Multivariate Decision Trees

Carla E. Brodley
Paul E. Utgoff
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003 USA

COINS Technical Report 92-82
December 1992

Abstract

Multivariate decision trees overcome a representational limitation of univariate decision trees: univariate decision trees are restricted to splits of the instance space that are orthogonal to the feature's axis. This paper discusses the following 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 some new and review some well-known methods for forming multivariate decision trees. The methods are compared across a variety of learning tasks to assess each method's ability to find concise, accurate decision trees. The results demonstrate that some multivariate methods are more effective than others. In addition, the experiments confirm that allowing multivariate tests improves the accuracy of the resulting decision tree over univariate trees.

Contents

1 Introduction **1**

2 Issues in multivariate tree construction **3**
- 2.1 Symbolic and numeric features 4
- 2.2 Filling in missing values 4
- 2.3 Numerical representation of a multivariate test 5
- 2.4 Finding the coefficients of a multivariate test 5
- 2.5 Feature selection 6
- 2.6 Avoiding overfitting 7

3 Learning the coefficients of a linear combination test **7**
- 3.1 Recursive Least Squares Procedure 7
- 3.2 The Pocket Algorithm 8
- 3.3 The Thermal Training Procedure 9
- 3.4 CART: Explicit reduction of impurity 11

4 Feature selection **11**
- 4.1 Sequential Backward Elimination 12
- 4.2 Sequential Forward Selection 13
- 4.3 Greedy Sequential Backward Elimination 13
- 4.4 Heuristic Sequential Search 14
- 4.5 Trading quality for simplicity 14

5 Pruning classifiers to avoid overfitting **15**

6 Evaluation of multivariate tree construction methods **16**
- 6.1 Experimental method 16
- 6.2 Learning coefficients 17
- 6.3 Feature selection methods 22
  - 6.3.1 Multivariate methods 22
  - 6.3.2 Multivariate and univariate methods 25
- 6.4 Pruning procedures 33

7 Conclusions **34**

1 Introduction

For inductive learning, decision-tree methods are attractive for three principal reasons. Firstly, the methods find trees that generalize well to the unobserved instances, assuming that the instances are described in terms of features that are correlated with the target concept. Secondly, the methods are efficient, generally requiring a total amount of computation that is proportional to the number of observed training instances. Finally, the resulting decision tree provides a representation of the concept that humans find easy to interpret.

A decision tree is either a leaf node containing the name of a class or a decision node containing a test. For each possible outcome of the test there is a branch to a decision tree. To classify an instance, one starts at the root of the tree and follows the branch indicated by the outcome of each test until a leaf node is reached. The name of the class at the leaf is the resulting classification.

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: features are tested in one or more *subtrees* of 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 Figure 1 and the corresponding univariate decision tree. The univariate decision tree approximates the hyperplane boundary, $y + x \leq 8$, with a series of orthogonal splits. In the figure, the dotted line represents the hyperplane boundary and the solid line represents the boundary of the univariate decision tree. This example illustrates the well known problem that a univariate test can only split a space with a boundary that is orthogonal to that feature'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.

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; each test in the tree is multivariate. For example, the multivariate decision tree for the data set shown in Figure 1 consists of one test node and two leaves. The test node is the multivariate test $y + x \leq 8$. Instances for which $y + x$ is less than or equal to 8 are classified as negative; otherwise they are classified as positive.

In this paper we describe and evaluate a variety of multivariate tree construction methods. The first part is devoted to discussing the issues that must be considered to

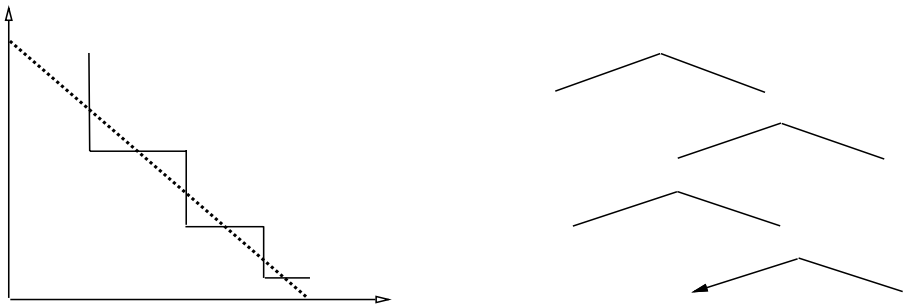

Figure 1: An example instance space; "+": positive instance, "-": negative instance. The corresponding univariate decision tree.

construct multivariate decision trees. We review some well-known methods and introduce some new methods for multivariate tree construction. The second part reports and discusses experimental results of comparisons among the various methods. The results indicate that some multivariate decision tree methods are more effective than others. In addition, our experiments confirm that allowing multivariate tests 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 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 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. In Section 3 we describe four algorithms for finding the coefficients of a linear combination test. In Section 4 we describe three well known and two new 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: simple tests may result in large trees that are difficult to understand, yet multivariate tests may result in small trees with tests that are 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. addition of multivariate tests affects general decision tree issues such as overfitting. Many of the issues for constructing multivariate decision trees are the same as for univariate decision trees. For both multivariate and univariate decision trees there are two stages in tree construction: building the decision tree from labeled examples and then pruning branches from the tree that are not statistically valid.

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 merit selection 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 a single 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 (1986a), Mingers (1989a), 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. A noisy instance is one for which either the class label is incorrect, some number of the attribute values are incorrect or a combination of the two. Noise can be caused by many different factors which include 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, et al. 1984) is to form linear combinations using only the ordered features. An alternative solution is to map each unordered feature to m ordered features, one for each observed value of the feature (Hampson & Volper, 1986; Utgoff & Brodley, 1990).

