

# Multivariate Decision Trees

CARLA E. BRODLEY

brodley@ecn.purdue.edu

*School of Electrical Engineering, Purdue University, West Lafayette, Indiana 47907*

PAUL E. UTGOFF

utgoff@cs.umass.edu

*Department of Computer Science, University of Massachusetts, Amherst, Massachusetts 01003*

**Editor:** Dennis Kibler

**Abstract.** Unlike a univariate decision tree, a multivariate decision tree is not restricted to splits of the instance space that are orthogonal to the features' axes. This article addresses several issues for constructing multivariate decision trees: representing a multivariate test, including symbolic and numeric features, learning the coefficients of a multivariate test, selecting the features to include in a test, and pruning of multivariate decision trees. We present several new methods for forming multivariate decision trees and compare them with several well-known methods. We compare the different methods across a variety of learning tasks, in order to assess each method's ability to find concise, accurate decision trees. The results demonstrate that some multivariate methods are in general more effective than others (in the context of our experimental assumptions). In addition, the experiments confirm that allowing multivariate tests generally improves the accuracy of the resulting decision tree over a univariate tree.

**Keywords:** decision trees, multivariate tests, linear discriminant functions, inductive learning

## 1 Introduction

One dimension by which decision trees can be characterized is whether they test more than one feature at a node. Decision trees that are limited to testing a single feature at a node are potentially much larger than trees that allow testing of multiple features at a node. This limitation reduces the ability to express concepts succinctly, which renders many classes of concepts difficult or impossible to express. This representational limitation manifests itself in two forms: subtrees are replicated in the decision tree (Pagallo & Haussler, 1990) and features are tested more than once along a path in the decision tree. For the replicated subtree problem, forming Boolean combinations of the features has been shown to improve the accuracy, reduce the number of instances required for training, and reduce the size of the decision tree (Pagallo, 1990; Matheus, 1990).

Repeated testing of features along a path in the tree occurs when a subset of the features are related numerically. Consider the two-dimensional instance space shown in Fig. 1 and the corresponding univariate decision tree, which approximates the hyperplane boundary,  $x + y \leq 8$ , with a series of orthogonal splits. In the figure, the dotted line represents the hyperplane boundary and the solid line represents the boundary defined by the univariate decision tree. This example illustrates the well known problem that a univariate test using feature  $F_i$  can only split a space with a boundary that is orthogonal to  $F_i$ 's axis (Breiman, Friedman, Olshen & Stone, 1984). This limits the space of regions in the instance space that can be represented succinctly, and can result in a large tree and poor generalization to the unobserved instances.

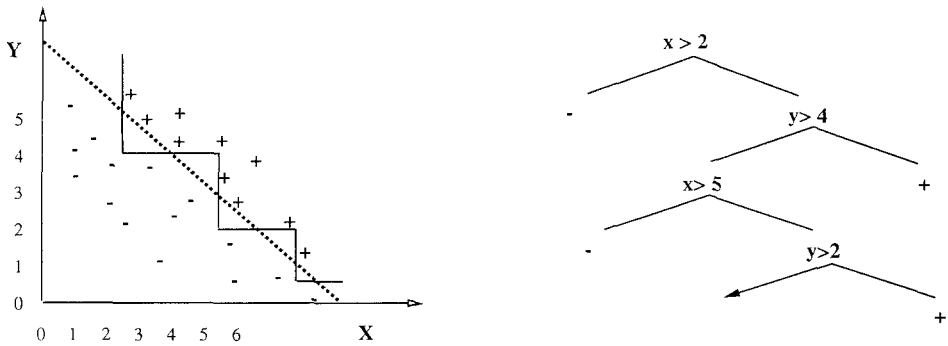


Figure 1. An example instance space; “+”: positive instance, “-”: negative instance and the corresponding univariate decision tree.

Multivariate decision trees alleviate the replication problems of univariate decision trees. In a multivariate decision tree, each test can be based on one or more of the input features. For example, the multivariate decision tree for the data set shown in Fig. 1 consists of one test node and two leaves. The test node is the multivariate test  $x + y \leq 8$ . Instances for which  $x + y$  is less than or equal to 8 are classified as negative; otherwise they are classified as positive.

In this article we describe and evaluate a variety of multivariate tree construction methods. The first part is devoted to discussing the issues that must be considered when constructing multivariate decision trees. We introduce several new methods for multivariate tree construction and review several well-known methods. The second part reports and discusses experimental results of comparisons among the various methods. The results indicate that some multivariate decision tree methods are generally more effective than others across a large number of domains. In addition, our experiments confirm that allowing multivariate tests generally improves the accuracy of the resulting decision tree over univariate decision trees. We focus on multivariate tests that are linear combinations of the initial features that describe the instances. The issues and methods we describe are not restricted to first-order linear combinations; they can be used to combine linearly higher-order features or features that are combinations of the initial features (see Sutton and Matheus (1991) for an approach to constructing more complex features).

Specifically, in Section 2 we first review the standard decision tree methodology and then outline the following issues one must consider in order to construct a multivariate decision tree: including symbolic features in multivariate tests, handling missing values, representing a linear combination test, learning the coefficients of a linear combination test, selecting the features to include in a test, and avoiding overfitting the training data. In Section 3 we describe four algorithms for finding the coefficients of a linear combination test. In Section 4 we describe two new and three well-known approaches to selecting which features to include in a multivariate test, and in Section 5 we present a new method for avoiding overfitting when multivariate tests are used in decision trees. In Section 6 we present the results of empirical experiments of the described methods for several different learning tasks and discuss the strengths and weaknesses of each presented method. Finally, in Section 7 we summarize our conclusions resulting from this evaluation and we outline an avenue for future work in multivariate decision tree research.

## 2 Issues in multivariate tree construction

There are two important advantages of decision-tree classifiers that one does not want to lose by permitting multivariate splits. Firstly, the tests in a decision tree are performed sequentially by following the branches of the tree. Thus, only those features that are required to reach a decision need to be evaluated. On the assumption that there is some cost in obtaining the value of a feature, it is desirable to test only those features that are needed. Secondly, a decision tree provides a clear statement of a sequential decision procedure for determining the classification of an instance. A small tree with simple tests is most appealing because a human can understand it. There is a tradeoff to consider in allowing multivariate tests: using only univariate tests may result in large trees that are difficult to understand, whereas the addition of multivariate tests may result in trees with fewer nodes, but the test nodes may be more difficult to understand.

In this section we describe issues of importance for multivariate decision tree construction. Sections 2.1 through 2.6 discuss issues specific to forming multivariate tests and general decision tree issues in the context of multivariate tests. Many of the issues for constructing multivariate decision trees are the same as for univariate decision trees. For both multivariate and univariate decision tree algorithms a tree is built from labeled examples. If the domain contains noisy instances, then some technique for pruning back the tree is used to avoid overfitting the training data.

Given a set of training instances, each described by  $n$  features and labeled with a class name, a top-down decision tree algorithm chooses the best test to partition the instances using some partition-merit criterion. The chosen test is then used to partition the training instances and a branch for each outcome of the test is created. The algorithm is applied recursively to each resulting partition. If the instances in a partition are from a single class, then a leaf node is created and assigned the class label of the single class. During decision tree construction, at each node, one wants to select the *test* that best divides the instances into their classes. The difference between univariate and multivariate trees is that in a univariate decision tree a test is based on just one feature, whereas in a multivariate decision tree a test is based on one or more features. There are many different partition-merit criteria that can be used to judge the “goodness of a split”; the most common appear in the form of an entropy or impurity measure. Breiman, et al. (1984), Quinlan (1986), Mingers (1989), Safavian and Landgrebe (1991), Buntine and Niblett (1992), and Fayyad and Irani (1992b) discuss and compare different partition-merit criteria.

In addition, for both univariate and multivariate decision trees, one wants to avoid overfitting the decision tree to the training data in domains that contain noisy instances. There are two types of noise: the class label is incorrect or some number of the attribute values are incorrect. Noise can be caused by many different factors, including faulty measurements, ill-defined thresholds and subjective interpretation (Quinlan, 1986b). Overfitting occurs when the training data contain noisy instances and the decision tree algorithm induces a classifier that classifies all instances in the training set correctly. Such a tree will usually perform poorly for previously unseen instances. To avoid overfitting, the tree must be pruned back to reduce the estimated classification error.

## 2.1 *Symbolic and numeric features*

One desires that a decision tree algorithm be able to handle both unordered (symbolic) and ordered (numeric) features. Univariate decision tree algorithms require that each test have a discrete number of outcomes. To meet this requirement, each ordered feature  $x_i$  is mapped to a set of unordered features by finding a set of Boolean tests of the form  $x_i > a$ , where  $a$  is in the observed range of  $x_i$ . See Fayyad and Irani (1992a) for a discussion of the issues involved in mapping ordered features to unordered features.

When constructing linear combination tests, the problem is reversed: how can one include unordered features in a linear combination test? One solution, used in CART (Breiman, Friedman, Olshen & Stone, 1984) is to form a linear combination using only the ordered features. An alternative solution is to map each multi-valued unordered feature to  $m$  numeric features, one for each observed value of the feature (Hampson & Volper, 1986), (Utgoff & Brodley, 1990).

In order to map an unordered feature to a set of numeric features, one needs to be careful not to impose an order on the values of the unordered feature. For a two-valued feature, one can simply assign 1 to one value and  $-1$  to the other. If the feature has more than two observed values, then each feature-value pair is first mapped to a propositional feature, which is TRUE if and only if the feature has the particular value in the instance (Hampson & Volper, 1986). The two-valued propositional feature is then mapped to a numeric feature, where a value of TRUE is mapped to 1 and a value of FALSE is mapped to  $-1$ . This mapping avoids imposing any order on the unordered values of the feature. With this encoding, one can create linear combinations of both ordered and unordered features.

## 2.2 *Filling in missing values*

For some instances, not all feature values may be available. In such cases, one would like to fill in the missing values. Quinlan (1989) describes a variety of approaches for handling missing values of unordered features. These include ignoring any instance with a missing value, filling in the most likely value, and combining the results of classification using each possible value according to the probability of that value. For linear combination tests, ignoring instances with missing values may reduce the number of available instances significantly as there may be few instances with all values present.

Another approach for handling a missing value is to estimate it using the sample mean, which is an unbiased estimator of the expected value. At *each node* in the tree, each encoded symbolic and numeric feature is normalized by mapping it to standard normal form, i.e., zero mean and unit standard deviation (Sutton, 1988). After normalization, missing values can be filled in with the sample mean, which is zero. In a linear combination test this has the effect of removing the feature's influence from the classification, because a feature with a value of zero does not contribute to the value of the linear combination. The normalization information is retained at each node in the tree. During classification, this information is used to normalize and fill in missing values of the instance to be classified.

## 2.3 *Numerical representation of a multivariate test*

For two-class learning tasks, a multivariate test can be represented by a linear threshold unit. For multiclass tasks, two possible representations are a linear threshold unit or a linear

machine (Nilsson, 1965; Duda & Hart, 1973). A linear threshold unit (LTU) is a binary test of the form  $W^T Y \geq 0$ , where  $Y$  is an instance description (a pattern vector) consisting of a constant 1 and the  $n$  features that describe the instance.  $W$  is a vector of  $n + 1$  coefficients, also known as weights. If  $W^T Y \geq 0$ , then the LTU infers that  $Y$  belongs to one class A, otherwise the LTU infers that  $Y$  belongs to the other class B. If there are more than two classes, then one may use the LTU to partition the set of observed instances into two subsets that each may contain instances from one or more classes. The subsets can be formed by using prior knowledge about which classes are most similar or by using a partition-merit criterion during training whose goal is to find the partition that best clusters the classes into meaningful subsets.

