International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 19, No. 3 (2005) 323–353
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

LINEAR DISCRIMINANT TREES

OLCAY TANER YILDIZ* and ETHEM ALPAYDIN†
Department of Computer Engineering
Boğaziçi University
34342 Istanbul, Turkey
**yildizol@cmpe.boun.edu.tr*
†*alpaydin@boun.edu.tr*

We discuss and test empirically the effects of six dimensions along which existing decision tree induction algorithms differ. These are: Node type (univariate versus multivariate), branching factor (two or more), grouping of classes into two if the tree is binary, error (impurity) measure, and the methods for minimization to find the best split vector and threshold. We then propose a new decision tree induction method that we name linear discriminant trees (LDT) which uses the best combination of these criteria in terms of accuracy, simplicity and learning time. This tree induction method can be univariate or multivariate. The method has a supervised outer optimization layer for converting a $K > 2$-class problem into a sequence of two-class problems and each two-class problem is solved analytically using Fisher's Linear Discriminant Analysis (LDA). On twenty datasets from the UCI repository, we compare the linear discriminant trees with the univariate decision tree methods C4.5 and C5.0, multivariate decision tree methods CART, OC1, QUEST, neural trees and LMDT. Our proposed linear discriminant trees learn fast, are accurate, and the trees generated are small.

Keywords: Decision trees; linear discriminant analysis; neural networks; multivariate analysis.

1. Introduction

A decision tree is made up of internal decision nodes and terminal leaves. When used for classification, the leaves carry the label of one of K classes, $\mathcal{C}_1, \ldots, \mathcal{C}_K$. The input vector is composed of d attributes, $\boldsymbol{x} = [x_1, \ldots, x_d]^T$, which may be numeric (ordered) or discrete (unordered). The decision nodes check for a particular input condition. Given an input to classify, starting from the root node, one applies the condition at each internal node and takes one of the branches depending on the outcome. This process is repeated recursively until a leaf node is hit at which point the label of the leaf constitutes the output.

In the case of a binary tree, a decision node m implements a boolean function

$$f_m(\boldsymbol{x}|\Phi) > 0 \tag{1}$$

parameterized by Φ, with two outcomes, true and false, labeling the two branches, left and right. Each $f_m(\boldsymbol{x})$ defines a discriminant in the d-dimensional input space dividing it into two. Different decision tree methods assume different models for f_m and the model class defines the shape of the discriminant. The *multivariate linear* model is

$$f_m(\boldsymbol{x}|\boldsymbol{w}_m, w_{m0}) = \sum_{j=1}^{d} w_{mj}x_j + w_{m0} = \boldsymbol{w}_m^T\boldsymbol{x} + w_{m0} \tag{2}$$

where $\boldsymbol{w}_m$ defines the split direction and w_{m0} is the threshold once $\boldsymbol{x}$ are projected onto $\boldsymbol{w}_m$.

A discrete attribute should be represented numerically, e.g. by 1-of-N encoding, before it can be used for a multivariate test. Similarly, a missing attribute should be filled in, for example, with the most likely value or the average, before it can be used at a node.

A special, simpler case of the multivariate linear model is the *univariate* model where a single input attribute is used:

$$f_m(\boldsymbol{x}|j, w_{m0}) = x_j + w_{m0}. \tag{3}$$

If the multivariate linear model is not complex enough, one can use a *multivariate nonlinear* model. Among the various ways in which one can write a nonlinear model, one possibility is to write it as a linear sum of nonlinear basis functions

$$f_m(\boldsymbol{x}|w_{m0}) = \sum_{j=1}^{J} w_{mj}\phi(\boldsymbol{x}|\boldsymbol{v}_{mj}) + w_{m0} \tag{4}$$

where $\phi(\cdot)$ are the nonlinear basis functions parameterized by $\boldsymbol{v}_{mj}$. This can be implemented as a multilayer perceptron where J is the number of hidden units, $\phi(\cdot)$ is the sigmoid function at the hidden units, and $\boldsymbol{v}_{mj}$, w_{mj} are the first and second layer weights, respectively.

1.1. *Tuning node complexity*

What any classifier, and in this case the decision tree, does is approximate the real (unknown) discriminant. With univariate nodes, we are limited to a piecewise approximation using axis-aligned hyperplanes. With multivariate linear nodes, we can use arbitrary hyperplanes and approximate the discriminant better.

The branching factor, i.e. the number of children of an internal node, has a similar effect in that it defines the number of discriminants that a node defines. A binary decision node, with a branching factor of two, defines one discriminant and separates the data into two. Thus with binary decision nodes, a K-class problem is converted into a sequence of 2-class problems. A K-way node separates the data into K parts at one time.

Thus there is a dependency between the complexity of a node, the branching factor, and the size of the tree. With complex nodes and large number of branches, the tree may be quite small but difficult to interpret. A tree with simple nodes and low branching factor may be large but interpretable (e.g. can be converted to simple IF-THEN rules); such a tree also better matches the underlying divide-and-conquer methodology of a tree.

A complex model with a larger number of parameters requires a larger training dataset and risks overfitting on a small amount of data. Hence one should be careful in tuning the complexity of a node with the properties of the data at hand. For example, using multivariate linear nodes, we are assuming that the input space can be divided using hyperplanes into localized regions (volumes) where classes, or groups of classes are linearly separable.

1.2. *Training the tree from data*

Given a training set $\mathcal{X}$, finding the smallest decision tree that classifies $\mathcal{X}$ correctly is NP-hard.[14] For large training sets and input dimensions, even for the univariate case, one cannot exhaustively search through the complete space of possible decision trees. Decision tree algorithms are thus greedy in that at each step, we decide on one decision node, and then continue recursively with the partition induced by the node. We look for f_m that best splits the data hitting node m, starting with the complete dataset in deciding on the root node. Example measures used to quantify the goodness of a split are entropy[16] and the Gini index.[2] Murthy *et al.*[14] described some other impurity indices. Our results and those of previous researchers indicate that there is no significant difference between these impurity measures.[15]

It has been shown that for the multivariate linear case the problem of finding the optimal split at a single node when optimality is measured in terms of misclassification errors is NP-hard.[14] The problem of finding the best split is then an optimization problem to find the best $(\boldsymbol{w}_m, w_{m0})$ pair that minimizes impurity (or some other error function). As one cannot do an exhaustive search over all possible values in the multivariate case, one resorts to an iterative local search algorithm, which does not guarantee optimality and may get stuck in local optima:

(1) Linear Discriminant Analysis was first used in Friedman[6] for constructing decision trees. The algorithm has binary splits at each node, where a split is like in C4.5, i.e. $x_i < w_0$ but x_i can be an original variable, transgenerated, or adaptive. Linear discriminant analysis is applied to construct an adaptive variable. Kolmogorof–Smirnoff distance is used as the error measure. When there are more than $K > 2$ classes, it converts the problem into K different subproblems, where each subproblem separates one class from others.

(2) In CART (Classification and Regression Trees),[2] parameter adaptation is through backfitting: at each step, all the coefficients w_{mj} except one is fixed and that coefficient is tuned for possible improvement in terms of impurity. One cycles through all j until there is no further improvement.

(3) In FACT (Fast Algorithm for Classification Trees),[12] with K classes a node can have K branches. Each branch has its modified linear discriminant function calculated using Linear Discriminant Analysis (LDA) and an instance is channeled to the ith branch to minimize an estimated expected risk.

(4) In Neural Trees,[8] each decision node uses a multilayer perceptron which implements multivariate *nonlinear* decision trees [Eq. (4)]. The nodes are binary and K classes are grouped into two using the supervised exchange heuristic which we discuss in Sec. 3. Backpropagation is used to learn parameters. To be able to compare with other multivariate linear methods, in our simulations, we replace the multilayer perceptron with a single layer, linear perceptron, and name such a tree ID-LP.

(5) In OC1 (Oblique Classifier),[14] an extension to CART is made to get out of the local optima. A small random vector is added to $\boldsymbol{w}_m$ once there is convergence through backfitting. Adding a vector perturbs all coefficients together and makes a conjugate jump in the coefficient space. Another extension proposed is to run the method several (20–50) times and choose the best solution in terms of impurity.

(6) In LMDT (Linear Machine Decision Trees),[4] with K classes, as in FACT, a node is allowed to have K branches. For each class, i.e. branch, there is a vector of parameters, and the node implements a K-way split. There is an iterative algorithm that adjusts the parameters of classes to minimize the number of misclassifications, rather than an impurity measure as entropy or Gini.

(7) QUEST (Quick Unbiased Efficient Statistical Tree)[11] is a revised version of FACT and uses binary splits at each decision node. It solves the problem of dividing K classes into two classes by using unsupervised 2-means clustering on the class means of the data. QUEST also differs from FACT in the way that it does not assume equal variances and uses Quadratic Discriminant Analysis (QDA) to find the two roots for the split point and uses the appropriate one.

(8) LTREE (Linear Tree)[7] is a multivariate decision tree algorithm with binary splits. LTREE uses linear discriminant analysis to construct new features, which are linear combinations of the original features. For all constructed features, the best split is found using C4.5's exhaustive search technique. Best of these is selected to create the two children of the current node. These new constructed features can also be used down the tree in the children of that node. Extensions of this algorithm uses quadratic discriminant QTREE and logistic discriminant LGTREE for constructing new features.

(9) CRUISE[10] (Classification Rule With Unbiased Interaction Selection and Estimation) is a multivariate algorithm with K-way nodes. Like FACT, CRUISE finds $K-1$ splits using linear discriminant analysis. The departure from FACT occurs when the split assigns the same class to all its K children. Because such a split is not useful, the best next class is chosen. Another departure occurs while assigning a class to a leaf: when there are two or more classes which have the same number of instances in that leaf, FACT selects randomly one

Table 1. Multivariate decision tree construction algorithms classified according to six dimensions, which are node type (*uni*variate, *lin*ear multivariate, *non*linear multivariate), branching factor, grouping algorithm for grouping $K > 2$ classes into two, error measure minimized, minimization method to find the direction vector $\boldsymbol{w}$, and minimization method to find the split point w_0.

Algorithm	Node	Br	Group	Error	Search $\boldsymbol{w}$	Search w_0
C4.5	Uni	2	—	Impurity	—	Exhaustive
Friedman's	Uni/Lin	2	—	Kolm-Smir	Analytical	Analytical
CART	Lin	2	—	Impurity	Backfitting	Exhaustive
FACT	Uni/Lin	K	—	Fisher's	Analytical	Analytical
ID-LP/MLP	Lin/Non	2	Sup	MSE	Gradient	Gradient
OC1	Lin	2	—	Info Gain	Hill climb	Exhaustive
LMDT	Lin	K	—	Misclass	Thermal	Thermal
QUEST	Uni/Lin	2	Unsup	Fisher's	Analytical	Analytical
Ltree	Lin	2	—	Info Gain	Analytical	Exhaustive
Cruise	Uni/Lin	K	—	Fisher's	Analytical	Analytical
LDT	Uni/Lin	2	Sup	Fisher's	Analytical	Analytical

of them but CRUISE selects the class which has not been assigned to any leaf node.

Because the univariate is a special case of the multivariate, most of these multivariate algorithms have their univariate versions, sometimes with slight modifications. In Table 1, we compare the algorithms in terms of the six dimensions along which these algorithms differ. These are: Node type (univariate versus multivariate), branching factor (two or K, number of classes), grouping of classes into two if the tree is binary, error (impurity) measure, and the methods for minimization to find the best split vector and split point.