In order to map an unordered feature to a numeric feature, 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 can be mapped to a propositional feature, which is TRUE if and only if the feature has the particular value in the instance (Hampson & Volper, 1986). This 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 equal to 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.

2.3 Numerical representation of a multivariate test

For two class learning tasks, a multivariate test can be represented by a linear threshold unit (Nilsson, 1965; Duda & Hart, 1973). For multiclass tasks, there are two possible representations: 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 $\mathbf{W}^T\mathbf{Y} \geq 0$, where $\mathbf{Y}$ is an instance description (a pattern vector) consisting of a constant 1 and the n features that describe the instance. $\mathbf{W}$ is a vector of $n+1$ coefficients, also known as weights. If $\mathbf{W}^T\mathbf{Y} \geq 0$, then the LTU infers that $\mathbf{Y}$ belongs to class 1, otherwise if $\mathbf{W}^T\mathbf{Y} < 0$, then the LTU infers that $\mathbf{Y}$ belongs to class 2. If there are more than two classes, then the LTU partitions the set of observed instances into two sets that each may contain instances from one or more classes.

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 $\mathbf{Y}$ be an instance description (a pattern vector) consisting of a constant 1 and the n features that describe the instance. Then each discriminant function $g_i(\mathbf{Y})$ has the form $\mathbf{W}_i^T\mathbf{Y}$, where $\mathbf{W}_i$ is a vector of $n+1$ coefficients. A linear machine infers instance $\mathbf{Y}$ to belong to class i if and only if $(\forall j, i \neq j)\ g_i(\mathbf{Y}) > g_j(\mathbf{Y})$. For the rare case in which $g_i(\mathbf{Y}) = g_j(\mathbf{Y})$ 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.

One dimension along which we 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 one class. A different criterion is used in the CART system (Breiman, et al. 1984). CART searches explicitly for a set of coefficients that maximizes a discrete impurity measure.

2.5 Feature selection

Which features should be selected for a linear combination test? One wants to minimize the number of features in the test to both 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 greedy search procedure must be used. Two well known approaches to selecting features for a linear combination test are *Sequential Backward Elimination (SBE)* and *Sequential Forward Selection (SFS)*. 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 quality 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 the final test or when using an SBE search, continue to eliminate features as long as there is not more than an $\alpha\%$ drop in the merit criterion measure.

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 greater than twice 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 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 the appropriate level. A full tree is grown, which classifies all the training instances correctly and then subtrees are pruned back to reduce future classification errors (Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1987). Quinlan (1987) and Mingers (1989b) compare commonly used methods.

We cannot apply these pruning methods directly to multivariate decision trees. 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, we 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 describe four different 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 (Frean, 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 feature, y, for instance i. For discrete classification problems, the true value of the dependent feature (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 minimize the mean-squared error, the RLS algorithm incrementally updates the coefficients using the error between the estimated value of the dependent feature and the true value. Specifically, after instance k is observed, RLS updates the weight vector, $\mathbf{W}$, as follows: $W_k = W_{k-1} - K_k(X_k^T W_k -$

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_k is a time-variable weighting matrix of size $n \times n$, which over time decreases and smoothes the errors made by the linear combination. P is the error covariance matrix. 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, because RLS minimizes the average 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 high 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 showing 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 probabalistically 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). The Pocket Algorithm was used in PT2, an incremental multivariate decision tree algorithm (Utgoff & Brodley, 1990).

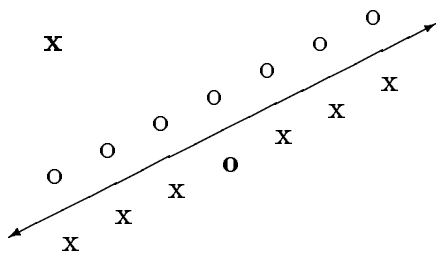

Figure 2: Nonseparable Instance Space

3.3 The Thermal Training Procedure

The thermal training procedure can be applied to a linear threshold unit or a linear machine. We discuss its application to linear machines here. (See Frean (1990) for a discussion of its application to an LTU.) One well known method for training a linear machine is the absolute error correction rule (Duda & Fossum, 1966), which adjusts $\mathbf{W}_i$ and $\mathbf{W}_j$, 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 $\mathbf{W}_i \leftarrow \mathbf{W}_i + c\mathbf{Y}$ and $\mathbf{W}_j \leftarrow \mathbf{W}_j - c\mathbf{Y}$, where correction $c = \frac{(\mathbf{W}_j - \mathbf{W}_i)^T\mathbf{Y}}{2\mathbf{Y}^T\mathbf{Y}} + \epsilon$, 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 Figure 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 are considered 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 Figure 2. To ensure that the weights converge, one needs to reduce the amount of *all* corrections.

Utgoff and Brodley (Utgoff & Brodley, 1991) extended these ideas to a linear machine, yielding a *thermal linear machine*. Decreasing attention is paid to large errors by using correction $c = \frac{\beta}{\beta + k}$, where β 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 β.

Table 1: Training a Thermal Linear Machine

1. Initialize β 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 $\mathbf{Y}$ 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 $\mathbf{W}_i$ and $\mathbf{W}_j$.
   (b) If the magnitude of the linear machine decreased on this adjustment, but increased on the previous adjustment, then anneal β to $a\beta - b$.
4. Go to step 2.

Default values are $a = 0.999$ and $b = 0.0005$.