A linear machine (LM) is a set of  $R$  linear discriminant functions that are used collectively to assign an instance to one of the  $R$  classes (Nilsson, 1965). Let  $Y$  be an instance description (a pattern vector) consisting of a constant 1 and the  $n$  features that describe the instance. Then each discriminant function  $g_i(Y)$  has the form  $W_i^T Y$ , where  $W_i$  is a vector of  $n + 1$  coefficients. A linear machine infers instance  $Y$  belongs to class  $i$  if and only if  $(\forall j, j \neq i) g_i(Y) > g_j(Y)$ . For the rare case in which  $g_i(Y) = g_j(Y)$  some arbitrary decision is made: our implementation of an LM chooses the smaller of  $i$  and  $j$  in these cases.

#### 2.4 Finding the coefficients of a multivariate test

Deferring the issue of which features should be included in a linear combination until the next section, this section addresses the issue of how to find the coefficients of a multivariate test. Given  $i$  features, one wants to find the set of coefficients that will result in the best partition of the training instances. Because the multivariate test will be used in a decision tree, this issue is different from finding the linear threshold unit or linear machine that has maximum accuracy when used to classify the training data. In a decision tree, the quality of a set of coefficients (and the test that they form) will be judged by how well the test partitions the instances into their classes. Indeed, the optimal set of coefficients for a set of instances observed at a node in the tree may not be best with respect to finding the optimal tree. The same problem exists for test selection during univariate decision tree construction.

One dimension along which one can differentiate coefficient learning algorithms is the partition-merit criterion they seek to maximize. Many coefficient algorithms maximize accuracy for the training data, and when embedded in a decision tree algorithm may fail to find a test that discriminates the instances (i.e., the test classifies all instances as from one class). This situation occurs when the highest *accuracy* can be achieved by classifying all of the instances as being from one class. However, for some data sets this characteristic may not be undesirable; for domains that contain noisy instances, this situation may occur only near the leaves of the decision tree. In such cases, classifying all instances as one class may result in the best classifier. A different criterion is used in the CART system (Breiman, Friedman, Olshen & Stone, 1984), which searches explicitly for a set of coefficients that maximizes a discrete impurity measure.

For some partition-merit criteria the problem of finding an optimal multivariate test is NP-Complete (Heath, Kasif, & Salzberg, 1993). One such criterion is to minimize the number of misclassified examples. For these types of criteria, one must use heuristic search techniques to find a good set of coefficients. A recent improvement has been to add a random component to a heuristic search procedure and to employ multiple searches (Heath

et al., 1993; Murthy, Kasif, Salzberg & Beigel, 1993). These two modifications decrease the chance of getting stuck in local minima when searching for a set of coefficients.

Another approach is to use linear programming (LP) to find a set of coefficients. Recently, Bennett and Mangasarian (1992) have created a linear programming approach that minimizes the average error of the misclassified objects. Unlike other LP methods, their method always finds an optimal nondegenerate solution in a single linear program without imposing any additional constraints.

## 2.5 Feature selection

Which features should be selected for a linear combination test? One wants to eliminate noisy and irrelevant features, minimize the number of features in the test to increase understandability, and decrease the number of features in the data that need to be evaluated. In addition, one wants to find the best combination with respect to some partition-merit criterion. For most data sets it will be impossible to try every combination of features because the number of possible combinations is exponential in the number of features. Therefore, some type of heuristic search procedure must be used. Two well-known greedy algorithms for selecting features for a linear combination test are *Sequential Backward Elimination (SBE)* and *Sequential Forward Selection (SFS)* (Kittler, 1986). An SBE search begins with all  $n$  features and removes those features that do not contribute to the effectiveness of the split. SBE is based on the assumption that it is better to search for a useful projection onto fewer dimensions from a relatively well informed state than it is to search for a projection onto more dimensions from a relatively uninformed state (Breiman, et al., 1984). An SFS search starts with zero features and sequentially adds the feature that contributes most to the effectiveness of the split. We will discuss these two approaches in detail in Section 4.

Another factor of feature selection is the decision of whether one is willing to trade accuracy for simplicity. If there is a cost associated with obtaining the value of a feature, then one can bias the selection process to select less expensive features. The criterion function used to select features can be a function of cost and quality. For example, one can restrict the number of features permitted in a test or when using an SBE search, continue to eliminate features as long as there is not more than some prespecified percentage drop in the partition-merit criterion.

In addition to searching for the best linear combination using some partition-merit criterion, one must also pay attention to the number of instances at a node relative to the number of features in the test. If the number of unique instances is not more than several times the dimensionality of the number of features in the test, then the test will *underfit* the training instances (Duda & Hart, 1973). In other words, when there are too few instances, there are many possible orientations for the hyperplane defined by the test, and there is no basis for selecting one orientation over another. In these cases the feature selection mechanism should only consider tests that will not underfit the training instances. We call this selection criterion the *underfitting criterion*. When this criterion is used with the SBE method, features will be eliminated until the test no longer underfits the training data. When used with the SFS method, features will be added only as long as the addition of a new feature will not cause the test to underfit the training data.

As mentioned in Section 2.4, coefficient learning methods that seek to maximize accuracy on the training instances may fail to find a test that discriminates the instances. To provide a partial solution to this dilemma, one can add a *discrimination criterion* to the

feature selection method. Then the goal of a feature selection algorithm is to find a linear combination test based on the fewest features that maximizes the partition merit criterion, discriminates the instances and does not underfit the training instances.

### 2.6 Avoiding overfitting

A common approach to correcting for overfitting in a decision tree model is to prune back the tree to an appropriate level. A tree is grown, which classifies all the training instances correctly, and then subtrees are pruned back to reduce future classification errors (Breiman, et al., 1984; Quinlan, 1987). Quinlan (1987b) and Mingers (1989b) compare commonly used methods.

Traditional pruning methods examine the effect of eliminating entire nodes or subtrees of the tree to determine if they should be pruned. However, although a multivariate test may overfit the training instances, pruning the entire node may result in even more classification errors. The issue here is the *granularity* of the nodes in a multivariate decision tree; the granularity can vary from a univariate test at one end of the spectrum to a multivariate test based on all  $n$  features at the other end. In the case where removing a multivariate test results in a higher estimated error, one can try to reduce the error by eliminating features from the multivariate test. Eliminating features generalizes the multivariate test; a multivariate test based on  $n - 1$  features is more general than one based on  $n$  features.

## 3 Learning the coefficients of a linear combination test

In this section we review four existing methods for learning the coefficients of a linear combination test. The first method, Recursive Least Squares (RLS) (Young, 1984), minimizes the mean-squared error over the training data. The second method, the Pocket Algorithm (Gallant, 1986), maximizes the number of correct classifications on the training data. The third method, Thermal Training (Freaun, 1990), converges to a set of coefficients by paying decreasing attention to large errors. The fourth method, CART's coefficient learning method (Breiman, et al. 1984), explicitly searches for a set of coefficients that minimizes the impurity of the partition created by the multivariate test. The RLS and CART methods are restricted to binary partitions of the data, whereas the Thermal and Pocket algorithms produce multiway partitions.

### 3.1 Recursive Least Squares procedure

The Recursive Least Squares Algorithm, invented by Gauss, is a recursive version of the Least Squares (LS) Algorithm. An LS procedure minimizes the mean squared error,  $\sum_i (y_i - \hat{y}_i)^2$  of the training data, where  $y_i$  is the true value and  $\hat{y}_i$  is the estimated value of the dependent variable,  $y$ , for instance  $i$ . For discrete classification problems, the true value of the dependent variable (the class) is either  $c$  or  $-c$ . In our implementation of the RLS procedure we use  $c = 1$ .

To find the coefficients of the linear function that minimizes the mean-squared error, the RLS algorithm incrementally updates the coefficients using the error between the estimated value of the dependent variable and the true value. Specifically, after instance  $k$  is observed,

RLS updates the weight vector,  $\mathbf{W}$ , as follows:

$$W_k = W_{k-1} - K_k(X_k^T W_{k-1} - y_k),$$

where  $X_k$  is the instance vector,  $y_k$  is the value of the independent variable (the class), and  $K_k = P_k X_k$  is the weight assigned to the update.  $P$ , the error-covariance matrix, is a weighting matrix of size  $n \times n$ , which over time decreases and smoothes the errors made by the linear combination (in statistics texts  $P$  is shown as  $\Sigma$ ). After each instance  $k$  is observed,  $P$  is updated as follows:

$$P_k = P_{k-1} - P_{k-1} X_k [1 + X_k^T P_{k-1} X_k]^{-1} X_k^T P_{k-1}$$

As more instances are observed, each individual instance has less effect on  $\mathbf{W}$ , because RLS minimizes the mean-squared error. This is reflected in the decreasing values of the entries of the matrix  $P$ .

RLS requires that one set the initial weights  $W_0$  and initialize  $P$ , the error-covariance matrix. If little is known about the true  $\mathbf{W}$  and  $W_0$  is set to zero, then the initial values of  $P$  should reflect this uncertainty: the diagonal elements should be set to large values to indicate a large initial error variance and little confidence in the initial estimate of  $W_0$ . Young (1984) suggests setting the diagonal elements to  $10^6$ . The off-diagonal elements should be set to zero when there is no *a priori* information about the covariance properties of  $\mathbf{W}$ . In this case, the best estimate is that they are zero. When there is no noise in the data and the number of instances is enough to determine  $\mathbf{W}$  uniquely, the RLS algorithm needs to see each instance only once. Otherwise, one must cycle through the instances some small number of times (our implementation cycles through the instances three times) to converge to a set of weights. For a detailed discussion of the RLS algorithm, see Young (1984).

### 3.2 The Pocket Algorithm

Gallant's (1986) Pocket Algorithm seeks a set of coefficients for a multivariate test that minimizes the *number* of errors when the test is applied to the training data. Note that this goal is different from the goal of the RLS training method, which minimizes the mean-squared error. The Pocket Algorithm uses the Absolute Error Correction Rule (Nilsson, 1965; Duda & Hart, 1973) to update the weights of an LTU (or an LM). For an LTU (or an LM), the algorithm saves in  $\mathbf{P}$  (the pocket) the best weight vector  $\mathbf{W}$  that occurs during normal perceptron training, as measured by the longest run of consecutive-correct classifications, called the *pocket count*. Assuming that the observed instances are chosen in a random order, Gallant shows that the probability of an LTU based on the pocket vector  $\mathbf{P}$  being optimal approaches 1 as training proceeds. The pocket vector is probabilistically optimal in the sense that no other weight vector visited so far is likely to be a more accurate classifier. The Pocket Algorithm fulfills a critical role when searching for a separating hyperplane because the classification accuracy of an LTU trained using the absolute error correction rule is unpredictable when the instances are not linearly separable (Duda & Hart, 1973). This algorithm was used previously in PT2, an incremental multivariate decision tree algorithm (Utgoff & Brodley, 1990).



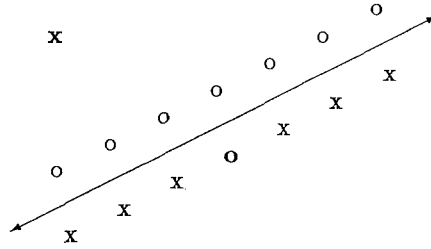


Figure 2. Nonseparable instance space.

### 3.3 The Thermal Training Procedure

The Thermal Training Procedure (Frean, 1990) can be applied to a linear threshold unit or a linear machine, but here we discuss its application to linear machines. One well known method for training a linear machine is the Absolute Error Correction Rule (Duda & Fossum, 1966), which adjusts  $W_i$  and  $W_j$ , where  $i$  is the class to which the instance belongs and  $j$  is the class to which the linear machine incorrectly assigns the instance. The correction is accomplished by  $W_i \leftarrow W_i + cY$  and  $W_j \leftarrow W_j - cY$ , where the correction,

$$c = \frac{(W_j - W_i)^T Y}{2Y^T Y} + \epsilon,$$

causes the updated linear machine to classify the instance correctly. In our implementation of this procedure  $\epsilon = 0.1$ . If the training instances are linearly separable, then cycling through the instances allows the linear machine to partition the instances into separate convex regions.