1.3. *Feature extraction*

The complexity of the search and the risk of overfitting can be reduced by decreasing dimensionality through feature extraction at each node. This is different from the usual approach in pattern recognition where feature extraction is done as a separate process before the classifier is trained. In tree induction, it is integrated into the training of the classifier and is done separately for each subproblem (decision node). This makes sense as in the localized region bounded by the discriminants defined by the nodes preceding that node in the tree, certain dimensions may not vary significantly and the subset of the data reaching that node may effectively lie in a lower-dimensional subspace, or certain dimensions which are not significant globally may be quite important locally or vice versa.

In the case of a univariate node (which is feature extraction from d to one dimension), there are d possible $\boldsymbol{w}_m$ that can be used and $N - 1$ possible w_{j0}, so one can do an exhaustive search over the $d \times (N - 1)$ possible $(\boldsymbol{w}_m, w_{m0})$. This is done in the C4.5 algorithm.[17] In its predecessor ID3 algorithm,[16] the features are symbolic and there is only one way to split at a feature and thus there are only d possibilities.

In a multivariate node, one can do feature selection and by reducing dimensionality, the complexity of search is reduced by setting the coefficients, w_{mj}, of the discarded dimensions to zero. One can do sequential backward search for feature selection and get rid of features whose loss do not increase impurity significantly; the other method is sequential forward search where one adds features one by one until their addition does not significantly decrease impurity.[2,4]

1.4. *Pruning*

A greedy algorithm is a local search method where at each step, one tries to make the best decision and proceeds to the next decision, never backtracking and reevaluating a decision after it has been made. Similarly in decision tree induction, once a decision node is fixed, it cannot be changed after its children have been created. This may cause suboptimal trees where, for example, subtrees are replicated. The only exception is the *pruning* of the tree.

In pruning, we consider replacing a subtree with a leaf node labeled with the class most heavily represented among the instances that are covered by the subtree. If there is overfitting, we expect the more complex subtree to learn the noise and perform worse than the simple leaf. If this is indeed the case on a validation set different from the training set, then the subtree is replaced by the leaf. Otherwise it is kept. It makes sense to start with the smaller subtrees closer to leaves and proceed up towards the root.

This process is called *post-pruning* to differentiate it from *pre-pruning*. In post-pruning, the tree is constructed until there is no misclassification error and then pruned simpler. In pre-pruning, the tree is not fully constructed until zero training error but is kept simple by early termination. At any node, if the dataset reaching that node is small, even if it is not pure, it is not further split and a leaf node is created instead of growing a subtree. Pre-pruning is faster. Post-pruning may be more accurate but is slower.

Pruning uses validation to finetune model complexity. Another possibility is to use Minimum Description Length to balance the complexity of the tree with the complexity of the data it describes.[18]

1.5. *Organization of the paper*

This paper is organized as follows. Our proposed method *linear discriminant trees* (LDT) is proposed in Sec. 2. Section 3 discusses two supervised methods by which $K > 2$ classes can be divided into two so that the binary classifier through a two-class discriminant analysis can be employed. Dataset details and comparison criteria are given in Sec. 4. Section 5 compares the univariate version of LDT with C4.5 and the univariate version of QUEST. Results with the multivariate version of LDT are given in Sec. 6. In Sec. 7, we discuss the effect of univariate versus multivariate nodes in a tree. In Sec. 8, we compare our proposed method LDT with C4.5, CART, OC1, LMDT, neural tree ID-LP, and QUEST in terms of accuracy, tree size and learning time. We conclude and discuss future work in Sec. 9.

2. Linear Discriminant Trees

We hereby propose linear discriminant trees which use a statistical approach to quickly determine the set of coefficients in a linear decision node. Finding the best split with Fisher's Linear Discriminant Analysis (LDA)[5] is done as a nested optimization problem, as in neural trees.[8] In the inner optimization problem, Fisher's linear discriminant replaces the neural network for finding a good split for the given two distinct groups of classes. The outer optimization problem is identical where at each node m, we search for the best split of K classes into two groups, C_m^L and C_m^R, as will be discussed in Sec. 3.

2.1. *Multivariate case*

The inner optimization problem of separating the two, left and right, groups is solved by linear discriminant analysis (LDA)[5] at each node of the decision tree. Throughout this section, keep in mind that the whole discussion is conditioned on one node and the calculations are repeated for each node as the tree is created in a greedy manner as explained in Sec. 1.2.

In LDA, we look for the direction, as defined by $\boldsymbol{w}$, such that when the data is projected onto this single dimension, the two groups are as well separated as possible. Well-separation implies that after they are projected; (i) the distance between the two means is large, and (ii) the scatter of instances around each mean are small.

Let us denote by $\boldsymbol{m}_L$ and m_L the means of instances from C_L before and after projection, respectively. Note that $\boldsymbol{m}_L \in \Re^d$ and $m_L \in \Re$.

$$m_L = \frac{1}{n_L} \sum_{\boldsymbol{x}^t \in C_L} \boldsymbol{w}^T \boldsymbol{x}^t = \boldsymbol{w}^T \boldsymbol{m}_L. \tag{5}$$

There are n_L samples in the left class group and n_R samples in the right class group. The *scatter* of samples from C_L after projection is

$$s_L^2 = \sum_{\boldsymbol{x}^t \in C_L} (\boldsymbol{w}^T \boldsymbol{x}^t - m_L)^2 \tag{6}$$

$$= \boldsymbol{w}^T \mathbf{S}_L \boldsymbol{w} \tag{7}$$

where

$$\mathbf{S}_L = \sum_{\boldsymbol{x}^t \in C_L} (\boldsymbol{x}^t - \boldsymbol{m}_L)(\boldsymbol{x}^t - \boldsymbol{m}_L)^T \tag{8}$$

and $s_L^2/(n_L - 1)$ is the variance of instances from C_L after projection onto $\boldsymbol{w}$. $\boldsymbol{m}_R$, m_R, $\mathbf{S}_R$, n_R and s_R^2 are similarly defined. The total within-class scatter matrix is $\mathbf{S} = \mathbf{S}_L + \mathbf{S}_R$.

Fisher's linear discriminant is

$$\boldsymbol{w} = \mathbf{S}^{-1}(\boldsymbol{m}_L - \boldsymbol{m}_R). \tag{9}$$

Multiplying $\boldsymbol{x}$ with $\boldsymbol{w}$, we make a projection from the d-dimensional space to one dimension where we can find a suitable threshold to separate the two groups. Assuming that the groups are normally distributed with equal variances, one can solve for the optimal threshold w_0 as

$$w_0 = -\frac{1}{2}(\boldsymbol{m}_L + \boldsymbol{m}_R)^T \mathbf{S}^{-1}(\boldsymbol{m}_L - \boldsymbol{m}_R) - \log \frac{n_L}{n_R}. \tag{10}$$

Another possibility is to find w_0 by testing for all possible split positions to minimize an impurity measure like entropy or Gini, or an error measure like the number of misclassifications. This iterative approach makes no assumption of normality or homogeneity but may be time consuming on large training sets.

So in our proposed method which we name *LDTm*, at each node m, we first divide $K > 2$ classes into two groups as C_m^L and C_m^R by an appropriate class selection procedure (as defined in Sec. 3), then the inner optimization procedure is carried out on $\mathcal{X}_m$, the sample at node m, by LDA to find the linear split, i.e. $\boldsymbol{w}_m$ using Eq. (9) and w_{m0} using Eq. (10). There is no iterative training at a node to find $(\boldsymbol{w}_m, w_{m0})$ (as with CART, OC1, LMDT, or neural trees) but only analytical calculations and we expect less training time with our proposed linear discriminant trees than with other multivariate tree induction methods.

Sample multivariate split by our proposed method LDTm on the *Iris* dataset (using only two dimensions for graphical purposes) is given in Fig. 1.

2.2. *Problem of singularity*

If there is a linear dependency between two or more features then the total within-class scatter matrix, $\mathbf{S}$, becomes singular and $\mathbf{S}^{-1}$ does not exist. A possible solution is *principal component analysis* (PCA),[19] where we find the eigenvectors and eigenvalues of the matrix $\mathbf{S}$ and get rid of the eigenvectors with zero (or very small) eigenvalues. Let us say that the eigenvalues of the matrix $\mathbf{S}$ are $\lambda_1, \ldots, \lambda_d$, sorted in decreasing order, and $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_d$ are the associated eigenvectors. We find the k eigenvectors that explain more than ϵ of the variance

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \cdots + \lambda_k + \cdots + \lambda_d} > \epsilon. \tag{11}$$

The instances are then projected from the original d-dimensional space onto the new k-dimensional space defined by multiplying with the leading k eigenvectors

$$z_j^t = \boldsymbol{c}_j^T \boldsymbol{x}^t, \quad j = 1, \ldots, k. \tag{12}$$

We then do LDA in this new space of z_j and find the new $\mathbf{S}_z$, means, and calculate the split. Because the eigenvectors are orthogonal, the new $\mathbf{S}_z$ has an inverse. Note that because PCA is also a linear mapping, the eigenvectors $\boldsymbol{c}_i$ can

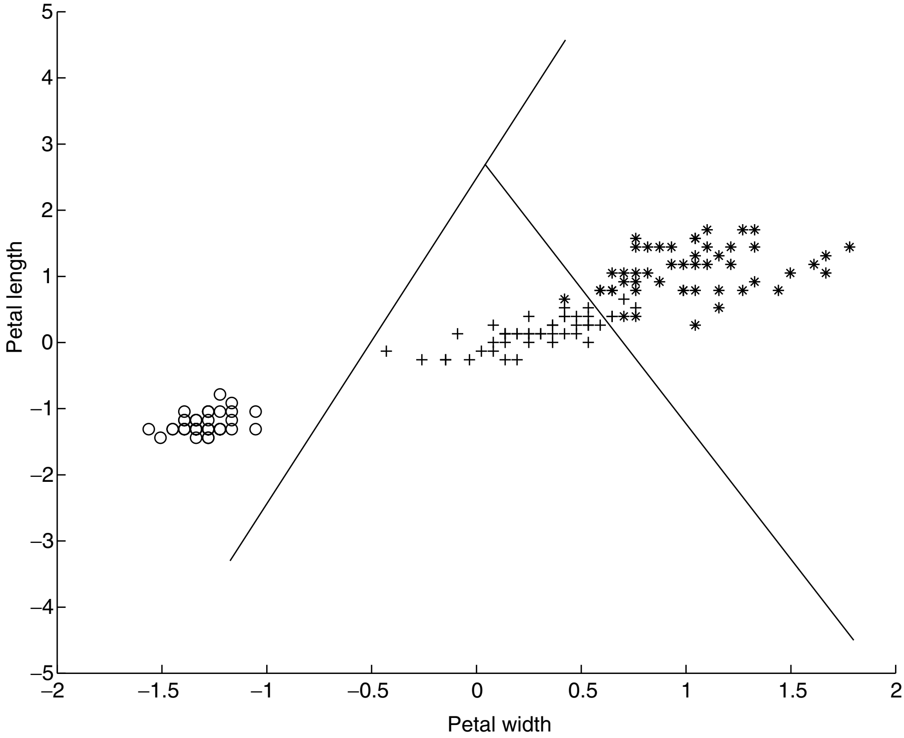

Fig. 1. Sample multivariate splits generated by LDTm for *Iris* dataset.

be incorporated into the w_{mj} so that there is no increase in memory need. It is always a good idea to normalize x_j to zero mean and unit variance before doing PCA as otherwise the variance of the original dimensions, and not the correlations, would dominate. After the projection is done, there is no need to pass over the entire training set to compute the new covariance matrix, which is costly as it is $\mathcal{O}(Nk^2)$. If $\mathbf{A}$ is the k by d matrix whose rows are the eigenvectors, the new k by k covariance matrix, $\mathbf{S}_z$, can be found as

$$\mathbf{S}_z = \mathbf{A}\mathbf{S}\mathbf{A}^T \tag{13}$$

also the new left and right mean vectors can be found as

$$\boldsymbol{m}_{Lz} = \mathbf{A}\boldsymbol{m}_{Lx} \tag{14}$$

where $\boldsymbol{m}_{Lx}$ and $\boldsymbol{m}_{Lz}$ are the mean vectors of the left group before and after applying PCA, respectively.