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 β because it is already annealing giving correction $c = \frac{\beta^2}{\beta+k}$.

Table 1 shows the algorithm for training a thermal linear machine. β is reduced geometrically by rate a, and arithmetically by constant b. This enables the algorithm to spend more time training with small values of β when it is refining the location of the decision boundary. Also note that β 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 β 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 and b remain the same throughout the experiments reported in this paper.

A thermal linear machine has converged when the magnitude of each correction, k, to the linear machine is larger than β for each instance in the training set. However, one does not need to wait 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 α 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 only one class; the impurity is at a maximum if all classes are represented equally in each partition. CART uses a merit criterion function to measure the impurity of a partition. CART uses only instances for which no values are missing. CART first normalizes each instance 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 $\mathbf{W} = (w_1, ..., w_n)$ such that $||\mathbf{W}||^2 = \sum_{i=1}^{n} w_i^2 = 1$, and searches for the best split of the form: $\sum_{i=1}^{n} w_i x_i \leq c$ as c ranges over all possible values for a given precision. The search algorithm cycles through the features, $x_1, ..., x_n$, at each step doing a search for an improved linear combination split. At the beginning of the L^{th} cycle, let the current linear combination split be $v \leq c$, where $v = \sum_{i=1}^{n} w_i x_i$. For fixed γ, 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 δ is carried out for $\gamma = -.25, 0, .25$. The resulting three splits are compared, using the chosen merit criterion, and the δ and γ corresponding to the best are used to update v, $v' = \sum_{i=1}^{n} w_i' x_i$, where $w_1' = w_1 - \delta$, $w_i' = w_i, i > 1$ and $c' = c + \delta\gamma$. This search is repeated for each $x_i, i = 2, ...n$ 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 a merit criterion is less than some small threshold, ε. After the final linear combination is determined, it is converted to a split on the original nonnormalized features.

4 Feature selection

At each test node in the tree one wants to minimize the number of features 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 greedy hill-climbing version of SBE. In Section 4.4 we describe a new heuristic approach that chooses at each node in the tree whether to perform a SFS or a SBE search and in Section 4.5 we describe a feature selection mechanism, used in the CART system, that trades quality for simplicity.

Table 2: Sequential Backward Elimination

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

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 merit criterion function 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 sequential backward elimination algorithm. To determine which feature to eliminate at Step 3, we need to find the coefficients for i linear combination tests, each with a different feature removed.

There are two choices that must be made to implement the SBE algorithm: the choice of merit criterion function and the stopping criterion. For example, a merit criterion function 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 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_{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 heuristic stopping criterion is based on the observation that if the accuracy drops by more than

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 $i = 1, T_{best} = T_1$.
3. Find the best T_{i+1} by adding the feature that causes the largest increase of the merit criterion.
4. If the best T_{i+1} is better than T_{best}, then set T_{best} = the best T_{i+1}.
5. If the stopping criterion is met, then stop and return T_{best}.
6. Otherwise, set $i = i + 1$, and go to 3.

10%, the chance of finding a better test based on fewer features is remote.

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 merit criterion function. 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 merit criterion function 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 Greedy Sequential Backward Elimination

The Greedy Sequential Backward Elimination (GSBE) search is a variation of the SBE algorithm. Instead of selecting the feature to remove by searching for the feature that causes the smallest decrease of the merit criterion function, GSBE selects the feature to remove that contributes the least to discriminability based on the magnitude of the

weights of the LTU (or linear machine). 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, GSBE 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 linear discriminant functions 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, GSBE 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). This elimination procedure is used in the LMDT algorithm (Brodley & Utgoff, 1992).

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 the best 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 merit criterion function. 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 choice.

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 instances 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 quality of the test is less 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, β, 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, 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, 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. 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, Friedman, Olshen & Stone, 1984; Quinlan, 1987).

To address the problem that a multivariate test can overfit, we introduce a modification to the basic pruning algorithm. We call this modified algorithm *multivariate tree pruning*. If pruning a subtree would result in more errors, then the 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. 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 saved 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 multivariate tree pruning.

6.1 Experimental method

In this section we describe the experimental methodology 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. 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.
   (b) Run each algorithm using this partition.
2. For each algorithm, sum the classification errors of the four runs.
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 crossvalidations were then averaged. In the experiments we report both the sample average and standard deviation of the errors each method makes on the independent test sets. To determine the significance of the differences among the learning methods we used paired t-tests. Because the same random splits of each data set were used for each method, the variances of the errors for any two methods are each due to effects that are point-by-point identical.

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, symbolic and Boolean), 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 set follow:

Breast: This data set comes from the breast cancer domain in oncology and was collected at the Institute of Oncology, Ljubljani. The two classes represent the reoccurrence or non-reoccurrence of breast cancer after an operation.

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 blood tests.

Cleveland: This data set, compiled by Dr. Robert Detrano, M.D. (Detrano, Janosi, Steinbrunn, Pfisterer, Schmid, Sandhu, Guppy, Lee & Froelicher, 1989), was collected at the Clevland Clinic Foundation. The task is to determine whether a patient does or does not have a 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.

Iris: Fisher's classic data set (Fisher, 1936), contains three classes of 50 instances each. Each class is a type of iris plant. One class is linearly separable from the other two, but the latter two are not linearly separable from each other.

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. The optimal Bayes classification rate for this data set is 74%.

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 3×3 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 hand segmented to create the class labels.

