology* (pp. 18–32). Berlin: Springer.

Moody, J., & Darken, C. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation, 1*, 281–294.

Murphy, K. (2001). The Bayes net toolbox for MATLAB. *Computing Science and Statistics, 33*.

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.

Nakai, K., & Kanehisa, M. (1991). Expert sytem for predicting protein localization sites in gram-negative bacteria. *PROTEINS: Structure, Function, and Genetics, 11*, 95–110.

Platt, J. C. (1998). *Sequential minimal optimization: A fast algorithm for training support vector machines* (Tech. Rep. No. MSR-TR-98-14). Redmond, WA: Microsoft Research.

Poggio, T., & Girosi, F. (1990a). Networks for approximation and learning. *Proc. of the IEEE, 78*(9), 1481–1497.

Poggio, T., & Girosi, F. (1990b). Regularization algorithms for learning that are equivalent to multilayer networks. *Science, 247*, 978–982.

Poggio, T., Torre, V., & Koch, C. (1985). Computational vision and regularization theory. *Nature, 317*, 314–319.

Powell, M. J. D. (1987). Radial basis functions for multivariable interpolation: A review. In J. C. Mason & M. G. Cox (Eds.), *Algorithms for approximation*. Oxford: Clarendon Press.

Pyeatt, L. D., & Howe, A. E. (1998). *Decision tree function approximation in reinforcement learning* (Tech. Rep. No. CS-98-112). Fort Collins, CO: Colorado State University.

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 Mateo, CA: Morgan Kauffman.

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.

Schapire, R. E., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning, 37*, 297–336.

Smola, A. J., & Schölkopf, B. (1998a). On a kernel-based method for pattern recognition, regression, function approximation and operator inversion. *Algorithmica, 22*, 211–231.

Smola, A. J., & Schölkopf, B. (1998b). *A tutorial on support vector regression* (Tech. Rep. No. NC-TR-98-030). London: Royal Holloway College, University of London, NeuroCOLT.

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). Piscataway, NJ: IEEE.

Suárez, A., & Lutsko, J. F. (1999). Globally optimal fuzzy decision trees for classification and regression. *IEEE Trans. Pattern Analysis and Machine Intelligence, 21*, 1297–1311.

Tikhonov, A. N., & Arsenin, V. Y. (1977). *Solutions of ill-posed problems.* Washington, DC: W. H. Winston.

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

Uther, W. T. B., & Veloso, M. M. (1998). Tree based discretization for continuous state space reinforcement learning. In *Proc. Sixteenth National Conference on Artificial Intelligence (AAAI-98).* Cambridge, MA: MIT Press.

Vapnik, V. (1998). *Statistical learning theory.* New York, USA: Springer-Verlag.

Vapnik, V., Golowich, S., & Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing. In M. Mozer, M. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems, 9* (pp. 281–287). Cambridge, MA: MIT Press.

Wang, X., & Dietterich, T. (1999). Efficient value function approximation using regression trees. In T. Dean (Ed.), *Proceedings of the IJCAI-99 Workshop on Statistical Machine Learning for Large-Scale Optimization.* San Francisco: Morgan Kaufmann.

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 *Proc. 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. In *Research and development in information retrieval* (pp. 46–54). New York: ACM.

Received April 11, 2005; accepted December 16, 2005.