This process is also a form of feature extraction and reduces dimensionality from d to k. In subset selection used in CART or LMDT for feature extraction, a subset k of the original dimensions are kept. Here we find k new dimensions, z_j, that are linear combinations of the original d dimensions, $\boldsymbol{x}$.

2.3. *Univariate case*

Note that after projecting the multivariate data on to the Fisher's linear discriminant we have a univariate problem where the split point is found using Eq. (10). We can use the same procedure to generate univariate decision nodes directly working on the original features. Given a split of K classes into two, as left and right groups, and assuming they are normally distributed, the split points are the roots of the equation

$$ax^2 + bx + c = 0 \tag{15}$$

where

$$\begin{aligned} a &= s_L^2 - s_R^2 \\ b &= 2(m_L s_R^2 - m_R s_L^2) \\ c &= (m_R s_L)^2 - (m_L s_R)^2 + 2 s_L^2 s_R^2 \log \frac{n_L s_R}{n_R s_L}. \end{aligned} \tag{16}$$

If the two groups have the same variance (as assumed in the multivariate case), there is only one root [Eq. (10)]. If the variances are different, there are two roots and we use the one which is between the two means. If neither of the two roots is between the means or there are no roots of the quadratic equation, we choose the middle point of the two means as the split point.

Since this is a univariate algorithm, we may use any feature of the dataset for finding a split at each node. We find the corresponding splits for each feature and select the one with the minimum entropy.

If the feature is discrete, we convert it to numeric using 1-of-N encoding and search for the best split in all. This keeps the tree binary and in our simulations, this works as well as making a N-way split as done in C4.5.

Figure 2 shows the splits produced by the univariate algorithm LDTu on the two-dimensional subset of the *Iris* dataset. This tree has the same accuracy as the multivariate tree of Fig. 1, indicating that going to the multivariate case may not be necessary in some cases.

3. Class Separation

In Sec. 2, one detail we seem to have skipped over is the following: the model of Eq. (1) makes a binary decision and we may have $K > 2$ classes and thus at each node, we need to find the best way of partitioning K classes into two subsets. If there are K classes available at a node then there are $2^{K-1} - 1$ distinct partitions available. This is because K bits can be written in 2^K different ways; half of them are complements of the other half and the binary string of all 0s (or all 1s) is useless.

So because the number of available partitions grows exponentially, we cannot test for all possible partitions and need heuristics to get a reasonable partition in polynomial time.

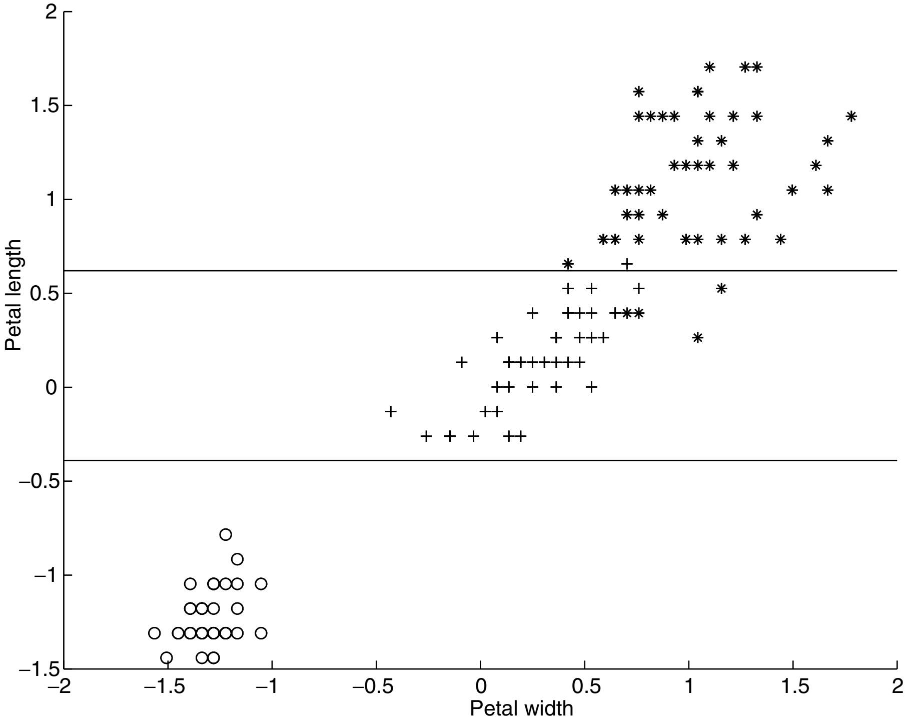

Fig. 2. Univariate splits generated by LDTu for *Iris* dataset. The first split is at −0.39.

3.1. *Selection method*

The first method we use in class separation is the selection method, which is a depth-first search method with no backtracking. Let m be a decision node and $C = \{C_1, \dots, C_K\}$ be the set of K classes at node m.

(1) Select two classes C_i and C_j at random and put one in C_m^L and the other in C_m^R.
(2) Train the discriminant (univariate or multivariate) with the given partition, e.g. using LDA. Do not consider the instances of other classes yet.
(3) For other classes in the class list, search for the class C_k that is best placed into one of the partitions, as quantified by an impurity measure like entropy.
(4) Add C_k to C_m^L or C_m^R depending on which side its instances fall more and continue adding classes one by one using steps 2 to 4 until no more classes are left.

This algorithm is sensitive to the initial class partition due to its depth-first nature. As a heuristic, we take the two furthest classes, as measured by the distance between their means, as the initial two classes. The algorithm traces steps 2 to 4 $K - 2$ times. So its complexity is $\mathcal{O}(K)$.

This is an agglomerative algorithm where at each iteration a class is merged with one of the two partitions. Its difference from agglomerative clustering is that this decision is based on a supervised measure like impurity or entropy and not on an unsupervised measure such as the distance in the input space.

3.2. *Exchange method*

The second class separation technique is the exchange method proposed in Ref. 8, for neural trees. This is a local search with backtracking. Let m be a decision node and $C = \{C_1, \ldots, C_K\}$ be the set of K classes at node m.

(1) Select an initial partition of C into C_m^L and C_m^R, both containing $K/2$ classes.
(2) Train the discriminant (univariate or multivariate) to separate C_m^L and C_m^R. Compute the entropy E_0 with information gain criterion.
(3) For each of the classes $k \in \{C_1, \ldots, C_K\}$ form the partitions $C_L(k)$ and $C_R(k)$ by changing the assignment of the class C_k in the partitions C_m^L and C_m^R, i.e. if it is in C_m^L, put it in C_m^R and vice versa.
(4) Find the discriminant for the partitions $C_L(k)$ and $C_R(k)$. Compute the entropy E_k and the decrease in the entropy $\Delta E_k = E_k - E_0$.
(5) Let ΔE^* be the maximum of the impurity decreases over all possible k and k^* be the k causing the largest decrease. If this impurity decrease is less than zero then exit else set $C_L = C_L(k^*), C_R = C_R(k^*)$, and go to step 2.

In order to get a good solution in reasonable time we use a heuristic to start in step 1, instead of starting randomly. The two classes C_i and C_j with the maximum intermean distance are found and placed into C_m^L and C_m^R. For each of the classes $k \in \{C_1, \ldots, C_m\}$, we find the one with the minimum intermean distance to C_m^L or C_m^R and then we put it into that group. We repeat this second step until no more classes are left.

The exchange method has complexity $\mathcal{O}(f \times K)$ where f is the number of iterations.

4. Datasets and Comparison Criteria

We have performed experiments on 20 datasets from the UCI repository.[13] Table 2 describes the properties of the datasets. We have implemented all the algorithms ourselves, except for C5.0, and our code is available for academic purposes.[9]

The three criteria we used for comparison are accuracy on the test set, tree size measured in terms of the number of nodes in the tree and learning time in seconds on a Pentium III-450 PC. For each method, we performed five two-fold cross-validation runs on each dataset and used the 5×2 cv F Test[1] to check for statistically significant difference. The results of the ten runs are then averaged and we report the mean and standard deviations, as well as checking for significance using the test. In tables, "$>$", "$\gg$" correspond to statistically significant difference with 95 and 99% confidence respectively.

Table 2. Description of the datasets.

Set	Classes	Instances	Features	Missing Values	Feature Values
BREast	2	699	10	Yes	numeric
BUPa	2	345	7	No	numeric
CAR	4	1728	7	No	symbolic
CYLinder	2	541	36	Yes	mixed
DERmatology	6	366	35	Yes	numeric
ECOli	8	336	8	No	numeric
FLAre	3	323	11	No	mixed
GLAss	7	214	10	No	numeric
HEPatitis	2	155	20	Yes	numeric
HORse	2	368	27	Yes	mixed
IRIs	3	150	5	No	numeric
IROnosphere	2	351	35	No	numeric
MONks	2	432	7	No	numeric
MUShroom	2	8124	23	Yes	symbolic
OPTdigits	10	3823	64	No	numeric
PENdigits	10	7494	16	No	numeric
SEGment	7	2310	19	No	numeric
VOTe	2	435	17	Yes	symbolic
WINe	3	178	14	No	numeric
ZOO	7	101	17	No	numeric

If in one comparison there are more than two methods to compare, we give two tables where in the first table the raw results are shown. When there are more than two methods, there can be various possible orderings and we give a second table which contains pairwise comparisons; the entry (i, j) in this second table gives the number of datasets (out of 20) on which method i is statistically significantly better than method j with at least 95% confidence. In the second table, row and column sums are also given. The row sum gives the number of datasets out of 20 where the algorithm on the row outperforms at least one of the other algorithms. The column sum gives the number of datasets where the algorithm on the column is outperformed by at least one of the other algorithms.

5. Comparison of Univariate Techniques

We compare the univariate version of our decision tree induction method (LDTu) in terms of accuracy, tree size and learning time with C4.5, C5.0, and the univariate version of QUEST. LDTu(1) assumes equal variances, LDTu(2) uses two different variances for the left and right groups — this is actually QDA. In both cases we use exchange algorithm for outer optimization. QUEST uses 2-means clustering in outer optimization of the problem and assumes two different variances for the two groups (QDA). C5.0 is a recent, improved version of C4.5 that is commercially available,[20] and we used its evaluation version that runs on datasets having less than 400 instances.

In terms of accuracy (Table 3), LDTu and QUEST are slightly better than C4.5. If we look at Table 4, we also see that in terms of the tree size (the number

Table 3. Comparison of accuracies of different univariate techniques in terms of percentage.