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 GSBE feature selection methods to assess the accuracy, running time and size of the trees

Table 4: Description of the Data Sets

Data Set	Classes	Instances	Features	Type	Missing
Breast	2	699	9	N,S	16
Bupa	2	353	6	N	0
Cleveland	2	303	13	N,S	6
Glass	6	214	9	N	0
Hepatitis	2	155	19	N,B	167
Iris	3	150	4	N	0
LED	10	500	7	B	0
Segment	7	3210	19	N	0

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, GSBE, requires that the input features be normalized, we normalize the instances at each node and retain the normalization information for testing. 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. How do the three methods compare in terms of the size of the trees generated?

Table 5 reports the sample average and standard deviation of the errors on the independent test sets from the ten crossvalidation runs for each two-class data set. We are restricted to two-class data sets because the RLS algorithm cannot be used with linear machines. The best coefficient training method for each feature selection method is reported in bold-face type for each data set. Overall RLS achieves the lowest error rate; except for the Breast data and for the Bupa data with SBE, RLS achieves the lowest error rate. Thermal is better than Pocket with two exceptions: the Bupa data

Table 5: Comparison of Coefficient Learning Algorithms: Errors

Algorithm	Selection	Breast	Bupa	Cleveland	Hepatitis
Pocket	SFS	29.2 ± 3.4	121.0 ± 6.0	65.2 ± 7.2	32.3 ± 6.1
RLS	SFS	30.3 ± 2.4	**115.4** ± 4.2	**55.9** ± 4.3	**26.2** ± 3.3
Thermal	SFS	**27.5** ± 3.7	120.4 ± 5.9	60.3 ± 5.9	29.3 ± 3.0
Pocket	SBE	30.3 ± 4.6	**113.3** ± 12.7	63.5 ± 4.7	30.6 ± 4.7
RLS	SBE	29.5 ± 3.6	113.7 ± 3.9	**56.2** ± 4.6	**27.5** ± 3.1
Thermal	SBE	**27.7** ± 3.1	119.2 ± 5.3	59.4 ± 4.5	31.6 ± 3.4
Pocket	GSBE	29.6 ± 5.4	122.2 ± 11.5	68.8 ± 6.9	32.3 ± 3.8
RLS	GSBE	29.2 ± 1.8	**116.4** ± 6.6	**53.2** ± 4.2	**27.2** ± 3.2
Thermal	GSBE	**27.9** ± 3.4	117.5 ± 7.9	59.8 ± 4.1	28.5 ± 4.2

set with GSBE selection and the Hepatitis data set with GSBE selection. In every case except two (SBE and GSBE with the Cleveland data set), RLS has the lowest sample standard deviation. In every case except SFS with the Breast data, Thermal has a lower sample standard deviation than Pocket. The most dramatic difference is for the Bupa data set with SBE, for which the Pocket Algorithm has a sample standard deviation of 12.7 error (approximately 10% of the errors) and the RLS algorithm has a standard deviation of only 3.9 errors (approximately 3.5%).

To determine whether the differences in the errors among the three methods are significant, we performed a paired t-test between each pair of coefficient learning algorithms for each feature selection method. The results of these tests are shown in Table 6. Each entry in the table reports the probability that the difference in the paired errors is due to chance; the smaller the value the more significant the difference is. For example, a value less than 0.05 means that the probability that the observed difference between the sample averages is due to chance is less than 5%. Values of less than 0.05 are reported in bold-face type to highlight significant differences.

By combining the results from Tables 5 and 6 we see that RLS is significantly *better* at the 0.05 level than both Thermal and Pocket in eleven out of the twenty-four possible cases. The Thermal Training algorithm is significantly better than the Pocket algorithm in only three out of twelve cases: the Cleveland data set with SFS, and each of the Cleveland and Hepatitis Data sets with GSBE. From this experiment, we conclude that for these four tasks the RLS algorithm finds the best linear combinations out of the three coefficient learning algorithms.

To test for any interaction between feature selection and coefficient algorithm we ran paired t-tests between each pair of feature selection methods for each of the three coefficient learning algorithms. There were no statistically significant differences at the .05 level except for the Hepatitis data set when thermal training was used. We defer further comparison of the feature selection methods to the next section.

Table 6: Comparison of Coefficient Learning Algorithms: Paired t-test

Selection	Coeff. Pair	Breast	Bupa	Cleveland	Hepatitis
SFS	Pocket-RLS	0.317	**0.014**	**0.016**	**0.008**
SFS	Pocket-Thermal	0.318	0.806	**0.039**	0.225
SFS	RLS-Thermal	0.087	**0.037**	0.103	**0.024**
SBE	Pocket-RLS	0.727	0.929	**0.001**	0.103
SBE	Pocket-Thermal	0.182	0.171	0.183	0.679
SBE	RLS-Thermal	0.121	**0.011**	0.080	**0.014**
GSBE	Pocket-RLS	0.834	0.132	**0.000**	**0.009**
GSBE	Pocket-Thermal	0.448	0.351	**0.005**	**0.026**
GSBE	RLS-Thermal	0.340	0.667	**0.012**	0.283

Table 7: Comparison of Coefficient Learning Algorithms: CPU Seconds

Algorithm	Selection	Breast	Bupa	Cleveland	Hepatitis
Pocket	SFS	29.4	22.3	81.6	86.7
RLS	SFS	107.9	20.9	139.1	204.2
Thermal	SFS	**24.2**	**14.3**	**38.9**	**26.7**
Pocket	SBE	**23.5**	12.3	35.1	32.9
RLS	SBE	172.5	32.3	257.5	420.0
Thermal	SBE	24.5	**10.8**	**31.5**	**31.0**
Pocket	GSBE	14.9	11.9	26.7	23.2
RLS	GSBE	34.5	9.1	34.6	40.9
Thermal	GSBE	**12.7**	**7.2**	**13.0**	**6.9**