If the instances are not linearly separable, then the error corrections will not cease, and the classification accuracy of the linear machine will be unpredictable. Frean (1990) has developed the notion of a “thermal perceptron”, which gives stable behavior even when the instances are not linearly separable. Frean observed that two kinds of errors preclude convergence when the instances are not linearly separable. First, as shown in the upper left portion of Fig. 2, if an instance is far from the decision boundary and would be misclassified, then the decision boundary needs a large adjustment in order to remove the error. On the assumption that the boundary is converging to a good location, relatively large adjustments become increasingly counterproductive. To achieve stability, Frean calls for paying decreasing attention to large errors. The second kind of problematic error occurs when a misclassified instance lies very close to the decision boundary, as shown to the right of the boundary in Fig. 2. To ensure that the weights converge, one needs to reduce the effect of *all* corrections.

Utgoff and Brodley (1991a) adapted these ideas to a linear machine, yielding a *thermal linear machine*. Decreasing attention is paid to large errors by using the correction factor:

$$c = \frac{\beta}{\beta + k},$$

Table 1. Training a thermal linear machine.

- 
1. Initialize  $\beta$  to 2.
  2. If linear machine is correct for all instances or  $emag/lmag < \alpha$  for the last  $2 * n$  instances, then return ( $n =$  the number of features).
  3. Otherwise, pass through the training instances once, and for each instance  $Y$  that would be misclassified by the linear machine and for which  $k < \beta$ , immediately
    - (A) Compute correction  $c = \frac{\beta^2}{\beta+k}$ , and update  $W_i$  and  $W_j$ .
    - (B) If the magnitude of the linear machine decreased on this adjustment, but increased on the previous adjustment, then anneal  $\beta$  to  $a\beta - b$ .
  4. Go to step 2.
- 

where  $\beta$  is annealed during training and

$$k = \frac{(W_j - W_i)^T Y}{2Y^T Y} + \epsilon.$$

As the amount of error  $k$  approaches 0, the correction  $c$  approaches 1 regardless of  $\beta$ . Therefore, to ensure that the linear machine converges, the amount of correction  $c$  is annealed regardless of  $k$ . This is achieved by multiplying  $c$  by  $\beta$ , because it is already annealing, giving the correction

$$c = \frac{\beta^2}{\beta + k}.$$

Table 1 shows the algorithm for training a thermal linear machine.  $\beta$  is reduced geometrically by rate  $a$ , and arithmetically by constant  $b$ . This enables the algorithm to spend more time training with small values of  $\beta$  when it is refining the location of the decision boundary. Also note that  $\beta$  is reduced only when the magnitude of the linear machine decreased for the current weight adjustment, but increased during the previous adjustment. In this algorithm, the magnitude of a linear machine is the sum of the magnitudes of its constituent weight vectors. The criterion for when to reduce  $\beta$  is motivated by the fact that the magnitude of the linear machine increases rapidly during the early training, stabilizing when the decision boundary is near its final location (Duda & Hart, 1973). The default values for  $a$  (0.999) and  $b$  (0.0005) remained the same throughout the experiments reported in this article.

A thermal linear machine has converged when the magnitude of each correction  $k$  to the linear machine is larger than  $\beta$  for each instance in the training set. However, one does not need to continue to train until convergence; the magnitude of the linear machine asymptotes quickly, and it is at this point that training is stopped to reduce the time required to find the coefficients. To determine this point, after each update the magnitude of the new set of weight vectors and the magnitude of the error correction  $emag$  is computed. If the magnitude of the new set of weight vectors is larger than any observed thus far, it is stored in  $lmag$ . Training stops when the ratio of the magnitude of the error correction to  $lmag$  is less than  $\alpha$  for the last  $2 * n$  instances, where  $n$  is equal to the number of features in the linear machine. Empirical tests show that setting  $\alpha = .01$  is effective in reducing total training time without reducing the quality of the learned classifier (Brodley & Utgoff, 1992).

### 3.4 CART: explicit reduction of impurity

CART searches explicitly for a set of coefficients that minimizes the impurity of the partition defined by the multivariate test (Breiman, et al. 1984). The impurity is minimized if each partition contains instances from a single class; the impurity is at a maximum if all classes are represented equally in each partition. To measure the impurity of a partition, CART uses one of five partition-merit criteria, specified by the user. Only instances for which no values are missing are used during training. Each instance is normalized by centering each value of each feature at its median and then dividing by its interquartile range. After normalization, the algorithm takes a set of coefficients  $W = (w_1, \dots, w_n)$  such that

$$\|W\|^2 = \sum_{i=1}^n w_i^2 = 1,$$

and searches for the best split of the form:

$$v = \sum_{i=1}^n w_i x_i \leq c,$$

as  $c$  ranges over all possible values for a given precision. The search algorithm cycles through the features,  $x_1, \dots, x_n$ , at each step doing a search for an improved linear combination split. At the beginning of the  $L^{\text{th}}$  cycle, let the current linear combination split be  $v \leq c$ . For fixed  $\gamma$ , CART searches for the best split of the form:  $v - \delta(x_1 + \gamma) \leq c$ , such that

$$\delta \geq \frac{v - c}{x_1 + \gamma}, x_1 + \gamma \geq 0$$

and

$$\delta \leq \frac{v - c}{x_1 + \gamma}, x_1 + \gamma \leq 0.$$

The search for  $\delta$  is carried out for  $\gamma = -0.25, 0.0, 0.25$ . The resulting three splits are compared, using the chosen partition-merit criterion, and the  $\delta$  and  $\gamma$  corresponding to the best are used to update  $v$ , producing:

$$v' = \sum_{i=1}^n w'_i x_i,$$

where  $w'_1 = w_1 - \delta$ ,  $w'_i = w_i$ ,  $i > 1$  and  $c' = c + \delta\gamma$ . This search is repeated for each  $x_i$ ,  $i = 2, \dots, n$  resulting in an updated split  $v_L \leq c_L$ . The final step of the cycle is to find the best  $c_L$ , and the system searches explicitly for the split that minimizes the impurity of the resulting partition. The result of this search is used to start the next cycle. The search for the coefficients continues until the reduction in the impurity as measured by the partition-merit criterion is less than some small threshold,  $\varepsilon$ . After the final linear combination is determined, it is converted to a split on the original non-normalized features.

Table 2. Sequential Backward Elimination.

- 
1. Find a set of coefficients for a test based on all  $n$  features, producing  $T_n$ .
  2. Set  $i = n$ ,  $T_{\text{best}} = T_n$ .
  3. Find the best  $T_{i-1}$  by eliminating the feature that causes the smallest decrease of the partition-merit criterion.
  4. If the best  $T_{i-1}$  is better than  $T_{\text{best}}$ , then set  $T_{\text{best}} =$  the best  $T_{i-1}$ .
  5. If the stopping criterion is met, then stop and return  $T_{\text{best}}$ .
  6. Otherwise, set  $i = i - 1$  and go to Step 3.
- 

## 4 Feature selection

At each test node in the tree, one wants to minimize the number of features included in the test. In this section we describe five methods for selecting the features to include in a linear combination test. The two basic approaches: Sequential Backward Elimination (SBE) and Sequential Forward Selection (SFS) are described in Sections 4.1 and 4.2. In Section 4.3 we describe a version of SBE that uses a measure of the dispersion of the set of weights to determine which feature to eliminate. In Section 4.4 we describe a new heuristic approach that chooses at each node in the tree whether to perform an SFS or SBE search and in Section 4.5 we describe a feature selection mechanism, used in the CART system, that trades quality for simplicity.

### 4.1 Sequential Backward Elimination

A Sequential Backward Elimination search is a top down search method that starts with all of the features and tries to remove the feature that will cause the smallest decrease of some partition-merit criterion that reflects the amount of classification information conveyed by the feature (Kittler, 1986). Each feature of a test either contributes to, makes no difference to, or hinders the quality of the test. An SBE search iteratively removes the feature that contributes least to the quality of the test. It continues eliminating features until a specified stopping criterion is met. Table 2 shows the general SBE algorithm. To determine which feature to eliminate (Step 3), the coefficients for  $i$  linear combination tests, each with a different feature removed, are computed using the chosen method for learning the coefficients of a linear combination test.

There are two choices that must be made to implement the SBE algorithm: the choice of partition-merit criterion and the stopping criterion. For example, a partition-merit criterion may measure the accuracy of the test when applied to the training data, or measure the entropy, as with the Gini (Breiman, et al. 1984) or Information Gain Ratio (Quinlan, 1986a) criteria. The stopping criterion determines when to stop eliminating features from the linear combination test. For example, the search can continue until only one feature remains or the search can be halted if the value of the partition-merit criterion for a test based on  $i - 1$  features is less than that for a test based on  $i$  features. During the process of eliminating features, the best linear combination test with the minimum number of features,  $T_{\text{best}}$ , is saved. When feature elimination ceases, the test for the decision node is the saved linear combination test. In our implementation of the SBE algorithm we use the following stopping criterion: continue to eliminate features as long the accuracy of the current test based on  $i$  features is either more accurate or is not more than 10% less accurate than the accuracy of best test found thus far, and two or more features remain to be eliminated. This

Table 3. Sequential Forward Selection.

- 
1. Select the best of  $n$  linear combination tests, each based on a different single feature, producing  $T_1$ .
  2. Set  $i = 1$ ,  $T_{\text{best}} = T_1$ .
  3. Find the best  $T_{i+1}$  by adding the feature that causes the largest increase of the partition-merit criterion.
  4. If the best  $T_{i+1}$  is better than  $T_{\text{best}}$ , then set  $T_{\text{best}} =$  the best  $T_{i+1}$ .
  5. If the stopping criterion is met, then stop and return  $T_{\text{best}}$ .
  6. Otherwise, set  $i = i + 1$ , and go to Step 3.
- 

heuristic stopping criterion is based on the observation that if the accuracy drops by more than 10%, the chance of finding a better test based on fewer features is remote.<sup>1</sup>

#### 4.2 Sequential Forward Selection

A Sequential Forward Selection search is a bottom up search method that starts with zero features and tries to add the feature that that will cause the largest increase of some partition-merit criterion. An SFS search iteratively adds the feature that results in the most improvement of the quality of the test. It continues adding features until the specified stopping criterion is met. During the process of adding features, the best linear combination test with the minimum number of features is saved. When feature addition ceases, the test for the decision node is the saved linear combination test. Table 3 shows the general SFS search algorithm.

Like the SBE algorithm, the SFS algorithm needs a partition-merit criterion and a stopping criterion. The stopping criterion determines when to stop adding features to the test. Clearly, the search must stop when all features have been added. The search can stop before this point is reached, and in our implementation we employ a heuristic stopping criterion, based on the observation that if the accuracy of the best test based on  $i + 1$  features drops by more than 10% over the best test observed thus far, then the chance of finding a better test based on more features is remote. This observation is particularly germane in domains where some of the features are noisy or irrelevant.

#### 4.3 Dispersion-Guided Sequential Backward Elimination

The Dispersion-Guided Sequential Backward Elimination (DSBE) search is a new variation of the SBE algorithm. Instead of selecting the feature to remove by searching for the feature that causes the smallest decrease of the partition-merit criterion, DSBE selects the feature to remove that contributes the least to discriminability based on the magnitude of the weights of the LTU (or LM). This reduces the search time by a factor of  $n$ ; instead of comparing  $n$  linear combination tests when deciding which of the  $n$  features to eliminate, DSBE compares only two linear combination tests ( $T_i$  to  $T_{i-1}$ ). To be able to judge the relative importance of the features by their weights, we normalize the instances before training using the procedure described in Section 2.2.

For an LTU test, we measure the contribution of a feature to the ability to discriminate by its corresponding weight's magnitude. We choose the feature corresponding to the weight of smallest magnitude as the one to eliminate. For an LM test, we evaluate a feature's contribution using a measure of the *dispersion* of its weights over the set of classes. A feature whose weights are widely dispersed has two desirable characteristics. Firstly, a weight with

a large magnitude causes the corresponding feature to make a large contribution to the value of the discriminant function, and hence discriminability. Secondly, a feature whose weights are widely spaced across the  $R$  linear discriminant functions ( $R$  is the number of classes) makes different contributions to the value of the discrimination function of each class. Therefore, one would like to eliminate the feature whose weights are of smallest magnitude and are least dispersed. To this end, DSBE computes, for each remaining feature, the average squared distance between the weights of the linear discriminant functions for each pair of classes and then eliminates the feature that has the smallest dispersion. This measure is analogous to the Euclidean Interclass Distance Measure for estimating error (Kittler, 1986).