Set	C4.5	C5.0	LDTu(1)	LDTu(2)	QUEST
BRE	94.68 ± 1.84	94.39 ± 1.45	94.16 ± 1.40	93.74 ± 1.65	93.76 ± 1.31
BUP	62.84 ± 3.39	62.02 ± 4.89	63.48 ± 4.85	62.09 ± 4.34	62.09 ± 4.15
CAR	86.08 ± 1.60	—	92.80 ± 1.51	92.71 ± 1.63	92.82 ± 1.65
CYL	67.29 ± 4.67	—	67.36 ± 2.75	67.84 ± 1.66	65.55 ± 5.00
DER	92.51 ± 2.42	94.37 ± 1.84	92.46 ± 2.17	92.40 ± 2.25	91.31 ± 5.53
ECO	78.27 ± 4.00	80.60 ± 4.37	78.69 ± 3.91	78.39 ± 3.26	74.94 ± 6.18
FLA	88.35 ± 2.55	88.48 ± 2.49	88.60 ± 2.72	88.60 ± 2.72	88.66 ± 2.81
GLA	60.09 ± 5.52	62.15 ± 4.89	60.00 ± 5.97	58.79 ± 5.75	59.07 ± 7.36
HEP	78.95 ± 4.48	79.99 ± 2.64	78.18 ± 5.22	77.81 ± 5.39	77.81 ± 5.39
HOR	73.80 ± 6.74	84.67 ± 2.49	83.86 ± 4.50	83.70 ± 3.88	83.86 ± 4.08
IRI	92.93 ± 3.33	92.93 ± 2.67	93.20 ± 2.22	94.00 ± 1.91	93.87 ± 2.01
IRO	86.15 ± 3.72	90.08 ± 2.35	85.52 ± 3.39	84.33 ± 4.06	84.73 ± 3.52
MON	89.81 ± 7.72	95.00 ± 10.58	86.20 ± 6.33	82.92 ± 8.33	78.84 ± 8.76
MUS	99.87 ± 0.11	—	99.86 ± 0.15	95.89 ± 5.12	95.89 ± 5.12
OPT	84.81 ± 0.84	—	84.27 ± 1.25	84.73 ± 1.04	83.46 ± 0.84
PEN	92.52 ± 0.60	—	93.24 ± 0.70	92.91 ± 0.51	92.44 ± 0.86
SEG	92.03 ± 0.93	—	92.40 ± 0.68	92.66 ± 0.78	92.24 ± 1.10
VOT	95.63 ± 0.66	95.26 ± 0.87	95.63 ± 0.66	95.63 ± 0.66	95.63 ± 0.66
WIN	86.63 ± 1.94	89.21 ± 2.50	87.98 ± 3.59	85.62 ± 5.94	86.85 ± 2.90
ZOO	82.97 ± 7.36	89.08 ± 6.22	82.97 ± 7.36	82.97 ± 7.36	83.95 ± 8.11

Method	C4.5	C5.0	LDTu(1)	LDTu(2)	QUEST	$\sum$
C4.5	—	0	0	0	0	0
C5.0	0	—	1	1	2	2
LDTu(1)	2	0	—	0	0	2
LDTu(2)	1	0	0	—	0	1
QUEST	1	0	0	0	—	1
$\sum$	2	0	1	1	2	

of nodes in the tree), the two also seem to generate smaller trees than C4.5. In terms of learning time (Table 5), we see that both LDTu and QUEST run faster than C4.5 as the two former calculate for the split point analytically whereas C4.5 tries all possible $N_m - 1$ splits. In this univariate case, QUEST runs faster than LDTu as the 2-means clustering is faster than the exchange heuristic. On the small datasets where we could run it, C5.0 is not significantly more accurate than C4.5 but is slightly more accurate than LDTu and QUEST; in terms of tree size, C4.5 and C5.0 trees seem comparable.

We see that there is no significant difference between the two versions of LDTu and thus assuming equal variances is not harmful, even for the univariate case, if the splitting into two is done using a supervised measure, as the exchange heuristic does.

6. Results with Multivariate Linear Discriminant Trees

Before comparing it with other multivariate linear trees, we investigate the effect of various choices on the multivariate version of our algorithm, LDTm.

Table 4. Comparison of tree sizes measured by the number of nodes (decision nodes + leaves) of different univariate techniques.

Set	C4.5	C5.0	LDTu(1)	LDTu(2)	QUEST
BRE	13.0 ± 5.0	10.8 ± 3.7	9.8 ± 3.7	10.8 ± 4.5	12.8 ± 6.3
BUP	17.4 ± 12.5	20.4 ± 10.2	26.0 ± 10.9	17.2 ± 10.2	17.6 ± 7.5
CAR	89.4 ± 10.4	—	61.2 ± 7.3	60.6 ± 9.2	60.4 ± 10.6
CYL	20.8 ± 8.7	—	25.6 ± 13.7	27.0 ± 12.9	22.0 ± 12.9
DER	12.4 ± 1.3	14.0 ± 1.4	12.4 ± 1.3	12.4 ± 1.3	12.2 ± 1.7
ECO	14.2 ± 4.6	14.2 ± 4.2	17.2 ± 10.7	17.6 ± 9.5	15.2 ± 6.0
FLA	4.7 ± 6.0	1.4 ± 0.8	1.6 ± 1.3	1.6 ± 1.3	2.4 ± 2.7
GLA	14.2 ± 4.0	23.0 ± 3.8	17.6 ± 7.4	15.6 ± 6.3	17.4 ± 4.6
HEP	2.8 ± 2.4	7.0 ± 2.5	3.2 ± 3.8	4.8 ± 4.7	4.8 ± 4.7
HOR	37.5 ± 19.5	8.8 ± 4.8	7.2 ± 4.6	8.4 ± 5.4	7.6 ± 4.2
IRI	5.4 ± 0.8	5.4 ± 0.8	6.4 ± 2.7	5.8 ± 1.4	6.4 ± 2.1
IRO	7.6 ± 2.7	16.6 ± 3.5	8.4 ± 4.7	10.6 ± 3.9	12.0 ± 6.5
MON	26.8 ± 12.3	12.6 ± 5.1	32.0 ± 10.2	36.6 ± 12.8	39.4 ± 13.5
MUS	26.8 ± 2.0	—	18.0 ± 2.4	15.0 ± 4.4	14.6 ± 4.0
OPT	107.4 ± 9.9	—	116.8 ± 14.1	117.2 ± 13.3	130.2 ± 14.3
PEN	134.4 ± 13.5	—	150.2 ± 15.4	154.0 ± 11.6	188.6 ± 15.3
SEG	42.8 ± 7.0	—	51.6 ± 8.9	56.8 ± 9.1	62.8 ± 9.1
VOT	4.0 ± 2.2	6.0 ± 2.2	4.0 ± 2.2	4.0 ± 2.2	4.0 ± 2.2
WIN	6.8 ± 2.6	8.6 ± 2.1	7.4 ± 1.6	7.4 ± 1.3	7.2 ± 1.5
ZOO	9.2 ± 2.4	11.8 ± 1.4	9.2 ± 2.4	9.2 ± 2.4	9.6 ± 2.7

Method	C4.5	C5.0	LDTu(1)	LDTu(2)	QUEST	$\sum$
C4.5	—	1	0	0	2	3
C5.0	0	—	1	1	1	1
LDTu(1)	2	1	—	0	1	4
LDTu(2)	2	1	0	—	0	3
QUEST	2	1	0	0	—	3
$\sum$	2	2	1	1	2	

In Sec. 6.1, we compare the two class separation heuristics; selection and exchange. The effect of ϵ, the proportion of variance explained, that determines the dimensionality in feature extraction through PCA, is analyzed in Sec. 6.2. Section 6.3 compares the effect of two different pruning techniques, prepruning and postpruning on LDTm. In Sec. 6.4, we compare finding the split point, w_0, analytically with doing an exhaustive search.

6.1. *Class separation*

The aim of this section is to find the effect of the class separation technique; selection or exchange, discussed in Sec. 3. Other variables such as impurity measure (entropy) or pruning technique (post-pruning) and the learning method (LDTm) are fixed. If there are only two classes available in a dataset, this is not included in the results because there is no need for class separation.

The fact that the goodness of partition of classes into two is measured by entropy and that the best split is then found using LDA may seem awkward but this is

Table 5. Comparison of learning times of different univariate techniques in terms of seconds.

Set	C4.5	C5.0	LDTu(1)	LDTu(2)	QUEST
BRE	1 ± 0	0 ± 0	1 ± 0	1 ± 0	1 ± 0
BUP	1 ± 0	0 ± 0	1 ± 0	1 ± 0	1 ± 0
CAR	11 ± 1	—	17 ± 2	19 ± 3	15 ± 1
CYL	5 ± 0	—	13 ± 2	14 ± 3	16 ± 2
DER	1 ± 0	0 ± 0	5 ± 1	5 ± 1	2 ± 0
ECO	1 ± 0	0 ± 0	2 ± 0	2 ± 0	1 ± 0
FLA	1 ± 0	0 ± 0	2 ± 1	2 ± 0	2 ± 0
GLA	1 ± 0	0 ± 0	1 ± 0	1 ± 0	1 ± 0
HEP	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
HOR	2 ± 0	0 ± 0	10 ± 2	10 ± 1	12 ± 1
IRI	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
IRO	1 ± 0	0 ± 0	2 ± 0	2 ± 1	2 ± 0
MON	1 ± 1	0 ± 0	2 ± 1	2 ± 1	2 ± 1
MUS	12 ± 2	—	216 ± 30	221 ± 42	263 ± 23
OPT	145 ± 8	—	926 ± 143	922 ± 202	243 ± 15
PEN	110 ± 8	—	292 ± 42	317 ± 52	179 ± 9
SEG	12 ± 1	—	46 ± 6	49 ± 14	22 ± 1
VOT	8 ± 2	0 ± 0	2 ± 0	2 ± 0	2 ± 0
WIN	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
ZOO	0 ± 0	0 ± 0	0 ± 0	1 ± 0	0 ± 0

Method	C4.5	C5.0	LDTu(1)	LDTu(2)	QUEST	$\sum$
C4.5	—	0	12	14	12	17
C5.0	14	—	14	13	14	14
LDTu(1)	1	0	—	3	0	3
LDTu(2)	1	0	0	—	0	1
QUEST	2	0	6	6	—	9
$\sum$	14	0	20	20	20	

because the two-class LDA splits two groups and it cannot be used to measure the separability of $K > 2$ classes. Entropy as we use measures the goodness of the separation of all K classes.

In none of the datasets, the selection method is more accurate than the exchange method in accuracy (Table 6). The exchange method is more accurate than selection method in three datasets out of 11. Two of these datasets, *Optdigits* and *Pendigits*, have ten classes and the other dataset, *Ecoli*, has eight classes. So it seems like the more classes there are, the better is the exchange method compared to the selection method, due to the large number of division candidates.

If the tree sizes are compared (Table 7), it is also seen that in two datasets, *Pendigits* and *Segment* (which has seven classes), out of 11, the exchange method is better while the selection method is never better.

Although the exchange method has larger time complexity, both methods performed similarly in terms of learning time (Table 8). This is due to the large tree size of the selection method, which is in turn because early on, the selection method cannot find splits as good as the exchange method, and generates larger trees.

Table 6. Accuracy comparison of class separation techniques. "$>$" and "$\gg$" implies 95% and 99% significant difference respectively.