Table 8: Comparison of Coefficient Learning Algorithms: Size

Algorithm	Selection	Breast		Bupa		Cleveland		Hepatitis	
		T	F	T	F	T	F	T	F
Pocket	SFS	2.2	5.0	10.1	3.2	3.7	6.2	1.8	8.7
RLS	SFS	2.0	5.1	2.9	4.9	1.3	8.9	1.1	13.6
Thermal	SFS	2.0	5.7	9.9	3.2	2.5	8.3	1.5	11.4
Pocket	SBE	1.9	6.8	7.9	4.6	3.5	10.7	2.5	17.0
RLS	SBE	1.9	4.8	3.9	4.7	1.4	9.1	1.3	11.9
Thermal	SBE	1.7	6.7	9.6	4.6	2.4	10.8	1.3	16.1
Pocket	GSBE	2.5	7.2	12.3	4.4	4.5	8.7	2.3	12.4
RLS	GSBE	1.8	5.0	3.2	4.6	1.1	8.5	1.3	10.4
Thermal	GSBE	2.4	6.2	9.9	4.5	2.9	9.8	1.5	13.5

In Table 7 we report the number of CPU seconds [1] each algorithm used to find a multivariate decision tree. We report the smallest time for each data set and feature selection method in bold-face type. The RLS algorithm takes much longer than the Pocket and Thermal algorithms 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 GSBE selection algorithm. Recall that the GSBE algorithm reduces computation time over SBE by a factor of n, where n is the number of features in the data set.

In Table 8 we compare the size of the trees resulting from each of the coefficient training algorithms. Each entry reports the average number of test nodes (T) and the average number of features tested per linear combination (F). 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. The RLS algorithm produces trees with fewer nodes than both the Thermal and Pocket algorithms, and the Thermal algorithm produces trees with fewer nodes than the Pocket algorithm. Note that this ranking corresponds to the ranking of the algorithms by error rate. The difference in the number of nodes is most striking for the Bupa and Cleveland data sets. The coefficient training method that produced the smallest average number of features tested per node varies from data set to data set.

We draw the following conclusions from this experiment:

1. Overall RLS achieves the best accuracy of the three methods.

[1] All experiments were run on a DEC Station 5000/200

2. This results 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 method for using RLS to learn a multiway partition. Therefore, when faced with a multiclass learning task the only approach to using RLS is to form subsets of the classes and use RLS to learn a binary partition. An important issue for future research is the application of the RLS rule to linear machines.

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.

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 at the end of Section 6.2, RLS is restricted to binary partitions of the data. Each algorithm uses the gain-ratio merit criterion (Quinlan, 1986a), the discrimination and underfitting criteria, and reduced error pruning. 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 Greedy Sequential Backward Elimination(GSBE) 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?

Table 9: Comparison of Multivariate Feature Selection Methods: Errors

Data Set	HSS	GSBE	SBE	SFS
Breast	26.6 ± 4.4	**25.1** ± 2.7	26.0 ± 2.2	26.3 ± 4.1
Bupa	115.6 ± 6.2	**112.9** ± 5.5	115.8 ± 6.4	113.1 ± 5.8
Cleveland	57.2 ± 4.2	**55.1** ± 3.4	57.2 ± 4.2	57.8 ± 2.2
Glass	**82.3** ± 4.6	87.5 ± 3.2	85.5 ± 7.1	86.3 ± 7.3
Hepatitis	**27.7** ± 2.8	29.0 ± 3.1	**27.7** ± 2.8	29.9 ± 2.7
Iris	8.7 ± 1.4	9.1 ± 3.7	8.3 ± 1.2	**7.5** ± 1.3
LED	268.2 ± 8.2	269.2 ± 10.3	267.2 ± 12.6	**266.5** ± 8.2
Segment	**113.2** ± 8.1	135.5 ± 6.5	116.3 ± 11.5	113.6 ± 12.8

In Table 9 we report the sample average and standard deviation for each method. The number of errors of the most accurate method is shown in bold-face type. For each data set, we report the results of a paired t-test for each pair of selection methods in Table 10 to assess the significance of the difference of the errors. Differences that are significant at the .05 level are shown in bold-face type.

SBE produced trees that made fewer errors than SFS for the Breast, Cleveland, Glass, and Hepatitis data sets. The only statistically significant differences between SBE and SFS were for the Bupa and Hepatitis Data sets. From these results we cannot conclude that starting from an informed position (SBE) results in trees with lower error rates.

HSS is a combination of SBE and SFS. Overall it performs better than both SBE and SFS. For two of the data sets, Glass and Segment, it achieves fewer errors, although the difference is not statistically significant at the 0.05 level. For the Cleveland Data set HSS produced the same tree as SBE in each of the ten four-fold crossvalidation runs. For this data set, SBE had a lower error rate than SFS. This observation, coupled with the result that HSS was never significantly worse than either SBE or SFS, provides 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 GSBE, we compare GSBE to SBE. A surprising result is that GSBE does better, albeit only slightly better, than SBE for three of the data sets. On the remaining data sets GSBE does not do much worse than SBE, with one exception: the Segment data set. For only two of the data sets, Bupa and Segment, is the difference statistically significant.