#### 4.4 *Heuristic Sequential Search*

The Heuristic Sequential Search (HSS) algorithm is a combination of the SFS algorithm and the SBE algorithm. Given a set of training instances, HSS first finds a linear combination test based on all  $n$  features and the best linear test based on only one feature. It then compares the quality of the two tests using the specified partition-merit criterion. If the test based on only one feature is better, then it performs a SFS search, otherwise it performs a SBE search. Although intuitively it may appear that HSS will never select the SFS search algorithm, in practice we have found that it does. If many of the features are irrelevant or noisy then the SFS algorithm will be the preferred method.

#### 4.5 *Trading quality for simplicity*

CART's linear combination algorithm differs from the previous four in three ways. Firstly, it uses only numeric features to form linear combination tests. Secondly, it uses only instances complete in the numeric features; if the value of any feature in an instance is missing, then the instance is excluded from the training instances for the linear combination. Finally, it may choose a linear combination test based on fewer features even if the accuracy of the test is lower than that of a test based on more features.

CART performs an SBE search to find a linear discriminant function that minimizes the impurity of the resulting partition. CART first searches for the coefficients of a linear combination based on all of the numeric features using the procedure described in Section 3.4. After the coefficients have been found, CART calculates, for each feature, the increase in the node impurity if the feature were to be omitted. The feature that causes the smallest increase,  $f_i$  is chosen and the threshold  $c$  is recalculated to optimize the reduction in impurity. If the increase in the impurity of eliminating  $f_i$  is less than a constant,  $\beta$ , times the maximum increase in impurity for eliminating one feature, then  $f_i$  is omitted and the search continues. Note that after each individual feature is omitted, CART searches for a new threshold  $c$ , but leaves the coefficients of the remaining features unchanged. After CART determines that further elimination is undesirable, the set of coefficients for the remaining features is recalculated. The best linear combination found by CART is added to the set of possible univariate tests and the best of this new set is chosen as a test at the current node. Therefore, even with the addition of a linear combination test, CART may still pick a univariate test. Indeed, we shall see in Section 6.3 that this is often the case.

## 5 Pruning classifiers to avoid overfitting

In Section 2.6 we discussed the issue of overfitting multivariate decision trees. In this section we describe how to prune back a multivariate decision tree. To address the problem that a multivariate test can itself overfit, we introduce a modification to the basic pruning algorithm. We call this modified algorithm *Multivariate Tree Pruning (MTP)*. The basic approach to pruning a decision tree is: for every non-leaf subtree examine the change in the estimated classification error if the subtree were replaced by a leaf labeled with the class of the majority of the training examples used to form a test at the root of the subtree. The subtree is replaced with a leaf if it lowers the estimated classification error; otherwise, the subtree is retained. There are many different methods for estimating the classification error of a subtree, which include using an independent set of instances or using crossvalidation on the training instances (Breiman, et al., 1984; Quinlan, 1987).

If pruning a subtree would result in more errors, then the MTP algorithm determines whether eliminating features from the multivariate test lowers the estimated classification error. This procedure is restricted to subtrees whose children are all leaves. (Eliminating features from an intermediate test node may change the partition that the node defines; the subtree rooted at that node would then need to be retrained to ensure the same level of accuracy.) During training, the instances used to form each test at the node are retained. To prune a multivariate test, the algorithm uses the SBE search procedure. It iteratively eliminates a feature, retrains the coefficients for the remaining features, using the original training instances and then evaluates the new test on the prune set. If the new test based on fewer features causes no rise in the estimated number of errors then elimination continues. Otherwise, the test that minimizes the estimated error rate is returned (for some data sets this will be the original test).

## 6 Evaluation of multivariate tree construction methods

To evaluate the various multivariate tree construction methods, we performed several experiments. In Section 6.1 we describe our experimental method and the data sets used in the experiments. The next three sections compare different aspects of multivariate tree construction: Section 6.2 compares the coefficient learning algorithms; Section 6.3 compares the feature selection algorithms; and Section 6.4 assesses the utility of the Multivariate Tree Pruning Algorithm.

### 6.1 Experimental method

In this section we describe the experimental method used in each of the next three sections. In each experiment we compare two or more different learning methods across a variety of learning tasks. For each learning method, we performed ten four-fold crossvalidation runs on each data set.<sup>2</sup> A crossvalidation run for one data set was performed as follows:

1. Split the original data randomly into four equal parts. For each of the four parts,  $P_i, i = 1, 2, 3, 4$ :
  - (A) Use part  $P_i$  for testing (25%) and split the remaining data (75%) randomly into training (50%) and pruning (25%) data.

Table 4. Description of the data sets.

Data Set	Classes	Instances	Features	Type	Missing
Breast	2	699	9	N	16
Bupa	2	345	6	N	0
Cleveland	2	303	13	N,S	6
Glass	6	213	9	N	0
Hepatitis	2	155	19	N,B	167
LED	10	1000	7	B	0
Segment	7	2310	19	N	0
Vowel	11	990	10	N	0

(B) Run each algorithm using this partition.

2. For each algorithm, sum the number of correct classifications of the four runs and divide by the total number of instances to compute the classification accuracy.
3. Average the other relevant measures, such as time spent learning or number of leaves in the tree.

The results of the ten four-fold cross-validations were then averaged. In the experiments we report both the sample average and standard deviation of each method's classification accuracy for the independent test sets. The conclusions that we draw in each of the following experiments must be considered in the context of our experimental assumptions: the specified experimental method for generating statistics of the algorithms; the strategy for filling in missing values and normalizing the instances; and the method of pruning back the classifiers.

Table 4 describes the chosen data sets, which were picked with the objective of covering a broad range of data set characteristics. We chose both two-class and multiclass data sets, data sets with different types of features (numeric (N), symbolic (S) and Boolean (B)), data sets for which some of the values may be missing, and data sets with different class proportions. The last column in Table 4 reports the number of values missing from each data set. Brief descriptions of each data are:

**Breast:** The breast cancer data consists of 699 instances, described by nine numeric features. The class of each instance indicates whether the cancer was benign or malignant (Mangasarian, Setiono & Wolberg, 1990).

**Bupa:** The task for this data set is to determine whether a patient has a propensity for a liver disorder based on the results of six blood tests.

**Cleveland:** This data set, compiled by Dr. Robert Detrano, et al. (1989), was collected at the Cleveland Clinic Foundation. The task is to determine whether a patient does or does not have heart disease.

**Glass:** In this domain the task is to identify glass samples taken from the scene of an accident. The examples were collected by B. German of the Home Office Forensic Science Service at Aldermaston, Reading, UK.

**Hepatitis:** The task for this domain is to predict from test results whether a patient will live, or die from hepatitis.

**LED:** Breiman, et al.'s (1984) data for the digit recognition problem consists of ten classes representing whether a 0-9 is showing on an LED display. Each attribute has a 10% probability of having its value inverted.



**Segment:** For this data set the task is to learn to segment an image into the seven classes: sky, cement, window, brick, grass, foliage and path. Each instance is the average of a  $3 \times 3$  grid of pixels represented by 17 low-level, real-valued image features. The data set was formed from seven images of buildings from the University of Massachusetts campus that were segmented by hand to create the class labels.

**Vowel:** The task is to recognize the eleven steady-state vowels of British English independent of the speaker. There are ten numeric features describing each vowel. Eight speakers were used to form a training set, and seven different speakers were used to form an independent test set. Each of the 15 speakers said each vowel six times creating 528 instances in the training set and 462 in the test set. For runs using this data set we retained the original training and test sets and therefore did not perform a cross-validation.

## 6.2 Learning coefficients

In this section, we compare three of the coefficient learning algorithms: the Pocket Algorithm, the RLS Procedure and the Thermal Training Procedure. Because the aim of this experiment is to compare the coefficient training methods for linear combinations only, we omit CART's training procedure from this comparison. (CART chooses from both linear combinations and univariate tests.) In this experiment we ran each of the three coefficient learning algorithms in conjunction with the SBE, SFS and DSBE feature-selection methods to assess the accuracy, running time and size of the trees produced by each. In each of the feature selection algorithms we used the Information Gain Ratio merit criterion (Quinlan, 1986a) and the discrimination and underfitting criteria described in Section 2.5. In addition, because one of the feature selection algorithms, DSBE, requires that the input features be normalized, we normalize the instances at each node and retain the normalization information for testing. Missing values were filled in using the sample mean (see Section 2.2).

To prune the trees we use the Reduced Error Pruning algorithm (Quinlan, 1987), which uses a set of instances, the *prune set*, that is independent of the training instances to estimate the error of a decision tree. Because our primary focus in this experiment is to evaluate the coefficient learning algorithms, we defer comparing the feature selection methods until the next section.

This section seeks to answer the following questions about the coefficient learning methods:

1. Which method achieves the best accuracy on the independent set of test instances?
2. Is there any interaction among the coefficient learning algorithms and the feature selection methods?
3. Which method takes the least amount of computation time?
4. Does the choice of coefficient learning algorithm have any affect on the size of the trees generated?

Table 5 reports the sample average and standard deviation of the classification accuracy on the independent test sets from the ten cross-validation runs for each two-class data set. We are restricted to two-class data sets because the RLS algorithm is defined for linear threshold units. RLS achieves the lowest error rate, except for the Breast data and for the Bupa data with SBE. Thermal is better than Pocket except for the Bupa data set.

Table 5. Comparison of coefficient learning algorithms: Accuracy.

Algorithm	Selection	Breast	Bupa	Cleveland	Hepatitis
Pocket	DSBE	95.91 ± 0.66	65.94 ± 2.91	78.25 ± 1.84	77.42 ± 2.06
RLS	DSBE	95.62 ± 0.32	67.28 ± 1.58	81.82 ± 1.13	81.29 ± 1.99
Thermal	DSBE	96.27 ± 0.53	65.01 ± 1.99	79.80 ± 1.84	80.52 ± 2.50
Pocket	SBE	96.15 ± 0.52	66.64 ± 2.09	78.51 ± 1.80	78.65 ± 3.11
RLS	SBE	95.44 ± 0.45	66.43 ± 1.85	81.12 ± 1.39	82.13 ± 1.80
Thermal	SBE	96.21 ± 0.47	65.77 ± 3.27	79.44 ± 1.76	78.90 ± 2.67
Pocket	SFS	95.94 ± 0.40	64.17 ± 2.07	78.78 ± 1.59	77.68 ± 3.51
RLS	SFS	95.49 ± 0.30	67.22 ± 1.68	80.92 ± 0.73	80.71 ± 1.76
Thermal	SFS	96.22 ± 0.62	64.03 ± 3.52	78.84 ± 1.50	79.94 ± 2.64

Table 6. Comparison of coefficient learning algorithms: Two-way ANOVA.

Grouping	Breast	Bupa	Cleveland	Hepatitis
Algorithm	.000	.006	.000	.000
Selection	.905	.159	.545	.779
Alg-Sel	.714	.292	.542	.346

Because the same random splits of a data set were used for each method, the difference in accuracies, within one selection method, are attributable to two factors: the coefficient learning algorithm’s ability to find the best concept representation and its dependence on the order in which instances are presented. Given a training set, RLS’s performance is invariant to the order in which instances are presented. However, each of the Pocket and Thermal algorithms may not find the same generalization given two different orderings of the instances. The sample standard deviations of each algorithm illustrate this difference; in every case RLS has the lowest sample standard deviation.

To determine whether the differences among the accuracies of the three coefficient learning algorithms are statistically significant,<sup>3</sup> we ran a two-way analysis of variance: for each data set, we grouped accuracy (the dependent variable) by algorithm, by selection, and by algorithm and selection. The results are shown in Table 6. The first row of the table shows the *p-value* that the differences among the coefficient learning algorithms is due to chance. The closer the *p-value* is to zero, the more significant the difference is. For example, a value less than 0.05 means that the probability that the observed difference among the accuracies of the ten four-fold cross-validation runs, grouped by algorithm, is due to chance is less than 5%. The second row of the table reports the probability that the differences among selection methods is due to chance and the final row reports the probability that the interaction between the coefficient training method and selection method is due to chance. The conclusions that can be drawn from this analysis are: there is a difference among algorithms, there is no difference among selection methods, and that the difference among algorithms does not depend on the selection method.