Set	Selection	Exchange	Significance
CAR	81.35 ± 2.93	84.03 ± 4.58	
DER	92.24 ± 4.91	96.12 ± 0.98	
ECO	70.42 ± 7.58	78.57 ± 2.98	$2 > 1$
FLA	88.79 ± 2.51	88.85 ± 2.47	
GLA	51.21 ± 9.43	55.98 ± 7.96	
IRI	66.40 ± 30.69	95.07 ± 3.27	
OPT	60.56 ± 9.67	91.22 ± 0.64	$2 \gg 1$
PEN	77.38 ± 5.46	94.92 ± 0.78	$2 \gg 1$
SEG	76.77 ± 7.62	90.20 ± 1.04	
WIN	89.21 ± 11.02	96.63 ± 2.90	
ZOO	71.08 ± 14.89	75.42 ± 9.87	

Table 7. Tree size comparison of class separation techniques, in terms of the number of nodes in the induced tree.

Set	Selection	Exchange	Significance
CAR	10.20 ± 6.81	8.80 ± 4.16	
DER	11.80 ± 2.15	11.20 ± 0.63	
ECO	12.40 ± 4.90	10.20 ± 1.93	
FLA	1.40 ± 1.26	1.00 ± 0.00	
GLA	12.40 ± 6.87	12.20 ± 6.27	
IRI	6.40 ± 4.33	5.80 ± 1.40	
OPT	59.60 ± 14.42	43.20 ± 11.29	
PEN	120.60 ± 25.12	67.00 ± 14.08	$1 > 2$
SEG	49.00 ± 8.69	26.00 ± 8.12	$1 \gg 2$
WIN	4.60 ± 1.26	5.00 ± 0.00	
ZOO	7.40 ± 3.37	7.20 ± 1.99	

Table 8. Learning time comparison of class separation techniques, in terms of the number of seconds it takes to learn the tree.

Set	Selection	Exchange	Significance
CAR	20 ± 11	17 ± 9	
DER	7 ± 3	8 ± 4	
ECO	3 ± 1	3 ± 1	
FLA	1 ± 1	1 ± 1	
GLA	2 ± 2	4 ± 3	
IRI	0 ± 0	0 ± 0	
OPT	2011 ± 618	2594 ± 1076	
PEN	1421 ± 420	957 ± 334	
SEG	131 ± 36	109 ± 67	$1 \gg 2$
WIN	0 ± 0	0 ± 0	
ZOO	1 ± 0	1 ± 0	

Thus we can conclude that the exchange method generalizes better and learns smaller and more accurate trees on some applications but not on the majority.

6.2. *Effect of PCA*

In Sec. 2.2, we saw that PCA must be used to solve the singular covariance matrix problem. In this section, we compare the two percentage levels $\epsilon = 0.90$ and $\epsilon = 0.99$ and find out its effect. With $\epsilon = 0.99$, we only get rid of singularity; with $\epsilon = 0.90$, we also do dimensionality reduction. On *Breast, Bupa, Iris, Wine*, PCA is never needed. The results on the other sixteen datasets are shown in Table 9 for accuracy, in Table 10 for tree sizes, and in Table 11 for learning time.

Results show that increasing ϵ from 0.90 to 0.99 improves accuracy (on eight out of 16) but does not significantly increase the tree size or learning time. Thus we propose to use a large ϵ as possible, just enough to get rid of the eigenvectors with zero eigenvalues, but not any eigenvector contributing to variance.

6.3. *Effect of pruning*

In this section, we wish to find out which pruning technique best fits with our new proposed algorithm LDT. Therefore we tested LDT with two different pruning techniques; pre-pruning and post-pruning.

For post-pruning, the validation set is 20% of the whole training set. For pre-pruning we stop splitting further when the data on a node is smaller than 5% of the total number of elements in the dataset.

Simulation results are given in Table 12 for accuracy, Table 13 for tree size and Table 14 for learning time. The results show that there is not a significant

Table 9. The effect of ϵ in LDT on accuracies on datasets where PCA is used.

Set	$\epsilon = 0.90$	$\epsilon = 0.99$	Significance
CAR	70.03 ± 1.75	84.03 ± 4.58	$2 > 1$
CYL	65.43 ± 3.91	67.32 ± 3.78	$2 > 1$
DER	92.90 ± 1.65	96.12 ± 0.98	$2 > 1$
ECO	79.94 ± 3.64	78.57 ± 2.98	
FLA	88.60 ± 2.72	88.85 ± 2.47	
GLA	57.01 ± 7.19	55.98 ± 7.96	
HEP	82.83 ± 3.66	81.03 ± 6.65	
HOR	74.95 ± 3.37	79.62 ± 6.40	
IRO	85.75 ± 3.20	87.12 ± 4.16	
MON	73.38 ± 3.45	73.56 ± 3.58	
MUS	95.90 ± 1.27	98.64 ± 0.22	$2 \gg 1$
OPT	86.44 ± 0.89	91.22 ± 0.64	$2 \gg 1$
PEN	91.93 ± 0.65	94.92 ± 0.78	$2 \gg 1$
SEG	82.02 ± 2.38	90.20 ± 1.04	$2 > 1$
VOT	89.24 ± 1.84	94.44 ± 2.32	$2 \gg 1$
ZOO	75.44 ± 8.05	75.42 ± 9.87	

Table 10. The effect of ϵ in LDT on tree size (number of nodes) on datasets where PCA is used.

Set	$\epsilon = 0.90$	$\epsilon = 0.99$	Significance
CAR	1.20 ± 0.63	12.00 ± 2.54	$2 \gg 1$
CYL	6.60 ± 5.15	16.40 ± 8.95	
DER	12.60 ± 1.58	12.80 ± 1.48	
ECO	10.40 ± 3.13	20.00 ± 2.71	
FLA	1.40 ± 1.26	5.60 ± 3.13	
GLA	10.40 ± 4.90	26.20 ± 4.44	
HEP	2.40 ± 1.35	8.60 ± 3.10	
HOR	7.20 ± 5.20	16.80 ± 4.05	
IRO	4.20 ± 1.40	11.60 ± 2.67	
MON	6.60 ± 1.58	7.80 ± 4.13	
MUS	18.60 ± 4.60	19.20 ± 4.85	
OPT	49.40 ± 9.28	59.40 ± 2.07	$1 > 2$
PEN	79.20 ± 10.39	89.00 ± 6.25	
SEG	46.40 ± 11.28	39.80 ± 11.08	
VOT	4.00 ± 1.05	9.80 ± 2.53	
ZOO	6.40 ± 1.65	11.80 ± 1.93	

Table 11. The effect of ϵ in LDT on learning time (seconds) on datasets where PCA is used.

Set	$\epsilon = 0.90$	$\epsilon = 0.99$	Significance
CAR	2 ± 3	17 ± 9	
CYL	32 ± 13	33 ± 15	
DER	7 ± 2	8 ± 4	
ECO	2 ± 1	3 ± 1	
FLA	1 ± 0	1 ± 1	
GLA	2 ± 1	4 ± 3	
HEP	0 ± 0	1 ± 0	
HOR	23 ± 21	34 ± 23	
IRO	2 ± 1	4 ± 3	
MON	2 ± 1	2 ± 2	
MUS	1041 ± 132	3423 ± 1424	
OPT	1838 ± 257	2594 ± 1076	
PEN	720 ± 85	957 ± 334	
SEG	133 ± 33	109 ± 67	
VOT	5 ± 3	3 ± 1	
ZOO	1 ± 0	1 ± 0	

difference between pre-pruning and post-pruning in terms of accuracy and learning time, whereas post-pruning trees are significantly smaller than pre-pruning trees in seven datasets out of 20. So we can say that post-pruning has better performance than pre-pruning with LDT.

6.4. *Finding the split point*

In our proposed algorithm, after we find the weights $\boldsymbol{w}_m$ of the linear split at any node using LDA, we must also determine the split point w_0. In this case we have

Table 12. Comparison of accuracies in terms of percentage. The table shows the results for pre-pruning and post-pruning.

Set	Pre-pruning	Post-pruning	Significance
BRE	95.97 ± 0.69	95.88 ± 1.00	
BUP	62.49 ± 6.67	60.92 ± 4.01	
CAR	84.56 ± 5.98	84.03 ± 4.58	
CYL	69.65 ± 2.89	67.32 ± 3.78	
DER	96.50 ± 0.97	96.12 ± 0.98	
ECO	82.68 ± 3.13	78.57 ± 2.98	
FLA	87.68 ± 1.88	88.85 ± 2.47	
GLA	55.98 ± 5.89	55.98 ± 7.96	
HEP	84.78 ± 2.83	81.03 ± 6.65	
HOR	81.79 ± 1.74	79.62 ± 6.40	
IRI	93.73 ± 9.15	95.07 ± 3.27	
IRO	88.60 ± 3.26	87.12 ± 4.16	
MON	74.77 ± 1.62	73.56 ± 3.58	
MUS	98.47 ± 0.36	98.64 ± 0.22	
OPT	92.04 ± 1.16	91.22 ± 0.64	
PEN	92.87 ± 1.10	94.92 ± 0.78	$2 > 1$
SEG	90.48 ± 1.43	90.20 ± 1.04	
VOT	94.94 ± 2.42	94.44 ± 2.32	
WIN	96.18 ± 2.44	96.63 ± 2.90	
ZOO	81.79 ± 7.77	75.42 ± 9.87	

Table 13. Comparison of tree sizes in terms of number of nodes. The table shows the results for pre-pruning and post-pruning.

Set	Pre-pruning	Post-pruning	Significance
BRE	7.2 ± 0.6	4.4 ± 1.6	
BUP	8.8 ± 3.2	2.8 ± 2.0	$1 \gg 2$
CAR	12.6 ± 7.0	8.8 ± 4.2	
CYL	13.0 ± 4.7	6.6 ± 4.0	
DER	14.2 ± 1.0	11.2 ± 0.6	$1 > 2$
ECO	20.6 ± 5.5	10.2 ± 1.9	$1 > 2$
FLA	5.8 ± 2.5	1.0 ± 0.0	
GLA	21.0 ± 5.7	12.2 ± 6.3	$1 > 2$
HEP	7.2 ± 3.7	2.4 ± 1.3	
HOR	15.4 ± 4.5	3.4 ± 2.1	$1 \gg 2$
IRI	6.8 ± 3.0	5.8 ± 1.4	
IRO	11.0 ± 3.3	4.6 ± 1.6	$1 > 2$
MON	7.8 ± 4.1	6.6 ± 1.8	
MUS	25.2 ± 3.8	21.4 ± 5.1	
OPT	87.6 ± 7.5	43.2 ± 11.3	$1 \gg 2$
PEN	90.4 ± 19.9	67.0 ± 14.1	
SEG	39.6 ± 9.0	26.0 ± 8.1	
VOT	11.2 ± 4.2	4.2 ± 1.4	
WIN	6.6 ± 1.3	5.0 ± 0.0	
ZOO	12.2 ± 3.0	7.2 ± 2.0	

Table 14. Comparison of learning times in terms of seconds. The table shows the results for pre-pruning and post-pruning.