In Table 11 we report the number of seconds used by each method. The ranking of the methods from least to most time is: GSBE, 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, GSBE must eliminate $n-1$ features, causing it to learn the coefficients for $n-1$ separate linear tests. If the time required to learn the coefficients for a linear combination based on i features

Table 10: Multivariate Feature Selection Methods: Paired t-test

Data Set	GSBE-HSS	GSBE-SBE	GSBE-SFS	HSS-SBE	HSS-SFS	SBE-SFS
Breast	0.115	0.424	0.263	0.728	0.827	0.826
Bupa	**0.029**	**0.020**	0.814	0.327	**0.005**	**0.002**
Cleveland	0.251	0.251	**0.027**	1.000	0.649	0.649
Glass	**0.032**	0.265	0.595	0.245	0.173	0.813
Hepatitis	0.142	0.142	0.419	1.000	**0.007**	**0.007**
Iris	0.733	0.516	0.152	0.438	**0.019**	0.104
LED	0.773	0.664	0.223	0.807	0.533	0.821
Segment	**0.000**	**0.002**	**0.000**	0.361	0.923	0.649

Table 11: Comparison of Multivariate Selection Methods: CPU Seconds

Data Set	HSS	GSBE	SBE	SFS
Breast	22.0	9.0	16.5	16.6
Bupa	31.3	9.3	30.0	21.0
Cleveland	205.9	33.7	261.6	125.2
Glass	18.8	5.4	10.0	16.2
Hepatitis	321.9	38.2	405.4	203.7
Iris	1.0	0.1	0.9	0.9
LED	91.7	23.6	36.6	71.5
Segment	545.3	128.1	476.9	425.1

is m_i, then the worst case time complexity of GSBE at a node is: $\sum_{i=n}^{2} m_i = O(mn)$, 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: $\sum_{i=n}^{2} i m_{i-1} = O(mn^2)$. SBE and SFS have the same asymptotic worst case time complexity. However, because m_i is smaller than m_{i+1}, we substitute i for m_i and rewrite the equation for SFS as $\sum_{i=1}^{n-1}(n-i+1)i = (n^3+9n^2+2n)/6$ and for SBE as $\sum_{i=n}^{2} i(i-1) = (2n^3-6n^2+4n)/6$. When $n \geq 15$, the worst case time for SBE is greater than SFS. In addition to worst case time complexity, we consider the average case: for most of the data sets an SBE search will stop before all features have been eliminated and an SFS search will stop before all feature have been added. Because m_i is typically less than m_{i+1}, SFS needs less time than SBE.

In Table 12 we report the average size of the trees found by each feature selection method. In each entry of the table, the first number is the average number of test nodes in the tree (T), the second number is the average number of features tested in

Table 12: Comparison of Multivariate Selection Methods: Size

Data Set	HSS			GSBE			SBE			SFS		
	T	F	L	T	F	L	T	F	L	T	F	L
Breast	2.1	7.0	3.1	2.8	5.5	3.8	2.0	6.9	3.0	2.0	5.7	3.0
Bupa	2.8	4.7	3.8	3.4	4.7	4.4	3.6	4.7	4.6	3.6	4.7	4.6
Cleveland	1.5	8.1	2.5	1.1	8.2	2.1	1.1	8.2	2.1	1.3	9.1	2.3
Glass	5.1	6.3	14.6	5.1	6.2	14.0	4.6	8.0	12.7	4.4	4.6	12.9
Hepatitis	1.3	10.3	2.3	1.3	11.6	2.3	1.3	11.6	2.3	1.2	13.1	2.2
Iris	1.6	2.9	3.7	1.4	2.5	3.4	1.6	2.6	3.6	1.5	2.1	3.5
LED	7.4	5.4	38.4	8.1	4.7	39.5	7.2	6.0	36.8	8.1	4.0	41.2
Segment	15.0	11.2	36.2	11.9	9.5	28.5	12.0	13.8	29.5	11.1	6.5	26.7

each linear combination test (F) and the final number is the average number of leaves in the tree (L). The method that produced the trees with the fewest number of nodes varies from data set to data set. For all data sets, GSBE produced trees with fewer features per linear combination test than SBE. In all but three cases, SFS tests fewer features per test than SBE. We found no relationship between the low error rates and the size of the trees in this experiment.

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 seven out of eight cases, GSBE was not significantly different from SBE at the .05 level. However, because GSBE is much faster than SBE, if time is a factor, then GSBE can be used in place of SBE.

4. The ranking of the methods from least to most computation time is GSBE, SFS, HSS, SBE.

5. SFS produced tests with fewer features than GSBE, which in turn produced tests with fewer features than SBE.

6.3.2 Multivariate and univariate methods

In the second feature selection experiment we compare the five feature selection procedures. 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 best from this new set was then chosen. We include two univariate decision tree algorithms, univariate CART and C4.5 (Quinlan, 1987) to assess the effect of adding multivariate tests on

accuracy, tree size and training time. We ran four versions of the CART program: CART with only univariate tests and three versions of CART with linear combinations added. The three versions all have a different value for β, the discard parameter described in Section 4.5. CART uses the Gini merit criterion (Breiman, et al. 1984). We ran each of the other feature selection procedures twice, once with the Gini criterion and once with the gain-ratio criterion, indicated in the tables by "-G" and "-I". For each algorithm in the experiment, the minimum number of objects on either side of a test was two. C4.5 was used with the following settings: windowing was disabled, so that like the other algorithms, C4.5 created only a single decision tree from the data it was given; and the gain ratio criterion was used to select tests at nodes.

Unlike the other decision tree algorithms, C4.5 does not use an independent set of instances to estimate the error during pruning; to prune the 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, in our experiments we include both the original C4.5 algorithm (the pruning confidence level was set to 10%, which is the default value) and a modified version that used 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.