The two-way analysis of variance indicated that the difference in accuracies is due to the choice of coefficient learning algorithm. The results reported in Table 5 indicate that RLS is the best overall and that Thermal is the second best method. To determine the statistical significance of the differences between RLS and the other two methods and

Table 7. Comparison of coefficient learning algorithms: Contrasts.

Contrast	Breast	Bupa	Cleveland	Hepatitis
RLS vs. Thermal and Pocket	.000	.003	.000	.000
Thermal vs. Pocket	.060	.312	.036	.005

Table 8. Comparison of coefficient learning algorithms: CPU seconds.

Algorithm	Selection	Breast	Bupa	Cleveland	Hepatitis
Pocket	DSBE	11.9	9.7	24.1	18.8
RLS	DSBE	32.0	9.4	33.7	38.1
Thermal	DSBE	8.6	5.4	9.4	5.1
Pocket	SBE	16.2	9.6	28.7	27.5
RLS	SBE	165.8	29.9	260.1	403.9
Thermal	SBE	16.7	8.6	24.1	24.4
Pocket	SFS	23.0	19.0	72.1	75.5
RLS	SFS	104.6	21.0	124.6	203.0
Thermal	SFS	15.4	11.2	32.0	21.4

between Thermal and Pocket, we performed two contrast experiments using a one-way analysis of variance, grouping accuracy by coefficient learning algorithm. Table 7 shows the  $p$ -values of these tests. The first row reports the probability that the difference between RLS and the two other methods is due to chance across all three feature selection methods and the second row reports the probability that the difference between Thermal and Pocket is due to chance across all three selection methods. For all four data sets, the difference between RLS and the other two methods is statistically significant. Note that in one case, the Breast data, RLS is not the best method. The difference between Thermal and Pocket is statistically significant for two out of four data sets. From this analysis and from the accuracies reported in Table 5 we conclude that, overall, the ranking of coefficient training algorithms is: RLS, Thermal and Pocket.

In Table 8 we report the number of CPU seconds each algorithm used to find a multivariate decision tree.<sup>4</sup> The RLS algorithm takes much longer than the Pocket and Thermal algorithms to find a linear combination test when used with the SFS or SBE selection algorithms. There are two contributing factors to the difference in time. Firstly, RLS updates the coefficient vector for each observed instance, whereas the Pocket and Thermal algorithms update the coefficient vector only if an observed instance would be classified incorrectly by the current LTU. The second factor is the number of operations performed per update: RLS must update the error covariance matrix,  $P$ , for each update and therefore needs  $O(n^2)$  operations ( $n$  is the number of features in the LTU), whereas the Pocket and Thermal algorithms need only  $O(n)$  operations per update. This difference in training time is greatly reduced for the DSBE selection algorithm. Recall that the DSBE algorithm reduces computation time over SBE by a factor of  $n$ , where  $n$  is the number of features in the data set. Note that for the Bupa data set, the difference in training time between RLS and the other two methods is less than that for the other data sets. This is due to the fact that training time for finding the decision trees is a function of the training time of the coefficient learning algorithm and tree size. For the Bupa data set, the trees produced with RLS had

Table 9. Comparison of coefficient learning algorithms: Tree size.

Algorithm	Selection	Breast		Bupa		Cleveland		Hepatitis	
		Tests	Size	Tests	Size	Tests	Size	Tests	Size
Pocket	DSBE	3.6	23.0	10.1	43.4	4.6	40.6	2.3	25.8
RLS	DSBE	1.5	8.3	2.8	13.5	1.5	11.7	1.3	13.6
Thermal	DSBE	2.4	16.6	9.8	43.1	2.3	22.3	1.5	19.5
Pocket	SBE	2.2	15.9	8.6	40.2	3.6	38.6	2.1	35.0
RLS	SBE	1.6	8.2	3.6	17.0	1.1	9.4	1.3	14.4
Thermal	SBE	2.0	13.7	9.2	43.3	2.8	31.5	1.4	21.4
Pocket	SFS	3.1	16.1	9.9	29.2	3.6	24.0	1.7	16.1
RLS	SFS	1.9	10.0	3.6	16.8	1.3	11.5	1.2	16.1
Thermal	SFS	2.4	13.9	9.4	31.6	3.2	24.7	1.4	16.5

fewer than half as many test nodes as those produced by Pocket and Thermal. For this data set, RLS was able to find better linear combination tests, thereby reducing training time.

In Table 9 we compare the size of the trees resulting from each of the coefficient training algorithms. Each entry reports the average number of test nodes (Tests) and the average of the sum of the number of features in each test node of the tree (Size). The first measure, Tests, gives the number of test nodes in the tree and the second, Size, gives a measure of tree complexity. Because each test in the tree is binary, the number of leaves for each tree is equal to the number of test nodes plus one. In every case the RLS algorithm produced trees with fewer test nodes and smaller size than each of the Thermal and Pocket algorithms. In 8 out of 12 cases the Thermal algorithm produced smaller trees than the Pocket algorithm. Note that this ranking corresponds to the overall ranking of the algorithms by accuracy.

We draw the following conclusions from this experiment:

1. Overall, RLS achieves the best accuracy of the three methods.
2. This result is invariant of which feature selection method was used. For these four data sets there was no interaction among the coefficient learning algorithms and the feature selection methods.
3. RLS requires far more CPU time than Thermal Training or the Pocket Algorithm. In all but one case Thermal Training was the fastest of the three methods.
4. For these data sets RLS produced the smallest trees.

Although RLS is computationally more expensive than the other two methods, overall it produced smaller, more accurate decision trees. The one drawback of the RLS algorithm is that at present we know of no direct application of RLS for learning a multiway partition. One approach to using RLS for multiclass tasks would be to form subsets of the classes and use RLS to learn a binary partition. An important issue for future research is the application of RLS to multiclass data sets.

### 6.3 Feature selection methods

In this section we evaluate the feature selection methods in two ways. The first experiment compares only multivariate feature selection methods. The second experiment compares multivariate, univariate, and multivariate plus univariate feature selection methods.

Table 10. Comparison of multivariate selection methods (two-class): Accuracy.

Selection	Breast	Bupa	Cleveland	Hepatitis
DSBE	95.62 ± 0.32	67.28 ± 1.58	81.82 ± 1.13	81.29 ± 1.99
HSS	95.48 ± 0.51	67.20 ± 3.51	81.09 ± 1.48	81.94 ± 1.85
SBE	95.44 ± 0.45	66.43 ± 1.85	81.12 ± 1.39	82.13 ± 1.80
SFS	95.49 ± 0.30	67.22 ± 1.68	80.92 ± 0.73	80.84 ± 1.80

Table 11. Comparison of multivariate selection methods (multiclass): Accuracy.

Selection	Glass	LED	Segment	Vowel
DSBE	58.92 ± 1.49	73.08 ± 1.03	94.13 ± 0.28	47.40 ± 5.55
HSS	61.36 ± 2.16	73.18 ± 0.82	95.10 ± 0.35	46.28 ± 2.09
SBE	59.86 ± 3.35	73.28 ± 1.26	94.97 ± 0.50	47.04 ± 4.02
SFS	59.48 ± 3.41	73.35 ± 0.82	95.08 ± 0.55	39.61 ± 3.86

*6.3.1 Multivariate methods.* In this experiment we use the RLS coefficient learning method for two-class data sets. We use Thermal Training procedure for multiclass data sets because, as discussed above, RLS produces tests that define binary partitions of the data. Each algorithm fills in missing values using the sample mean at each node, uses the Information Gain Ratio merit criterion (Quinlan, 1986a), the discrimination and underfitting criteria, and the Reduced Error Pruning algorithm. The algorithms differ only in the feature selection procedure used. This section seeks to answer the following questions:

1. Is SBE better than SFS because of starting from an informed position?
2. Does Heuristic Sequential Search (HSS) select the best method?
3. Does Dispersion-Guided Sequential Backward Elimination (DSBE) give up any accuracy over SBE? Is the dispersion heuristic a good one?
4. How do the methods compare in terms of computation time?
5. How is tree size affected by the selection method?

Tables 10 and 11 report the sample average and standard deviation of the accuracy for each method for the two-class and multiclass data sets respectively.

SBE produced trees that made fewer errors than SFS for the Cleveland, Glass, Hepatitis and Vowel data sets. The only statistically significant differences between SBE and SFS, at the 0.05 level using paired *t*-tests, were for the Bupa, Hepatitis and Vowel data sets. Although for one case (Vowel) this difference was quite large, from these results we cannot conclude that starting from an informed position (SBE) results in trees with lower error rates; for some data sets SBE is a better search bias, whereas for others SFS is a better bias.

HSS is a combination of SBE and SFS. Overall it performs better than either of SBE and SFS. For two of the data sets, Glass and Segment, it achieves slightly higher accuracy than each of SBE and SFS. HSS was never significantly worse than either SBE or SFS, providing evidence that the heuristic for selecting between SBE and SFS is effective for choosing which of the two is a better search strategy at each node.

To evaluate the heuristic dispersion measure used in DSBE, we compare DSBE to SBE. A surprising result is that DSBE does better, albeit only slightly better, than SBE for five of the data sets. On the remaining data sets DSBE does not do much worse than SBE. For

Table 12. Comparison of multivariate selection methods: CPU seconds.

Selection	Breast	Bupa	Clev.	Glass	Hep.	LED	Seg.	Vowel
HSS	156.6	33.6	210.7	20.6	315.3	100.2	534.8	121.0
DSBE	32.2	9.1	34.4	5.7	37.6	25.7	104.3	13.8
SBE	165.5	29.7	260.9	10.1	404.6	38.9	469.9	78.4
SFS	104.9	20.8	125.4	17.1	202.8	75.1	411.7	54.3

only two of the data sets, Bupa and Segment, is the difference statistically significant at the 0.05 level using a paired  $t$ -test. One would expect DSBE to perform more poorly than SBE if, during feature elimination with DSBE, the features eliminated did not have a dispersion much lower than several others. In these cases, a different feature may be a better choice. An alternative approach that we have not tried yet, would be to combine DSBE and SBE: at each step, select one of  $i$  linear combination tests, each with a different feature removed from the set of  $i$  lowest dispersion features.

In Table 12 we report the number of seconds used by each method. The ranking of the methods from least to most time is: DSBE, SFS, HSS, SBE. These results correspond to the ranking of the methods by the number of linear combination tests that each method must evaluate in the worst case. In the worst case, DSBE must eliminate  $n - 1$  features, causing it to learn the coefficients for  $n - 1$  separate linear combination tests. If the time required to learn the coefficients for a linear combination based on  $i$  features is  $m_i$ , then the worst-case time complexity of DSBE at a node is:

$$\sum_{i=2}^n m_i = O(mn),$$

where  $m$  is the worst-case time for learning the coefficients for a linear combination test. For SFS, the worst-case time complexity at a node is:

$$\sum_{i=1}^{n-1} (n - i + 1)m_i = O(mn^2).$$

For SBE, the worst case time complexity at a node is:

$$\sum_{i=2}^n i m_{i-1} = O(mn^2).$$

SBE and SFS have the same asymptotic worst-case time complexity. However, because  $m_i$  is less than  $m_{i+1}$ , we can substitute  $i$  for  $m_i$  and rewrite the equation for SFS as

$$\sum_{i=1}^{n-1} (n - i + 1)i = (n^3 + 3n^2 - 4n)/6$$

and for SBE as

$$\sum_{i=2}^n i(i - 1) = (n^3 - 3n^2 + 2n)/3.$$

When  $n > 8$ , the worst-case time for SBE is greater than for SFS. In addition to worst-case time complexity, we consider the average case: for most data sets an SBE search will stop before all features have been eliminated and an SFS search will stop before all features have been added. Because time  $m_i$  is typically less than time  $m_{i+1}$ , on average SFS needs less time than SBE.