Set	Pre-pruning	Post-pruning	Significance
BRE	1 ± 0	2 ± 1	
BUP	1 ± 1	1 ± 1	
CAR	13 ± 8	17 ± 9	
CYL	24 ± 21	33 ± 15	
DER	8 ± 2	8 ± 4	
ECO	3 ± 1	3 ± 1	
FLA	1 ± 1	1 ± 1	
GLA	3 ± 1	4 ± 3	
HEP	1 ± 1	1 ± 0	
HOR	42 ± 36	34 ± 23	
IRI	0 ± 0	0 ± 0	
IRO	6 ± 4	4 ± 3	
MON	1 ± 1	2 ± 2	
MUS	1457 ± 389	3423 ± 1424	
OPT	1961 ± 718	2594 ± 1076	
PEN	737 ± 391	957 ± 334	
SEG	83 ± 33	109 ± 67	
VOT	5 ± 3	3 ± 1	
WIN	1 ± 0	0 ± 0	
ZOO	1 ± 0	1 ± 0	

two choices: either we use Eq. (7) and find w_0 analytically which assumes that the two groups are normal with equal variances; or, we can search through all possible split points, as in C4.5, to minimize impurity. In this section we compare these two alternatives. Keeping all other factors same, we generate trees analytically using Eq. (7) and by doing an exhaustive search, and compare the induced trees.

In terms of accuracy (Table 15), there is not any significant difference between two methods. Only on *Ecoli* dataset, exhaustive search is significantly better than the analytic method. This is expected as LDA gives the optimal separating discriminant if the two groups are normal distributed with equal covariance matrices.[5] The class parameters, i.e. mean and scatter, and the separating discriminant are estimated to maximize the likelihood and assuming a shared covariance matrix. Exhaustive search looks for a split to minimize entropy. Finding the vector to optimize one criterion and then finding the split point to optimize another is disturbing. Doing them both under the same assumption of normality makes more sense and should be preferred.

In terms of tree sizes (Table 16), analytical method creates smaller trees on two out of twenty datasets. Learning time results (Table 17) show that exhaustive search is always slower than analytical method but the difference is significant in six datasets out of twenty. These are not necessarily the large datasets. It seems that it is the matrix operations necessary to find the linear discriminant (which is a function of the input dimensionality, i.e. $\mathcal{O}(d^2)$, and not the training set size)

Table 15. Comparison of accuracies in terms of percentage. The table shows the results for analytical versus exhaustive methods of split point selection.

Set	Analytical	Exhaustive	Significance
BRE	95.88 ± 1.00	96.57 ± 0.74	
BUP	60.92 ± 4.01	66.03 ± 3.91	
CAR	84.03 ± 4.58	89.10 ± 1.34	
CYL	67.32 ± 3.78	66.39 ± 4.43	
DER	96.12 ± 0.98	96.28 ± 1.85	
ECO	78.57 ± 2.98	79.40 ± 3.25	$2 > 1$
FLA	88.85 ± 2.47	88.48 ± 2.32	
GLA	55.98 ± 7.96	55.98 ± 4.14	
HEP	81.03 ± 6.65	81.68 ± 4.80	
HOR	79.62 ± 6.40	79.89 ± 6.13	
IRI	95.07 ± 3.27	94.27 ± 2.18	
IRO	87.12 ± 4.16	86.03 ± 4.03	
MON	73.56 ± 3.58	78.10 ± 4.37	
MUS	98.64 ± 0.22	98.43 ± 0.45	
OPT	91.22 ± 0.64	91.36 ± 0.66	
PEN	94.92 ± 0.78	95.39 ± 0.47	
SEG	90.20 ± 1.04	92.74 ± 1.34	
VOT	94.44 ± 2.32	93.51 ± 2.71	
WIN	96.63 ± 2.90	95.51 ± 3.00	
ZOO	75.42 ± 9.87	76.25 ± 9.26	

Table 16. Comparison of tree sizes in terms of number of nodes. The table shows the results for analytical versus exhaustive methods of split point selection.

Set	Analytical	Exhaustive	Significance
BRE	4.4 ± 1.6	4.0 ± 2.2	
BUP	2.8 ± 2.0	14.0 ± 10.1	
CAR	8.8 ± 4.2	21.2 ± 5.8	$2 > 1$
CYL	6.6 ± 4.0	12.0 ± 9.0	
DER	11.2 ± 0.6	11.4 ± 0.8	
ECO	10.2 ± 1.9	11.4 ± 2.8	
FLA	1.0 ± 0.0	1.8 ± 2.5	
GLA	12.2 ± 6.3	11.4 ± 5.2	
HEP	2.4 ± 1.3	2.6 ± 2.1	
HOR	3.4 ± 2.1	5.4 ± 4.0	$2 > 1$
IRI	5.8 ± 1.4	5.0 ± 0.0	
IRO	4.6 ± 1.6	5.0 ± 2.1	
MON	6.6 ± 1.8	15.0 ± 8.8	
MUS	21.4 ± 5.1	24.2 ± 5.8	
OPT	43.2 ± 11.3	44.0 ± 5.6	
PEN	67.0 ± 14.1	74.8 ± 9.3	
SEG	26.0 ± 8.1	37.4 ± 7.5	
VOT	4.2 ± 1.4	5.0 ± 2.3	
WIN	5.0 ± 0.0	5.0 ± 0.0	
ZOO	7.2 ± 2.0	8.0 ± 2.4	

Table 17. Comparison of learning times in terms of seconds. The table shows the results for analytical versus exhaustive methods of split point selection.

Set	Analytical	Exhaustive	Significance
BRE	2 ± 1	3 ± 1	
BUP	1 ± 1	11 ± 3	$2 \gg 1$
CAR	17 ± 9	95 ± 21	$2 \gg 1$
CYL	33 ± 15	116 ± 23	$2 \gg 1$
DER	8 ± 4	9 ± 1	
ECO	3 ± 1	6 ± 1	$2 \gg 1$
FLA	1 ± 1	14 ± 5	$2 > 1$
GLA	4 ± 3	6 ± 1	
HEP	1 ± 0	1 ± 1	
HOR	34 ± 23	31 ± 8	
IRI	0 ± 0	0 ± 0	
IRO	4 ± 3	12 ± 2	$2 \gg 1$
MON	2 ± 2	16 ± 5	
MUS	3423 ± 1424	3348 ± 695	
OPT	2594 ± 1076	2453 ± 353	
PEN	957 ± 334	1226 ± 177	
SEG	109 ± 67	194 ± 11	
VOT	3 ± 1	5 ± 2	
WIN	0 ± 0	0 ± 0	
ZOO	1 ± 0	1 ± 0	

Table 18. Number of univariate and multivariate nodes, and percentage of the multivariate nodes to all nodes in the hybrid tree according to the specified rules.

Rule	Uni	Multi	Multi%
Level of the node ≤ 4	835	156	16%
Level of the node > 4	1211	42	3%
Number of data in the node ≥ 500	65	79	55%
Number of data in the node < 500	1981	119	6%
Number of classes in the node > 5	228	140	38%
Number of classes in the node ≤ 5	1818	58	3%
Ratio of training examples in the node $>$%20	620	130	17%
Ratio of training examples in the node $\leq$%20	1426	68	5%

that dominates the learning time and not the exhaustive search to find w_0 (which is $\mathcal{O}(N)$).

We thus conclude that exhaustive search method for split point selection does not offer any significant difference and sometimes it also increases learning time.

7. Univariate versus Multivariate Nodes

We wrote in the introduction that there is a dependency between the complexity of a node and the size of the tree: with complex nodes the tree may be quite small; with simple nodes one may grow large trees. However, a complex model with a larger number of parameters requires a larger training dataset and risks overfitting on a small amount of data. Each node type has a certain bias (axis-aligned versus arbitrary oblique split) and may be appropriate in a different situation.

The aim of this section is to investigate which type of node is better under which circumstances. To see this, at each node, we train and compare two possible nodes; univariate and linear multivariate, and use the 5 × 2 cv F test, to choose one. We then continue tree induction recursively in the same manner. Only if the test indicates that the multivariate node has better accuracy with at least 95% significance do we choose the multivariate node, otherwise we choose the univariate node due to its simplicity. We investigate this approach of a hybrid tree in more detail in Ref. 21 and call it the *omnivariate decision tree* which is composed of multivariate nonlinear, multivariate linear and univariate nodes. In this study, we only allow multivariate linear and univariate nodes, and our aim is to use it to understand which type of node is better when.

Running this algorithm ten times over all 30 datasets, there are a total of 2,244 internal decision nodes which may be thought of as 2,244 different problems on which the two node types are compared. Analyzing the nature of the problem and the decision made, i.e. the chosen node type, we try to find out the conditions leading to the two types of nodes.

Table 19 shows the proportion of univariate and multivariate nodes. We see that overall 90% of the nodes are univariate and only 10% of the nodes are multivariate

Table 19. Number of univariate and multivariate nodes in the hybrid tree.

Set	Uni	Multi
BRE	3.6 ± 1.1	0.0 ± 0.0
BUP	10.2 ± 3.5	0.2 ± 0.4
CAR	24.8 ± 4.2	0.5 ± 0.7
CYL	10.4 ± 5.7	0.3 ± 0.5
DER	4.6 ± 1.0	0.8 ± 0.4
ECO	5.2 ± 2.8	0.0 ± 0.0
FLA	0.3 ± 0.7	0.0 ± 0.0
GLA	6.1 ± 2.1	0.4 ± 0.7
HEP	0.9 ± 1.9	0.0 ± 0.0
HOR	2.9 ± 1.7	0.1 ± 0.3
IRI	2.2 ± 0.8	0.1 ± 0.3
IRO	4.1 ± 2.2	0.0 ± 0.0
MON	10.1 ± 2.8	0.2 ± 0.4
MUS	8.3 ± 1.6	1.5 ± 1.3
OPT	22.1 ± 6.0	8.4 ± 1.3
PEN	55.7 ± 5.9	5.5 ± 1.8
SEG	20.8 ± 3.3	2.4 ± 0.8
VOT	1.4 ± 1.2	0.1 ± 0.3
WIN	2.7 ± 0.7	0.0 ± 0.0
ZOO	2.9 ± 0.7	0.7 ± 0.7
$\sum$	199.3	21.2
%	90.4	9.6

nodes. On six datasets, multivariate nodes are never selected. This indicates that most problems can be solved univariate only.

We then want to find a connection between the attributes of the datasets and the type of nodes selected in hybrid decision tree. The attributes of datasets are: (1) Level of the node, (2) Number of features, (3) Number of classes, (4) Number of classes in that node, (5) Total number of training samples for that dataset, (6) Number of training examples in that node, and (7) The ratio of the training examples of that node to the total number of training examples of the whole dataset.

Taking these inputs and the node type as the class code, we run the C4.5 algorithm on this sample of 2,244 instances to check if we can find simple explanatory rules. Unfortunately, the generated decision tree is not so simple. It has 65 nodes. But analyzing the tree on a feature basis, we find simple rules using a single feature that give us information. According to the decision tree, we see that the probability of using a multivariate node increases (over its prior probability of 10%) when (Table 18)

- The number of training samples in that node is over 500.
- The number of classes in that node is larger than five.
- The ratio of the training examples is larger than 20%.
- The level of the node is smaller than or equal to four.

As shown in Fig. 3, we see that multivariate nodes, when they appear, they do so early on in the tree closer to the root where there is enough data. This is expected because as we go down the tree, we have easier problems (having less classes) in effectively smaller-dimensional subspaces and at the same time, we have smaller training data and multivariate model overfits and simple, e.g. univariate, splits suffice and generalize better. It seems like it is not the training set percentage *per se* that affects the node type but rather the number of data instances. It may also be said that when there are more classes, separating them is a more complicated task and the multivariate split is preferred over the univariate split. Of course, whether a univariate or a multivariate node is more appropriate depends actually on the complexity of the unknown discriminant we are approximating and this complexity is only partially reflected by the number of classes or the number of training examples. The moral we would like to convey is that a univariate model may be sufficient for most nodes during tree construction but a judicious use of the multivariate node may make the tree smaller and more accurate.