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 the best method?
4. How does CART's multivariate method compare to the other methods? Is CART sensitive to the choice of β?
5. Is there any interaction between the choice of merit criterion and feature selection procedure?
6. How do the methods compare in terms of CPU time?

Table 13 shows the errors for each of the two-class data sets and Table 14 shows the errors for each of the multiclass data sets. We report the fewest errors in bold-face type. The tables show that adding multivariate tests improves accuracy over univariate decision trees for all four two-class data sets and for three of the four multiclass data sets. For one of the multiclass data sets, the Glass data set, adding multivariate tests lowers accuracy, although not significantly. In summary, adding multivariate tests improves the error rate significantly in five out of eight cases (Breast, Bupa, Cleveland, LED and Segment), improves the error rate slightly in two cases (Hepatitis and Glass), and lowers performance in one case (Glass).

In Tables 15 and 16 we show the size of the trees found for each method. Each entry in Table 15 shows the number of test nodes (T) and the average number of features per

Table 13: Comparison of Feature Selection Methods (Two-class): Errors

Algorithm	Breast	Bupa	Cleveland	Hepatitis
C4.5	40.9 ± 3.0	130.3 ± 8.6	81.2 ± 6.5	30.5 ± 1.9
RP-C4.5	43.0 ± 5.0	137.1 ± 14.3	79.8 ± 5.6	30.7 ± 4.4
CART (Uni)	45.3 ± 6.0	124.2 ± 7.9	69.5 ± 5.8	31.8 ± 2.3
CART (β=0.0)	29.3 ± 2.9	122.2 ± 9.6	62.8 ± 7.1	31.3 ± 3.2
CART (β=0.1)	28.8 ± 2.9	123.5 ± 8.3	61.1 ± 4.6	31.5 ± 4.1
CART (β=0.2)	29.9 ± 1.7	121.3 ± 9.8	63.0 ± 7.1	31.3 ± 3.5
GSBE-G + Uni	**24.8** ± 2.6	116.1 ± 7.3	56.5 ± 3.0	29.6 ± 3.4
GSBE-I + Uni	26.5 ± 3.7	115.4 ± 5.0	58.5 ± 4.7	31.3 ± 4.0
HSS-G + Uni	26.5 ± 3.4	**113.7** ± 9.8	**52.9** ± 2.5	29.3 ± 4.3
HSS-I + Uni	27.6 ± 2.7	120.4 ± 7.0	59.1 ± 4.9	**29.0** ± 3.6
SBE-G + Uni	27.0 ± 3.2	**113.7** ± 9.8	**52.9** ± 2.5	29.3 ± 4.3
SBE-I + Uni	27.7 ± 3.1	120.4 ± 7.0	59.1 ± 4.9	**29.0** ± 3.6
SFS-G + Uni	26.7 ± 2.1	127.2 ± 9.2	73.1 ± 6.4	31.3 ± 5.1
SFS-I + Uni	26.4 ± 2.1	120.0 ± 5.2	61.4 ± 3.0	33.1 ± 1.7

Table 14: Comparison of Feature Selection Methods (Multiclass): Errors

Algorithm	Glass	Iris	LED	Segment
C4.5	**77.6** ± 6.4	9.8 ± 2.6	274.4 ± 10.0	127.2 ± 16.2
RP-C4.5	81.7 ± 6.4	10.3 ± 1.7	273.1 ± 6.4	137.9 ± 10.0
CART (Uni)	77.8 ± 5.4	9.4 ± 1.6	281.9 ± 10.3	137.8 ± 11.3
CART (β=0.0)	84.9 ± 5.4	8.3 ± 2.2	280.3 ± 11.3	125.7 ± 12.7
CART (β=0.1)	85.1 ± 4.6	8.2 ± 1.8	279.1 ± 7.0	115.3 ± 12.8
CART (β=0.2)	85.7 ± 4.5	8.6 ± 2.2	278.8 ± 9.6	119.0 ± 10.5
GSBE-G + Uni	83.6 ± 5.0	8.6 ± 2.6	**262.3** ± 13.4	115.3 ± 10.0
GSBE-I + Uni	84.1 ± 6.4	9.2 ± 2.3	267.9 ± 7.2	123.0 ± 9.4
HSS-G + Uni	80.2 ± 6.5	7.9 ± 2.1	263.5 ± 9.1	128.5 ± 8.5
HSS-I + Uni	80.7 ± 5.5	8.0 ± 1.9	266.8 ± 10.2	125.1 ± 6.3
SBE-G + Uni	85.1 ± 6.6	8.5 ± 2.0	267.3 ± 9.4	125.6 ± 9.7
SBE-I + Uni	84.2 ± 4.8	8.7 ± 2.6	265.4 ± 11.7	127.1 ± 6.5
SFS-G + Uni	83.0 ± 5.0	**7.2** ± 2.9	263.7 ± 6.3	**109.6** ± 9.2
SFS-I + Uni	85.3 ± 7.3	8.5 ± 1.5	266.5 ± 10.3	113.7 ± 13.6