No single selection method produced the smallest trees in terms of total tree size. We omit a comparison table as all it would illustrate is that no one method or set of methods results in the smallest trees. The methods did differ in the number of features tested per node: SFS produced tests with fewer features than DSBE, which in turn produced tests with fewer features than SBE. However, this difference did not affect overall tree size because the trees constructed using SFS had more test nodes than those constructed using SBE.

In summary, the conclusions we draw from this experiment are:

1. Which of SBE and SFS produces better trees varies from data set to data set; starting from an informed position (SBE) is not always better.
2. HSS is effective for selecting which of SBE or SFS will produce the better tree.
3. In six out of eight cases, DSBE was not significantly different from SBE at the .05 level. Because DSBE is much faster than SBE, if time is a factor, then DSBE can be used in place of SBE.
4. The ranking of the methods from least to most computation time is DSBE, SFS, HSS, SBE.
5. Overall tree size is not affected by selection method. SFS produced tests with fewer features than DSBE, which in turn produced tests with fewer features than SBE. However, SFS produced trees with more test nodes than DSBE, which in turn produced trees with more test nodes than SBE.

*6.3.2 Multivariate and univariate methods.* In the second feature selection experiment we compare the multivariate decision tree algorithms to the univariate decision tree algorithms. In this experiment the best linear combination test found by a multivariate test procedure was added to the set of possible univariate tests, and the test that maximized the partition-merit criterion from this new set was then chosen. In the tables that report the results of this experiment, we differentiate these algorithms from those used in the previous two experiments by adding “+ Uni” to each method’s name. To assess the effect of adding multivariate tests on accuracy, tree size and training time, we included four univariate decision tree algorithms: univariate CART, C4.5 (Quinlan, 1987), and two algorithms that we implemented, Uni-I and Uni-G, that use the Information Gain Ratio and the Gini merit criterion respectively. Univariate CART was used as a control case for the multivariate CART methods. We ran three versions of CART with linear combinations added, each with a different value for  $\beta$ , CART’s discard parameter described in Section 4.5. Recall that CART differs from the other multivariate methods; it builds linear combination tests using only the numeric features and only those instances that are not missing values. CART uses the Gini merit criterion (Breiman, et al. 1984). We ran each of the other multivariate feature selection procedures twice, once with the Gini criterion and once with the Information Gain Ratio criterion, indicated in the tables by “-G” and “-I”, respectively. Uni-I and Uni-G are the control cases for comparison to the multivariate methods that we implemented. C4.5 was run with the following settings: windowing was disabled, so that like the other algorithms, C4.5 created a single decision tree from the data it was given; the Information Gain

Table 13. Comparison of feature selection methods (two-class): Accuracy.

Algorithm	Breast	Bupa	Cleveland	Hepatitis
C4.5 (Uni)	94.15 ± 0.57	62.23 ± 2.48	73.20 ± 2.16	80.32 ± 1.23
RP-C4.5 (Uni)	93.85 ± 0.72	60.26 ± 4.14	73.66 ± 1.83	80.19 ± 2.82
CART (Uni)	93.52 ± 0.86	64.00 ± 2.28	77.06 ± 1.91	79.48 ± 1.51
CART ( $\beta = 0.0$ )	95.81 ± 0.42	64.58 ± 2.79	79.27 ± 2.35	79.81 ± 2.04
CART ( $\beta = 0.1$ )	95.88 ± 0.41	64.20 ± 2.41	79.83 ± 1.50	79.68 ± 2.66
CART ( $\beta = 0.2$ )	95.72 ± 0.24	64.84 ± 2.84	79.21 ± 2.35	79.81 ± 2.26
Uni-G	94.79 ± 0.98	63.13 ± 2.62	75.81 ± 1.81	79.94 ± 3.27
DSBE-G + Uni	96.45 ± 0.37	66.35 ± 2.12	81.35 ± 1.00	80.90 ± 2.20
HSS-G + Uni	96.21 ± 0.49	67.04 ± 2.85	82.54 ± 0.82	81.10 ± 2.75
SBE-G + Uni	96.14 ± 0.45	67.04 ± 2.85	82.54 ± 0.82	81.10 ± 2.75
SFS-G + Uni	96.18 ± 0.29	63.13 ± 2.68	75.87 ± 2.12	79.81 ± 3.26
Uni-I	94.76 ± 0.49	64.46 ± 2.44	76.14 ± 2.83	79.55 ± 3.12
DSBE-I + Uni	96.21 ± 0.52	66.55 ± 1.45	80.69 ± 1.57	79.81 ± 2.56
HSS-I + Uni	96.05 ± 0.39	65.10 ± 2.04	80.50 ± 1.61	81.29 ± 2.32
SBE-I + Uni	96.04 ± 0.44	65.10 ± 2.04	80.50 ± 1.61	81.29 ± 2.32
SFS-I + Uni	96.22 ± 0.30	65.22 ± 1.50	79.74 ± 0.99	78.65 ± 1.12

Ratio criterion was used to select tests at nodes; and the pruning confidence level was set to 10%, which is the default value.

Unlike the other decision tree algorithms, C4.5 does not estimate the error during pruning with the error of an independent set of instances; to prune a tree, C4.5 estimates the error from the training data. Because the goal of this experiment is to compare selection methods for decision trees, we want all other aspects of the algorithms to be identical. Therefore, we chose not to give C4.5 the pruning instances as part of the training data, in order to keep the comparison fair. We included a modified version that uses reduced error pruning, which we call RP-C4.5. CART uses the independent set of pruning instances to estimate the true error for cost-complexity pruning (Breiman, et al. 1984).

The second feature-selection experiment seeks to answer the following questions:

1. Does adding multivariate tests improve performance over univariate decision trees?
2. How does the addition of multivariate tests affect tree size?
3. Is adding multivariate tests to univariate tests a better method than finding the best multivariate test?
4. How does CART's multivariate method compare to the other multivariate methods? Is CART sensitive to the choice of  $\beta$ ?
5. Is there any interaction between the choice of partition-merit criterion and feature-selection procedure?
6. How do the methods compare in terms of CPU time?

Tables 13 and 14 show the sample accuracy and standard deviation for each of the two-class and multiclass data sets respectively. For all four two-class data sets, adding multivariate tests improved the accuracy over univariate decision trees, with the exception of SFS-G for the Bupa, Cleveland and Hepatitis data sets. For the multiclass data sets the results were mixed. Each of the univariate control methods (CART-Uni, Uni-G and Uni-I) produced more accurate decision trees for the Glass data than their corresponding



Table 14. Comparison of feature selection methods (multiclass): Accuracy.

Algorithm	Glass	LED	Segment	Vowel
C4.5 (Uni)	63.57 ± 2.99	72.56 ± 1.00	94.49 ± 0.70	42.42 ± na
RP-C4.5 (Uni)	61.64 ± 3.00	72.69 ± 0.64	94.03 ± 0.43	na ± na
CART (Uni)	63.47 ± 2.54	71.81 ± 1.03	94.03 ± 0.49	42.16 ± na
CART ( $\beta = 0.0$ )	60.14 ± 2.56	71.97 ± 1.13	94.56 ± 0.55	47.42 ± na
CART ( $\beta = 0.1$ )	60.05 ± 2.16	72.09 ± 0.70	95.01 ± 0.56	48.01 ± na
CART ( $\beta = 0.2$ )	59.77 ± 2.10	72.12 ± 0.96	94.85 ± 0.46	43.16 ± na
Uni-G	66.34 ± 3.04	73.48 ± 1.24	94.45 ± 0.62	45.89 ± na
DSBE-G + Uni	62.39 ± 2.95	73.22 ± 0.90	95.01 ± 0.43	44.20 ± 3.38
HSS-G + Uni	60.94 ± 3.70	73.34 ± 1.28	94.44 ± 0.37	44.96 ± 3.39
SBE-G + Uni	61.22 ± 3.01	73.41 ± 0.99	94.56 ± 0.42	48.12 ± 3.90
SFS-G + Uni	61.22 ± 2.44	73.31 ± 0.63	95.26 ± 0.40	45.97 ± 2.48
Uni-I	64.04 ± 4.22	74.00 ± 0.71	94.45 ± 0.48	40.48 ± na
DSBE-I + Uni	61.17 ± 2.89	73.27 ± 0.76	94.68 ± 0.41	47.06 ± 2.10
HSS-I + Uni	60.85 ± 2.84	73.23 ± 0.87	94.58 ± 0.27	46.23 ± 4.06
SBE-I + Uni	61.17 ± 5.35	73.30 ± 0.73	94.50 ± 0.28	45.71 ± 3.25
SFS-I + Uni	61.17 ± 2.45	73.53 ± 0.77	95.08 ± 0.59	45.46 ± 3.81

Table 15. Comparison of univariate control to the multivariate methods.

Contrast	Breast	Bupa	Clev.	Glass	Hep.	LED	Seg.	Vowel
CART	0.000	0.571	0.003	0.000	0.725	0.485	0.000	na
Info	0.000	0.139	0.000	0.030	0.403	0.178	0.096	na
Gini	0.000	0.005	0.000	0.000	0.441	0.665	0.027	na

multivariate methods. For the LED data set, the multivariate and univariate methods performed roughly the same. For the Segment data set, multivariate CART performed better than univariate CART; DSBE-G and SFS-G performed better than Uni-G (there was almost no difference among SBE-G, HSS-G and Uni-G); and SFS-I performed better than Uni-I and the rest of the multivariate methods that used the Information Gain Ratio criterion. For the Vowel data set, multivariate CART performed better than univariate CART, SBE-G performed better than the rest of the Gini methods, and all of the multivariate methods that used the Information Gain Ratio performed better than their univariate control case (Uni-I).

To determine whether the differences among accuracies of the univariate methods to the multivariate methods are statistically significant, we performed a one-way analysis of variance for each set of methods (CART, Gini, and Info). For each set we contrasted the univariate control to the multivariate methods as a group. Table 15 shows the  $p$ -values for each set of methods for each data set. Note that no analysis was performed for the Vowel data set, because each of the univariate methods was run only once using the specified training and test sets.<sup>5</sup> The results of this experiment and the results reported in Tables 13 and 14 show that each group of multivariate methods is significantly better than their univariate control for the Breast, Cleveland and Segmentation data sets. The univariate methods are significantly better for the Glass data set. For the remaining data sets, neither the univariate control or the multivariate methods *as a group* are significantly better; for each of the

Table 16. Comparison of feature selection methods (two-class): Tree size.

Algorithm	Breast		Bupa		Cleveland		Hepatitis	
	Tests	Size	Tests	Size	Tests	Size	Tests	Size
C4.5	4.8	4.8	14.8	14.8	8.7	8.7	2.5	2.5
RP-C4.5	4.7	4.7	10.4	10.4	6.3	6.3	2.8	2.8
CART (Uni)	3.0	3.0	2.6	2.6	3.7	3.7	1.2	1.2
CART ( $\beta = 0.0$ )	1.0	6.5	2.6	7.0	1.2	11.0	0.5	1.6
CART ( $\beta = 0.1$ )	1.0	4.1	2.4	6.0	1.1	6.9	0.8	1.7
CART ( $\beta = 0.2$ )	1.0	4.0	1.8	5.6	1.3	5.5	0.7	2.0
Uni-G	5.8	5.8	9.8	9.8	7.5	7.5	4.1	4.1
DSBE-G + Uni	2.2	13.6	8.9	20.7	3.1	18.4	1.7	14.7
HSS-G + Uni	2.3	13.8	9.1	27.5	2.5	24.5	1.8	27.8
SBE-G + Uni	1.6	10.1	9.1	27.5	2.5	24.5	1.8	27.8
SFS-G + Uni	2.0	10.4	9.9	9.9	7.5	7.5	4.2	4.2
Uni-I	7.3	7.3	11.5	11.5	8.1	8.1	3.9	3.9
DSBE-I + Uni	2.5	15.1	8.8	17.9	3.6	21.1	2.0	13.1
HSS-I + Uni	2.3	13.4	9.1	20.3	3.2	21.7	1.5	15.9
SBE-I + Uni	1.8	11.2	9.1	20.3	3.2	21.7	1.5	15.9
SFS-I + Uni	2.3	12.3	9.6	19.5	3.3	21.6	2.0	16.3

Table 17. Comparison of feature selection methods (multiclass): Tree size.