8. Overall Comparison of Decision Tree Methods

The seven decision tree construction methods both univariate and multivariate, are compared. These are: univariate C4.5, multivariate linear CART, oblique classifier OC1, neural trees with single layer perceptron nodes ID-LP, linear machine decision trees LMDT, and the multivariate versions of QUEST and our algorithm LDT. We

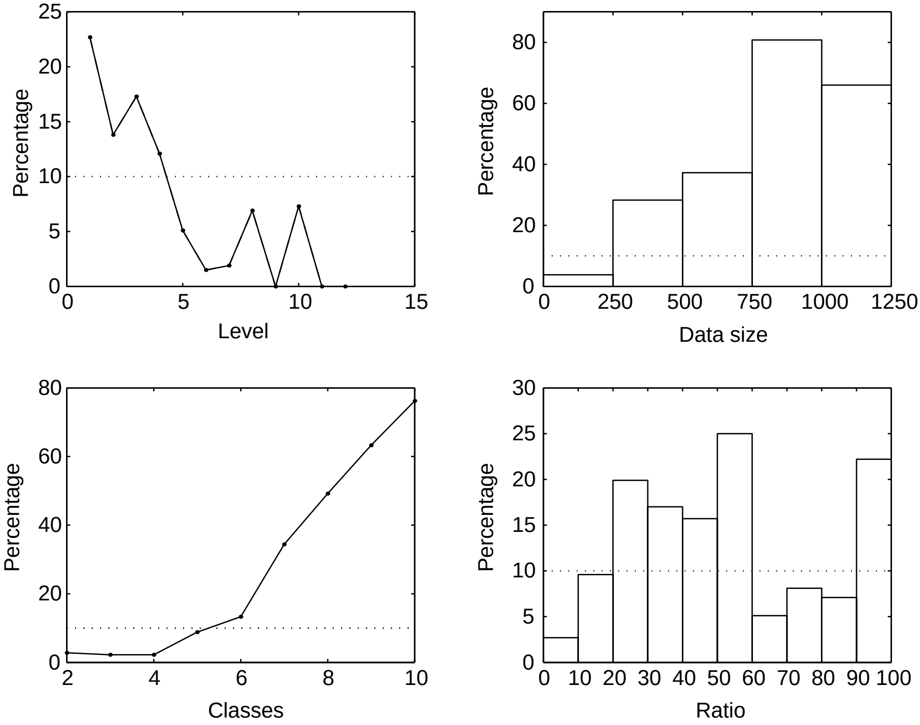

Fig. 3. The percentage of multivariate nodes as a function of level in the tree, data size in the node, number of classes in the node, and the percentage of training set reaching to the node. The default probability of 10% is shown dashed. We see that the more complex multivariate node is used early in the tree, when there is enough data, and/or if the number of classes is large.

use exchange method for class separation in training ID-LP and LDTm. We always use post-pruning. In LDT, when PCA is used, we take $\epsilon = 0.99$.

In Table 20, we see that the accuracies of C4.5, LDT, QUEST, ID-LP and LMDT are nearly equal and they seem to outperform CART and OC1. LDT, QUEST and ID-LP construct K-way classification using binary splits and generalize as well as the K-way splitting LMDT.

In Table 21, we see that C4.5 constructs the smallest trees in terms of the number of parameters. After C4.5, the smallest trees are constructed by LDT and ID-LP. LMDT performs better than OC1 and QUEST, which perform better than CART.

Table 22 shows the comparison of learning time of the methods. CART and OC1 take the most time in learning, whereas C4.5 and LMDT have the smallest learning time. Remember again that this is due to the K-way splitting nodes of LMDT. LDT and ID-LP with binary nodes are reasonably fast. All multivariate methods except LDT and QUEST in training a decision node are iterative, i.e. they optimize iteratively an error function, and have no guarantee of converging to the optimum

Table 20. Comparison of accuracies of different decision tree methods in terms of percentage.

Set	C4.5	CART	ID-LP	LMDT	OC1	QUEST	LDT
BRE	94.7 ± 1.8	94.9 ± 1.4	96.3 ± 0.8	96.0 ± 1.0	95.9 ± 0.8	96.2 ± 0.7	95.9 ± 1.0
BUP	62.8 ± 3.4	61.7 ± 3.4	64.2 ± 3.8	60.6 ± 4.1	63.1 ± 2.8	62.4 ± 4.8	60.9 ± 4.0
CAR	86.1 ± 1.6	83.8 ± 2.0	92.7 ± 1.8	85.5 ± 1.7	87.2 ± 2.0	85.5 ± 2.0	84.0 ± 4.6
CYL	67.3 ± 4.7	59.5 ± 4.1	69.2 ± 2.8	68.9 ± 2.6	64.7 ± 3.7	68.3 ± 2.4	67.3 ± 3.8
DER	92.5 ± 2.4	80.9 ± 4.6	97.2 ± 1.0	95.6 ± 1.7	78.0 ± 4.8	96.3 ± 1.3	96.1 ± 1.0
ECO	78.3 ± 4.0	74.6 ± 3.8	80.0 ± 4.1	78.3 ± 5.6	76.5 ± 3.2	78.6 ± 2.9	78.6 ± 3.0
FLA	88.4 ± 2.6	81.6 ± 3.6	88.1 ± 2.7	88.9 ± 2.5	87.7 ± 2.7	88.6 ± 2.4	88.9 ± 2.5
GLA	60.1 ± 5.5	53.9 ± 4.2	57.4 ± 7.5	54.5 ± 3.6	56.5 ± 6.4	57.3 ± 5.9	56.0 ± 8.0
HEP	79.0 ± 4.5	79.0 ± 4.0	81.6 ± 4.4	79.4 ± 4.1	80.4 ± 3.3	83.7 ± 4.4	81.0 ± 6.7
HOR	73.8 ± 6.7	77.0 ± 3.0	80.0 ± 6.6	79.7 ± 6.5	81.0 ± 6.9	80.8 ± 6.5	79.6 ± 6.4
IRI	92.9 ± 3.3	89.3 ± 4.4	95.5 ± 2.4	85.7 ± 9.2	92.8 ± 5.0	95.5 ± 1.8	95.1 ± 3.3
IRO	86.2 ± 3.7	86.8 ± 4.0	87.9 ± 2.8	85.2 ± 3.0	85.5 ± 3.7	85.9 ± 2.9	87.1 ± 4.2
MON	89.8 ± 7.7	91.2 ± 6.9	72.3 ± 5.7	71.1 ± 6.3	80.4 ± 3.2	77.5 ± 6.1	73.6 ± 3.6
MUS	99.9 ± 0.1	93.5 ± 1.8	100.0 ± 0.0	100.0 ± 0.1	99.4 ± 0.3	98.5 ± 0.2	98.6 ± 0.2
OPT	84.8 ± 0.8	81.4 ± 2.1	92.4 ± 0.8	93.1 ± 0.9	72.3 ± 2.7	91.3 ± 0.6	91.2 ± 0.6
PEN	92.5 ± 0.6	87.1 ± 2.9	94.9 ± 2.8	95.9 ± 0.8	89.0 ± 0.4	96.0 ± 0.3	94.9 ± 0.8
SEG	92.0 ± 0.9	88.1 ± 1.7	87.6 ± 10.7	90.2 ± 0.9	87.0 ± 1.7	89.1 ± 1.3	90.2 ± 1.0
VOT	95.6 ± 0.7	90.3 ± 3.2	95.5 ± 0.9	94.8 ± 0.8	93.4 ± 1.6	94.4 ± 2.2	94.4 ± 2.3
WIN	86.6 ± 1.9	87.3 ± 4.4	96.1 ± 1.9	94.0 ± 2.4	85.4 ± 4.0	95.7 ± 2.6	96.6 ± 2.9
ZOO	83.0 ± 7.4	70.0 ± 9.7	79.2 ± 9.6	83.8 ± 7.3	72.0 ± 10.9	74.5 ± 7.4	75.4 ± 9.9

Method	C4.5	CART	ID-LP	LMDT	OC1	QUEST	LDT	$\sum$
C4.5	—	5	0	2	5	2	2	7
CART	0	—	1	1	0	0	1	2
ID-LP	5	7	—	1	5	2	1	7
LMDT	3	6	0	—	4	1	1	7
OC1	0	2	0	1	—	0	0	2
QUEST	2	7	0	0	3	—	0	7
LDT	3	6	0	1	4	0	—	6
$\sum$	4	10	1	3	7	3	3	

point and there is thus a problem of when to stop training. LDT and QUEST solve for the discriminant analytically and have no such problem.

LDT has a learning time complexity of $\mathcal{O}(Nd^2)$ for constructing the covariance matrix (which is the costliest part), $\mathcal{O}(d^4)$ for finding the eigenvectors of the covariance matrix, $\mathcal{O}(d^2k)$ for converting the covariance matrix to the new k-dimensional covariance matrix and $\mathcal{O}(k^3)$ for taking the inverse of this new covariance matrix. ID-LP has a learning time of $\mathcal{O}(Nde)$, where e stands for the number of epochs to train the neural network. So we can say that comparing $\mathcal{O}(Nd^2)$ of LDT and $\mathcal{O}(Nde)$ of ID-LP, LDT is faster than ID-LP if e, the number of epochs to train the neural network, is larger than d, the dimensionality of the problem.

Although the accuracies of LDT and QUEST are comparable, we see in Table 21 that in six datasets, LDT generates smaller trees than QUEST whereas QUEST trees are never smaller than LDT trees. We see in Table 22 that on 11 datasets, LDT learns faster than QUEST whereas QUEST never learns faster. This is different

Table 21. Comparison of tree complexities of different decision tree methods in terms of the number of parameters.