Algorithm	Glass			LED		
	Tests	Size	L	Tests	Size	L
C4.5	12.9	12.9	13.9	22.1	22.1	23.1
C4.5-RP	7.8	7.8	8.8	22.5	22.5	23.5
CART (Uni)	7.0	7.0	8.0	19.6	19.6	20.6
CART ( $\beta = 0.0$ )	4.2	15.1	5.2	13.5	25.7	14.5
CART ( $\beta = 0.1$ )	4.7	13.2	5.7	13.1	22.3	14.1
CART ( $\beta = 0.2$ )	4.7	9.4	5.7	13.8	20.4	14.8
Uni-G	9.6	9.6	10.6	23.5	23.5	24.5
DSBE-G + Uni	5.7	33.0	15.0	6.7	33.5	33.0
HSS-G + Uni	4.6	25.7	13.0	6.9	33.7	35.4
SBE-G + Uni	4.7	29.2	13.4	6.4	31.0	32.6
SFS-G + Uni	4.8	21.2	13.6	6.7	30.4	33.5
Uni-I	10.8	10.8	11.8	23.1	23.1	24.1
DSBE-I + Uni	5.4	31.1	15.0	7.2	35.9	34.0
HSS-I + Uni	5.0	30.5	13.4	8.4	40.1	38.4
SBE-I + Uni	5.2	34.6	13.8	7.9	38.5	37.7
SFS-I + Uni	4.9	20.5	14.1	8.0	32.2	38.8

Bupa and Hepatitis data sets, at least one multivariate method from each group performs better than its univariate control method. In conclusion, adding multivariate tests improves accuracy in six cases (Breast, Bupa, Cleveland, Hepatitis, Segment and Vowel), lowers accuracy in one case (Glass) and makes no difference in one case (LED).

In Tables 16, 17 and 18 we show the size of the trees found for each method. Each entry in Table 16 shows the number of test nodes (Tests) and the average size of the tree (Size).

Table 18. Comparison of feature selection methods (multiclass): Tree size.

Algorithm	Segment			Vowel		
	Tests	Size	L	Tests	Size	L
C4.5	26.1	26.1	27.1	61.0	61.0	62.0
C4.5-RP	21.9	21.9	22.9	na	na	na
CART (Uni)	25.0	25.0	26.0	59.2	59.2	60.2
CART ( $\beta=0.0$ )	13.6	54.4	14.3	19.2	81.6	20.2
CART ( $\beta=0.1$ )	13.7	28.8	14.7	24.4	67.8	25.4
CART ( $\beta=0.2$ )	14.2	25.7	15.2	33.6	63.7	34.6
Uni-G	25.0	25.0	26.0	77.0	77.0	78.0
DSBE-G + Uni	14.6	131.0	34.0	22.6	128.0	63.0
HSS-G + Uni	14.1	152.5	34.7	19.3	87.5	61.4
SBE-G + Uni	17.3	181.7	35.9	18.9	103.8	57.1
SFS-G + Uni	10.3	63.7	26.1	21.1	79.7	63.1
Uni-I	27.2	27.2	28.2	76.0	76.0	77.0
DSBE-I + Uni	15.9	134.3	34.7	24.1	122.4	69.9
HSS-I + Uni	14.5	161.7	34.9	17.6	87.4	56.7
SBE-I + Uni	18.1	195.5	37.2	17.5	95.5	57.8
SFS-I + Uni	11.3	69.7	26.5	19.5	75.8	63.3

Table 19. Feature selection methods (two-class): MV vs. UV.

Algorithm	Breast		Bupa		Cleveland		Hepatitis	
	MV	UV	MV	UV	MV	UV	MV	UV
CART ( $\beta = 0.0$ )	1.0	0.0	2.5	0.2	1.2	0.0	0.3	0.2
CART ( $\beta = 0.1$ )	1.0	0.0	2.2	0.2	1.1	0.0	0.4	0.3
CART ( $\beta = 0.2$ )	1.0	0.0	1.7	0.1	1.3	0.0	0.3	0.4
DSBE-G + Uni	1.4	0.7	3.4	5.5	1.8	1.3	1.2	0.5
HSS-G + Uni	1.6	0.7	3.2	5.9	1.5	1.0	1.3	0.5
SBE-G + Uni	1.3	0.4	3.2	5.9	1.5	1.0	1.3	0.5
SFS-G + Uni	1.4	0.6	0.4	9.4	1.0	6.4	0.2	4.0
DSBE-I + Uni	1.6	1.0	2.7	6.1	2.0	1.7	1.2	0.9
HSS-I + Uni	1.5	0.8	3.0	6.1	1.7	1.5	1.1	0.4
SBE-I + Uni	1.4	0.4	3.0	6.1	1.7	1.5	1.1	0.4
SFS-I + Uni	1.5	0.9	3.0	6.6	1.9	1.5	1.1	0.9

In Tables 17 and 18 we include the number of leaves (L). The number of test nodes includes both linear combination tests and univariate tests. In Tables 19 and 20 we break down this measure into the number of linear combination tests (MV) and univariate tests (UV). Note that adding MV's and UV's gives the number of test nodes.<sup>6</sup>

For the two-class data sets, the multivariate methods create trees with fewer tests and leaves, but the trees are more complex. For the multiclass data sets, the multivariate methods, with the exception of CART, have either the same or more leaves than the univariate methods. On average, the trees learned by CART have fewer leaves than each of the other methods. However, for the LED data set, CART produced trees with more nodes. In addition, for both the two-class and multiclass data sets, CART produced trees with the fewest features tested per node. Examination of the number of linear tests versus univariate tests for the multiclass

Table 20. Feature selection methods (multiclass): MV vs. UV.

Algorithm	Glass		LED		Segment		Vowel	
	MV	UV	MV	UV	MV	UV	MV	UV
CART ( $\beta=0.0$ )	2.9	1.3	5.3	8.2	7.2	6.5	13.5	5.7
CART ( $\beta=0.1$ )	2.8	1.9	4.0	9.9	6.6	7.7	17.1	7.3
CART ( $\beta=0.2$ )	2.8	1.9	4.0	9.9	6.6	7.7	18.6	15.0
DSBE-G + Uni	4.4	1.3	5.1	1.6	9.3	4.7	17.7	4.9
HSS-G + Uni	3.8	0.7	5.8	1.1	8.8	2.5	15.5	3.8
SBE-G + Uni	3.8	0.9	5.2	1.2	8.5	2.7	14.9	4.0
SFS-G + Uni	4.0	0.8	5.5	1.2	8.1	1.7	15.7	5.4
DSBE-I + Uni	4.2	1.2	5.3	2.0	9.9	5.9	15.1	9.0
HSS-I + Uni	3.9	1.1	6.8	1.6	9.2	3.5	12.6	5.0
SBE-I + Uni	4.2	1.0	6.0	1.9	9.2	3.5	12.9	4.6
SFS-I + Uni	4.3	0.6	6.5	1.5	9.0	2.5	14.1	5.4

Table 21. Multivariate (M) versus multivariate + univariate (MU) tests.

Data Set	HSS		DSBE		SBE		SFS	
	Best	Sig.	Best	Sig.	Best	Sig.	Best	Sig.
Breast	MU	0.040	MU	0.019	MU	0.041	MU	0.000
Bupa	M	0.098	M	0.283	M	0.005	M	0.008
Cleveland	M	0.078	M	0.010	M	0.073	M	0.012
Glass	M	0.081	MU	0.048	MU	0.035	MU	0.012
Hepatitis	M	0.690	M	0.047	M	0.460	M	0.297
LED	MU	0.896	MU	0.656	MU	0.958	MU	0.376
Segment	M	0.005	MU	0.005	M	0.038	no dif	0.988
Vowel	M	0.977	M	0.868	M	0.451	MU	0.000

data sets shows that the ratio of univariate to multivariate tests for the trees produced by CART is higher than that of each of the other multivariate methods. In summary, adding multivariate tests decreases the number of test nodes, but increases the complexity of the tests and the overall size of the resulting tree.

To answer the question of whether the best approach is to add multivariate tests to univariate tests, we compare the results of this experiment to the results of the previous section. Note that the same random splits for each data set were used in both experiments. To compare the two approaches, multivariate-only and multivariate-plus-univariate, we examine the accuracy for each data set for each feature selection method. We use the results of the multivariate-plus-univariate trees that were built using the Information Gain Ratio criterion, because this was the partition-merit criterion used in the previous experiment. The results of this comparison are shown in Table 21. Each entry shows the more accurate of the two approaches, multivariate-only (M) or multivariate-plus-univariate (MU), and reports the results of a paired *t*-test. Excluding the Breast data set, for the two-class data sets, using multivariate tests only produced more accurate classifiers than choosing from both multivariate and univariate tests. For the Glass data set, multivariate plus univariate is significantly better than multivariate-only (M) for three cases. This is not surprising

considering that for this data set, univariate decision trees outperform the multivariate decision tree methods. There were no significant differences between M and MU for the LED and Vowel data sets, with the exception of SFS for the Vowel data. Finally for the Segment data set, the results were mixed; both HSS and SBE multivariate-only performed better than multivariate-plus-univariate, there was no difference for SFS, and multivariate-plus-univariate was better for DSBE. In conclusion, multivariate-only was statistically significantly better in 7 out of 32 cases and multivariate-plus-univariate was statistically significantly better in 9 out of 32 cases. That multivariate-only was ever better is surprising. For most of these cases, RLS was used to construct the multivariate tests. We conjecture that for some data sets, minimizing the mean-squared error will result in better decision trees. Indeed, mixing univariate tests constructed using the Information Gain Ratio and multivariate tests constructed by minimizing the mean-squared error may result in poorer performance than trees constructed to maximize a single partition-merit criterion.

To assess the affect of different choices for  $\beta$  in the CART algorithm, we performed a one-way analysis of variance. There were no statistically significant differences for any data set at the 0.05 level. Note that for the Vowel data set, there is a large difference in accuracy among the three values of  $\beta$ , but the Vowel data set was not included in the analysis because it was run only once for each value. Multivariate CART never produced the fewest errors for any of the data sets. Because there was no statistical difference in errors between the three settings of  $\beta$  we conjecture that CART's coefficient learning method does not find as good a set of coefficients as the RLS or Thermal algorithms. Indeed, a comparison of SBE-G to CART shows that SBE-G always outperforms CART, except for the Segment data set. There are two differences between CART with  $\beta$  set to 0.0 and SBE-G. Firstly, the coefficient training algorithms are different. Secondly, after CART eliminates a feature, it re-learns the weight for the constant (the threshold); the coefficients remain untouched. Only after elimination ceases, does CART retrain the coefficients. The SBE-G algorithm in contrast re-learns the coefficients for each elimination. One problem with CART's approach is that after a feature is eliminated the relative importance of the remaining features may change. Therefore, one should recalculate the coefficients after each feature is eliminated to avoid eliminating features erroneously. For the multiclass tasks there is a third difference between SBE-G and CART; SBE-G produces multivariate tests that are multiway partitions, whereas CART's multivariate tests are binary partitions. We do not however attribute the difference between the results to this third difference; even for the two-class problems, where the tests formed by both methods are binary, SBE-G performs better than CART.

To test whether there is an interaction between the choice of merit criterion and the feature selection method we ran a paired  $t$ -test for each data set to determine whether the difference in the errors between merit criteria were significant. From the results shown in Table 22 we see no interaction between the merit criteria and the feature selection methods. However, there was some interaction between choice of merit criterion and data set. For the Cleveland data set the choice of merit criterion was very important; independent of the selection algorithm, trees formed using the gain ratio criterion made fewer errors.