Set	C4.5	CART	ID-LP	LMDT	OC1	QUEST	LDT
BRE	19 ± 8	56 ± 15	13 ± 4	27 ± 13	32 ± 18	22 ± 17	20 ± 9
BUP	26 ± 19	170 ± 15	13 ± 12	36 ± 46	43 ± 30	83 ± 23	8 ± 8
CAR	134 ± 16	323 ± 39	167 ± 76	746 ± 222	286 ± 75	371 ± 97	91 ± 48
CYL	31 ± 13	1563 ± 174	299 ± 141	455 ± 345	306 ± 227	356 ± 212	200 ± 0
DER	18 ± 2	487 ± 85	181 ± 0	204 ± 0	278 ± 98	188 ± 15	185 ± 11
ECO	21 ± 7	150 ± 23	40 ± 10	86 ± 30	46 ± 15	68 ± 17	42 ± 9
FLA	7 ± 9	411 ± 78	21 ± 33	0 ± 0	11 ± 18	24 ± 48	1 ± 0
GLA	21 ± 6	229 ± 23	43 ± 17	151 ± 49	55 ± 27	96 ± 34	63 ± 35
HEP	4 ± 4	138 ± 36	12 ± 11	27 ± 36	30 ± 27	22 ± 17	16 ± 14
HOR	56 ± 29	1338 ± 257	130 ± 67	310 ± 187	179 ± 146	169 ± 124	120 ± 102
IRI	8 ± 1	29 ± 7	13 ± 0	18 ± 7	14 ± 2	14 ± 2	15 ± 4
IRO	11 ± 4	278 ± 68	62 ± 24	116 ± 33	138 ± 58	102 ± 76	66 ± 28
MON	40 ± 19	68 ± 41	31 ± 21	56 ± 20	72 ± 22	67 ± 16	23 ± 7
MUS	40 ± 3	1429 ± 222	69 ± 0	132 ± 0	436 ± 97	905 ± 309	695 ± 175
OPT	161 ± 15	2298 ± 131	852 ± 144	2938 ± 483	2595 ± 216	2707 ± 277	1394 ± 373
PEN	201 ± 20	692 ± 91	411 ± 92	1488 ± 246	1463 ± 292	1079 ± 94	595 ± 127
SEG	64 ± 10	443 ± 90	197 ± 69	788 ± 161	539 ± 146	621 ± 141	251 ± 81
VOT	6 ± 3	276 ± 90	38 ± 11	83 ± 43	55 ± 29	59 ± 23	55 ± 24
WIN	10 ± 4	64 ± 17	31 ± 0	39 ± 0	36 ± 10	31 ± 0	31 ± 0
ZOO	13 ± 4	219 ± 45	62 ± 21	109 ± 7	57 ± 23	75 ± 26	57 ± 18

Method	C4.5	CART	ID-LP	LMDT	OC1	QUEST	LDT	$\sum$
C4.5	—	19	9	11	10	10	10	19
CART	0	—	0	1	1	1	0	1
ID-LP	0	17	—	5	5	6	2	18
LMDT	0	9	0	—	1	1	1	9
OC1	0	12	0	2	—	1	0	12
QUEST	0	11	0	0	0	—	0	11
LDT	0	15	0	5	5	6	—	18
$\sum$	0	19	9	11	12	11	10	

than what we saw in univariate trees. In univariate trees QUEST is better than LDTu, since in univariate trees there is no matrix calculation, so there is not much difference between univariate LDA and QDA in finding the split point. In this case grouping classes is more important. Since QUEST uses unsupervised 2-means, it is faster than LDT.

In multivariate trees, LDT is faster than QUEST. Since QUEST assumes unequal variances, there are two covariance matrix calculations, which takes longer time. In LDT, equal variances are assumed, and so only one covariance matrix calculation is needed.

Also if there is singularity in the covariance matrix or matrices, after applying PCA with new dimensions, we need to calculate again two new covariance matrices for QUEST, but one new covariance matrix for LDT.

Table 22. Comparison of learning times of different decision tree methods in terms of seconds.

Set	C4.5	CART	ID-LP	LMDT	OC1	QUEST	LDT
BRE	1 ± 0	107 ± 17	1 ± 0	0 ± 0	138 ± 24	6 ± 2	2 ± 1
BUP	1 ± 0	252 ± 23	1 ± 0	0 ± 0	159 ± 13	14 ± 2	1 ± 1
CAR	11 ± 1	1178 ± 148	17 ± 6	20 ± 5	1835 ± 119	360 ± 78	17 ± 9
CYL	5 ± 0	4589 ± 343	9 ± 2	6 ± 1	2297 ± 362	153 ± 27	33 ± 15
DER	1 ± 0	858 ± 170	7 ± 1	1 ± 0	561 ± 76	12 ± 2	8 ± 4
ECO	1 ± 0	221 ± 25	4 ± 1	1 ± 0	143 ± 21	13 ± 3	3 ± 1
FLA	1 ± 0	1032 ± 203	1 ± 1	1 ± 1	342 ± 84	17 ± 5	1 ± 1
GLA	1 ± 0	320 ± 25	2 ± 1	1 ± 0	138 ± 16	12 ± 3	4 ± 3
HEP	0 ± 0	209 ± 47	0 ± 0	0 ± 0	85 ± 13	1 ± 1	1 ± 0
HOR	2 ± 0	3481 ± 1101	4 ± 1	3 ± 0	1890 ± 271	58 ± 18	34 ± 23
IRI	0 ± 0	31 ± 11	0 ± 0	0 ± 0	21 ± 5	0 ± 0	0 ± 0
IRO	1 ± 0	544 ± 94	1 ± 0	1 ± 0	337 ± 47	21 ± 4	4 ± 3
MON	1 ± 1	126 ± 61	1 ± 0	1 ± 0	155 ± 14	11 ± 3	2 ± 2
MUS	12 ± 2	33613 ± 2942	136 ± 72	19 ± 2	70100 ± 8054	5669 ± 984	3423 ± 1424
OPT	145 ± 8	9148 ± 713	662 ± 153	223 ± 23	42076 ± 2279	5935 ± 615	2594 ± 1076
PEN	110 ± 8	3311 ± 350	823 ± 167	187 ± 21	22690 ± 808	2350 ± 198	957 ± 334
SEG	12 ± 1	1212 ± 170	85 ± 15	30 ± 7	3350 ± 726	423 ± 73	109 ± 67
VOT	8 ± 2	805 ± 167	2 ± 1	1 ± 0	233 ± 46	5 ± 2	3 ± 1
WIN	0 ± 0	84 ± 26	1 ± 0	0 ± 0	49 ± 13	1 ± 0	0 ± 0
ZOO	0 ± 0	453 ± 61	3 ± 0	0 ± 0	47 ± 8	1 ± 0	1 ± 0

Method	C4.5	CART	ID-LP	LMDT	OC1	QUEST	LDT	$\sum$
C4.5	—	20	9	6	20	15	6	20
CART	0	—	0	0	6	0	0	6
ID-LP	1	20	—	0	20	14	1	20
LMDT	6	20	10	—	20	15	5	20
OC1	0	7	0	0	—	0	0	7
QUEST	1	20	2	0	20	—	0	20
LDT	0	20	2	0	20	11	—	20
$\sum$	6	20	10	6	20	15	6	

9. Conclusions

In this paper, we propose a new decision tree induction method, LDT, based on Fisher's Linear Discriminant Analysis. Our algorithm has both univariate and multivariate versions and our experimental results comparing our algorithm with other univariate and multivariate methods indicate that the proposed method is accurate, learns fast, and generates small trees.

Our results indicate that in the majority of cases, a univariate method suffices. In the case of the univariate version of our method, LDTu, when there are numeric features, we propose to use the univariate version of our algorithm as it calculates the split position analytically and thus is faster than C4.5. With discrete features, converting them to a set of 0/1 variables and treating them as numeric variables keeps the tree binary and works as well as making a N-way split as done in C4.5.

Among the multivariate algorithms, the multivariate version of our proposed method, LDTm, stands out as a good method as it is quite accurate, generates small trees, and learns fast. It uses binary decision nodes which help interpretability. It has a sound statistical basis and under certain assumptions, finds the optimal discriminant. There is no error function to be minimized iteratively and thus has no problems associated with convergence. The splitting of $K > 2$ classes into two is done using a heuristic that employs a supervised measure that takes class information into account and, for example, is advantageous over the unsupervised method 2-means clustering that QUEST uses.

We advocate that though a univariate model may be sufficient for most cases, a judicious use of the multivariate node may make the tree smaller and more accurate. In our experiments, we see that using a multivariate tree or having some nodes of a tree multivariate can be justified if the node is early in the tree, if there are enough training instances, or if the number of classes in that node is large.

Acknowledgments

An earlier version of this work was presented at the 17th International Conference on Machine Learning. This work has been supported by the Turkish Academy of Sciences, in the framework of the Young Scientist Award Program (EA-TÜBA-GEBIP/2001-1-1) and Boğaziçi University Scientific Research Project 02A104D. We would like to thank the editor and anonymous reviewers for their constructive comments.

References

1. E. Alpaydın, Combined 5×2 cv F test for comparing supervised classification learning algorithms, *Neural Comput.* **11** (1999) 1975–1982.
2. L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees* (Wadsworth International Group, Belmont, CA, 1984).
3. L. A. Breslow and D. W. Aha, *Simplifying Decision Trees: A Survey*, NCARAI Technical Report No. AIC-96-014, Navy Center for Applied Research in AI, Naval Research Laboratory, Washington DC, USA, 1997.
4. C. E. Brodley and P. E. Utgoff, Multivariate decision trees, *Mach. Learn.* **19** (1995) 45–77.
5. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis* (Wiley, NY, 1973).
6. J. H. Friedman, A recursive partitioning decision rule for non-parametric classification, *IEEE Trans. Comput.* **26** (1977) 404–408.
7. J. Gama, Discriminant trees, *16th Int. Conf. Machine Learning* (Morgan Kaufmann, 1999), pp. 134–142.
8. H. Guo and S. B. Gelfand, Classification trees with neural network feature extraction, *IEEE Trans. Neural Networks* **3** (1992) 923–933.
9. ICELL Tree Induction Software. http://haydut.cmpe.boun.edu.tr/icell.html.
10. H. Kim and W. Loh, Classification trees with unbiased multiway splits, *J. Amer Stat. Assoc.* **96** (2001) 589–604.
11. W. Loh and Y. S. Shih, Split selection methods for classification trees, *Stat. Sin.* **7** (1997) 815–840.

12. W. Loh and N. Vanichsetakul, Tree-structured classification via generalized discriminant analysis, *J. Amer. Stat. Assoc.* **83** (1988) 715–725.
13. C. J. Merz and P. M. Murphy, *UCI Repository of Machine Learning Databases*, http://www.ics.uci.edu/˜ mlearn/MLRepository.html, 1998.
14. S. K. Murthy, S. Kasif and S. Salzberg, A system for induction of oblique decision trees, *J. Artif. Intell. Res.* **2** (1994) 1–32.
15. S. K. Murthy, Automatic construction of decision trees from data: a multi-disciplinary survey, *Data Min. Knowl. Disc.* **4** (1998) 345–389.
16. J. R. Quinlan, Induction of decision trees, *Mach. Learn.* **1** (1986) 81–106.
17. J. R. Quinlan, *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Meteo, CA, 1993).
18. J. R. Quinlan and R. L. Rivest, Inferring decision trees using the minimum description length principle, *Inform. Comput.* **80** (1989) 227–248.
19. A. C. Rencher, *Methods of Multivariate Analysis* (Wiley, NY, 1995).
20. RuleQuest Research, *Data Mining Tools*, http://www.rulequest.com.
21. O. T. Yıldız and E. Alpaydın, Omnivariate decision trees, *IEEE Trans. Neural Networks* **12** (2001) 1539–1547.

Olcay Taner Yıldız received his B.S, M.S and Ph.D. degrees in computer science from Boğaziçi University, Istanbul, Turkey in 1997, 2000, 2005 respectively. He is currently doing postdoctoral study at the University of Minnesota.

His research interests include model selection, decision trees, neural networks, and robotics.

Ethem Alpaydın received his Ph.D. degree in computer science from Ecole Polytechnique Fédérale de Lausanne, Switzerland in 1990 and did postdoctoral work at the International Computer Science Institute (ICSI), Berkeley in 1991. Since then he has been teaching in the Department of Computer Engineering at Boğaziçi University, Istanbul, where he is currently professor. He had visiting appointments at MIT in 1994, ICSI in 1997 (as a Fulbright scholar) and IDIAP, Switzerland in 1998. He received the young scientist award from the Turkish Academy of Sciences in 2001 and the scientific encouragement award from the Turkish Scientific and Technical Research Council in 2002. His book, *Introduction to Machine Learning*, has recently been published by The MIT Press.