Table 23 reports the average number of CPU seconds used by each decision tree method. An entry of 0.0 indicates that the method required less than 0.1 seconds. The univariate methods needed the least amount of time. Multivariate CART needed far less time than the other multivariate methods. One reason that multivariate CART is faster is because CART does not re-learn the coefficients each time a feature is eliminated from a test during the

Table 22. Comparison of merit criteria (Gini vs. Info): paired *t*-test.

Selection	Breast	Bupa	Clev.	Glass	Hep.	LED	Seg.	Vowel
DSBE	0.309	0.782	0.064	0.202	0.033	0.898	0.129	0.132
HSS	0.509	0.160	0.008	0.959	0.734	0.765	0.173	0.480
SBE	0.659	0.160	0.008	0.983	0.734	0.775	0.685	0.032
SFS	0.755	0.064	0.001	0.973	0.373	0.025	0.438	0.720

Table 23. Comparison of feature selection methods: CPU seconds.

Algorithm	Breast	Bupa	Clev.	Glass	Hep.	LED	Seg.	Vowel
C4.5	1.0	0.1	0.0	0.0	0.0	0.0	25.5	10.0
RP-C4.5	1.0	0.0	0.0	0.1	0.0	0.0	25.8	na
CART (Uni)	3.0	1.2	1.3	1.2	1.0	5.0	10.6	8.5
CART ( $\beta = 0.0$ )	5.4	4.0	4.6	3.4	1.0	8.4	41.1	26.1
CART ( $\beta = 0.1$ )	6.1	4.2	5.0	3.8	1.0	8.8	44.3	36.3
CART ( $\beta = 0.2$ )	6.1	4.3	5.2	3.9	1.1	8.9	44.4	36.2
Uni-G	5.6	3.1	4.4	4.0	2.2	7.1	461.2	71.0
DSBE-G + Uni	9.8	16.5	56.8	6.7	55.0	29.1	296.7	40.4
HSS-G + Uni	23.9	45.6	325.0	23.9	513.0	104.3	654.5	127.6
SBE-G + Uni	17.5	42.4	398.6	15.8	609.9	47.6	467.6	103.5
SFS-G + Uni	18.4	32.6	230.9	18.8	432.2	78.5	733.8	102.4
Uni-I	4.3	3.0	4.0	3.3	2.0	7.1	327.8	59.0
DSBE-I + Uni	9.0	13.3	45.3	6.6	47.3	28.0	286.1	43.3
HSS-I + Uni	21.3	37.7	250.6	20.3	363.5	97.7	614.7	129.0
SBE-I + Uni	14.3	33.0	276.0	10.9	415.6	41.5	366.3	106.2
SFS-I + Uni	17.4	26.4	166.9	17.0	219.7	74.9	696.3	104.4

feature selection process. By far the most time consuming part of forming a multivariate decision tree is finding the coefficients of the linear combination tests.

From this experiment we draw the following conclusions:

1. In general, adding multivariate tests improves the performance over univariate decision trees. Excluding the Glass data set, at least one multivariate method performed better than its corresponding univariate control method.
2. Adding multivariate tests decreases the number of nodes in the tree, but increases the number of features tested per node.
3. Adding a multivariate test to the set of possible univariate tests and then selecting the best did not perform better than multivariate tests only.
4. Multivariate CART did not perform as well as the other multivariate decision tree methods. CART is not sensitive to our three choices of  $\beta$  for these data sets.
5. There was no interaction between the choice of merit criterion and feature selection method. However, there was an interaction between the choice of merit criterion and data set.
6. The univariate methods require far less CPU time than the multivariate methods. Of the multivariate methods, CART required the least amount of time.

Table 24. Difference in pruning methods for the Hepatitis data set.

Selection	Accuracy	Test size	Time
HSS	+0.6	-1.2	+0.9
DSBE	+0.4	-1.2	+1.1
SBE	+0.1	-1.7	+1.1
SFS	+0.0	-1.3	+0.5

#### 6.4 Pruning procedures

In this section we compare multivariate tree pruning (MTP) to the basic pruning algorithm (BTP), which prunes subtrees but not multivariate tests. For each of the ten crossvalidation runs of each data set, we grew a tree and then pruned it in two different ways: once with MTP and once with BTP. Except for the Hepatitis data set the trees produced by the two pruning methods were identical. Table 24 reports the difference (MTP – BTP) between the two methods for the Hepatitis data set. We report the increase in accuracy, the reduction in the average number of features tested per node, and the increase in computation time of the MTP method over the BTP method. The number of nodes remains the same for each of the methods, because multivariate test pruning is used only when replacing a node with a leaf will result in more classification errors. From these results, we conclude that in most cases MTP will not improve accuracy.

In conclusion, pruning methods for multivariate decision trees need further investigation. In our implementation of MTP we simplify a multivariate test only if all of its children are leaves. A more general approach would try to simplify *subtrees* near the leaves of the tree. Such an approach would require backtracking to create a subtree with simpler tests.

## 7 Conclusions

This article began by asserting that multivariate tests alleviate a representational limitation of univariate decision trees. In the context of our experimental assumptions, the results in Section 6.3.2 show that multivariate decision trees make fewer errors on previously unobserved instances than univariate decision trees for seven of the eight tasks. In addition to illustrating that multivariate tests are useful, this article compared various well-known and new methods for constructing multivariate decision trees. Of the four coefficient learning algorithms for linear combination tests described (CART, Pocket, RLS and Thermal), the Recursive Least Squares (RLS) method was shown to be superior on the data sets chosen. However as pointed out in Section 6.2, RLS is restricted to two-class problems. An open research problem is how to apply RLS to multiclass tasks. Of the five feature selection methods described, the results from the experiment reported in Section 6.3.1 show that Heuristic Sequential Search (HSS) is the best method. The HSS feature selection method begins to address a fundamental issue in machine learning: automatic detection of the best learning strategy for a given data set.

For one data set, adding multivariate tests led to lower classification accuracy than univariate tests only. Although the set of univariate tests are contained in the set of all possible multivariate tests, the univariate tests in our implementation were constructed using either

the Gini or Information Gain Ratio criteria, whereas the multivariate tests were constructed to minimize the mean-squared error (for two-class problems) or maximize classification accuracy (for the multiclass tasks). Which of these methods is best, depends on the data set itself. An important direction for future research is how to determine which test construction method will produce the best test for each test node in a tree.

## Acknowledgments

This material is based on work supported by the National Science Foundation under Grant No. IRI-9222766, by the National Aeronautics and Space Administration under Grant No. NCC 2-658, and by the Office of Naval Research through a University Research Initiative Program, under contract number N00014-86-K-0764. The pixel segmentation data set was donated by B. Draper from the Visions Lab at UMASS. The Breast, Bupa, Cleveland, Glass, Hepatitis, LED and Vowel data sets are from the UCI machine learning database. We wish to thank J. Ross Quinlan for providing us with a copy of C4.5 and California Statistical Software, Inc. for providing us with a copy of the CART program. Thanks to Mike Sutherland and Eva Goldwater for their statistical expertise. We thank Jason Catlett, Dennis Kibler, Jeff Clouse, Jamie Callan, Tom Fawcett, and Margie Connell for their suggestions and our reviewers whose detailed comments helped improve this article.

## Notes

1. The 10% cutoff was derived from experiments using the Segmentation, Hyper-Thyroid and DNF-5 data sets from the UCI database. Values less than 10% hindered performance, where as values greater than 10% did not increase the accuracy of the resulting classifier.
2. For the Vowel data set, we used the original training and test sets and therefore did not perform a crossvalidation.
3. We define statistical significance as: the probability of an event occurring due to chance is less than 0.05.
4. All experiments were run on a DEC Station 5000/200.
5. The multivariate methods were each run ten times, because they train the linear machines using instances sampled randomly from the training data.
6. This number may be off in the least significant decimal place due to round-off errors.

## References

- Bennett, K.P., & Mangasarian, O.L. (1992). Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1, 23–34.
- Breiman, L., Friedman, J.H., Olshen, R.A., & Stone, C.J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.
- Brodley, C.E., & Utgoff, P.E. (1992). *Multivariate versus univariate decision trees*, (Coins Technical Report 92-8), Amherst, MA: University of Massachusetts, Department of Computer and Information Science.
- Buntine, W., & Niblett, T. (1992). A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8, 75–85.
- Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., & Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, 64, 304–310.



- Duda, R.O., & Fossum, H. (1966). Pattern classification by iteratively determined linear and piecewise linear discriminant functions. *IEEE Transactions on Electronic Computers*, EC-15, 220–232.
- Duda, R.O., & Hart, P.E. (1973). *Pattern classification and scene analysis*. New York: Wiley & Sons.
- Fayyad, U.M., & Irani, K.B. (1992a). On the handling of continuous-valued attribute in decision tree generation. *Machine Learning*, 8, 87–102.
- Fayyad, U.M., & Irani, K.B. (1992b). The attribute selection problem in decision tree generation. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 104–110). San Jose, CA: MIT Press.
- Frean, M. (1990). *Small nets and short paths: Optimising neural computation*. Doctoral dissertation, Center for Cognitive Science, University of Edinburgh.
- Gallant, S.I. (1986). Optimal linear discriminants. *Proceedings of the International Conference on Pattern Recognition* (pp. 849–852). IEEE Computer Society Press.
- Hampson, S.E., & Volper, D.J. (1986). Linear function neurons: Structure and training. *Biological Cybernetics*, 53, 203–217.
- Heath, D., Kasif, S., & Salzberg, S. (1993). Induction of oblique decision trees. *The Thirteenth International Joint Conference on Artificial Intelligence*, 1002–1007.
- Kittler, J. (1986). Feature selection and extraction. In Young & Fu (Eds.), *Handbook of pattern recognition and image processing*. New York: Academic Press.
- Mangasarian, O., Setiono, R., & Wolberg, W. (1990). Pattern recognition via linear programming: Theory and application to medical diagnosis. *SIAM Workshop on Optimization*.
- Matheus, C.J. (1990). *Feature construction: An analytic framework and an application to decision trees*. Doctoral dissertation, Department of Computer Science, University of Illinois, Urbana-Champaign, IL.
- Mingers, J. (1989a). An empirical comparison of selection measures for decision tree induction. *Machine Learning*, 3, 319–342.
- Mingers, J. (1989b). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4, 227–243.
- Murthy, S., Kasif, S., Salzberg, S., & Beigel, R. (1993). OC1: Randomized induction of oblique decision trees. *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 322–327.
- Nilsson, N.J. (1965). *Learning machines*. New York: McGraw-Hill.
- Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 71–99.
- Pagallo, G.M. (1990). *Adaptive decision tree algorithms for learning from examples*. Doctoral dissertation, University of California at Santa Cruz.
- Quinlan, J.R. (1986a). Induction of decision trees. *Machine Learning*, 1, 81–106.
- Quinlan, J.R. (1986b). The effect of noise on concept learning. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. San Mateo, CA: Morgan Kaufmann.
- Quinlan, J.R. (1987). Simplifying decision trees. *International Journal of Man-machine Studies*, 27, 221–234.
- Quinlan, J.R. (1989). Unknown attribute values in induction. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 164–168). Ithaca, NY: Morgan Kaufmann.
- Safavian, S.R., & Langrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21, 660–674.
- Sutton, R.S. (1988). *NADALINE: A normalized adaptive linear element that learns efficiently*, (GTE TR88-509.4), GTE Laboratories Incorporated.
- Sutton, R.S., & Matheus, C.J. (1991). Learning polynomial functions by feature construction. *Machine Learning: Proceedings of the Eighth International Workshop* (pp. 208–212). Evanston, IL: Morgan Kaufmann.
- Utgoff, P.E., & Brodley, C.E. (1990). An incremental method for finding multivariate splits for decision trees. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 58–65). Austin, TX: Morgan Kaufmann.
- Utgoff, P.E., & Brodley, C.E. (1991). *Linear machine decision trees*, (COINS Technical Report 91-10), Amherst, MA: University of Massachusetts, Department of Computer and Information Science.
- Young, P. (1984). *Recursive estimation and time-series analysis*. New York: Springer-Verlag.