Table 15: Comparison of Feature Selection Methods (Two-class): Size

Algorithm	Breast		Bupa		Cleveland		Hepatitis	
	T	F	T	F	T	F	T	F
C4.5	4.8	1.0	14.8	1.0	8.7	1.0	2.5	1.0
RP-C4.5	4.7	1.0	10.4	1.0	6.3	1.0	2.8	1.0
CART (Uni)	3.0	1.0	2.6	1.0	3.7	1.0	1.2	1.0
CART (β=0.0)	1.0	6.5	2.6	2.7	1.2	8.3	0.5	3.2
CART (β=0.1)	1.0	4.1	2.4	2.5	1.1	6.3	0.8	2.1
CART (β=0.2)	1.0	4.0	1.8	3.1	1.3	4.2	0.7	2.9
GSBE-G + Uni	2.2	6.3	8.9	2.3	3.1	5.9	1.7	8.7
GSBE-I + Uni	2.5	5.9	8.8	2.0	3.6	5.8	2.0	6.5
HSS-G + Uni	2.3	6.0	9.1	3.0	2.5	10.0	1.8	15.9
HSS-I + Uni	2.3	5.9	9.1	2.2	3.2	6.7	1.5	10.4
SBE-G + Uni	1.6	6.1	9.1	3.0	2.5	10.0	1.8	15.9
SBE-I + Uni	1.8	6.3	9.1	2.2	3.2	6.7	1.5	10.4
SFS-G + Uni	2.0	5.3	9.9	1.0	7.5	1.0	4.2	1.0
SFS-I + Uni	2.3	5.2	9.6	2.0	3.3	6.4	2.0	8.3

test (F), which can range from one to the number of input features. Each entry in Table 16 shows the number of test nodes (T), the average number of features per *test* (F), and the number of leaves (L). The number of tests includes both linear combination tests and univariate tests. In Tables 17 and 18 we break down this number 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 T, although this number may be off in the least significant decimal place due to round-off errors.

In a decision tree with univariate tests one would like to minimize the number of nodes and leaves. In a multivariate decision tree, one would like to minimize the number of nodes, leaves and the number of features included in each multivariate test in the tree. Fayyad and Irani (1990) have shown probabalistically that minimizing the number of leaves in a decision tree leads to the most accurate classifiers; in a decision tree each leaf represents a homogeneous region in the instance space and the tests at the internal nodes describe these regions. In addition, the results of Blumer et al. (1987) show that in general, a small classifier is preferable to a large classifier.

For the two-class data sets the multivariate methods create trees with fewer tests and leaves, but the nodes 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

Table 16: Comparison of Feature Selection Methods (Multiclass): Size

Algorithm	Glass			Iris			LED			Segment		
	T	F	L	T	F	L	T	F	L	T	F	L
C4.5	12.9	1.0	13.9	2.5	1.0	3.5	22.2	1.0	23.2	26.1	1.0	27.1
RP-C4.5	7.7	1.0	8.7	2.2	1.0	3.2	22.5	1.0	23.5	21.9	1.0	22.9
CART (Uni)	7.0	1.0	8.0	2.5	1.0	3.5	19.6	1.0	20.6	25.0	1.0	26.0
CART (β=0.0)	4.2	3.6	5.2	2.1	1.7	3.1	13.5	1.9	14.5	13.6	4.0	14.3
CART (β=0.1)	4.7	2.8	5.7	2.1	1.6	3.1	13.1	1.7	14.1	13.7	2.1	14.7
CART (β=0.2)	4.7	2.0	5.7	2.3	1.3	3.3	13.8	1.5	14.8	14.2	1.8	15.2
GSBE-G + Uni	5.3	5.5	13.9	1.5	2.8	3.5	7.2	4.6	33.0	14.6	9.0	34.0
GSBE-I + Uni	5.5	5.4	14.6	1.6	2.9	3.6	7.4	4.9	34.9	15.9	8.4	34.7
HSS-G + Uni	5.3	5.4	14.1	1.4	2.4	3.4	7.9	4.5	33.3	14.1	10.9	34.7
HSS-I + Uni	5.1	6.1	14.0	1.5	2.4	3.5	8.2	5.0	37.3	14.5	11.1	34.9
SBE-G + Uni	4.3	6.6	12.6	1.3	2.7	3.3	7.6	4.8	31.4	17.3	10.5	35.9
SBE-I + Uni	4.7	6.6	13.1	1.4	2.4	3.4	8.5	4.6	34.3	18.1	10.8	37.2
SFS-G + Uni	4.9	4.2	14.0	1.5	1.9	3.5	7.2	4.4	34.7	10.3	6.2	26.1
SFS-I + Uni	4.8	4.7	13.6	1.4	2.0	3.4	7.8	4.0	37.0	11.3	6.2	26.5

of linear tests versus univariate tests for the multiclass data sets shows that the ratio of univariate to multivariate tests for the trees produced by CART is higher than that of any of the other multivariate methods. In summary, adding multivariate tests decreases the number of tests, but increases the complexity of the tests.

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 errors for each data set for each feature selection method. We use the results of the multivariate plus univariate trees that were built using the gain-ratio merit criterion, because the gain ratio was used in the previous experiment. The results of this comparison are shown in Table 19. Each entry shows the better of the two approaches, multivariate only (M) or multivariate plus univariate (MU), and reports the results of a paired t-test. The results show that using MU is better for eight out of 24 cases. Across the four feature selection methods, selecting from the combined set of possible univariate tests and a multivariate tests made a significant difference in only two cases: Glass with HSS and Segment with GSBE. What is surprising is that using only multivariate tests is significantly better in ten out of 24 cases.

To assess the effect of different choices for β in the CART algorithm we ran paired t-tests between each pair of different settings for each data set. There were no statistically significant differences at the 0.05 level. Multivariate CART never produced the fewest errors for any of the data sets. Because there was no statistical difference in

Table 17: Feature Selection Methods (Two-class): MV vs. UV

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

Table 18: Feature Selection Methods (Multiclass): MV vs. UV

Algorithm	Glass		Iris		LED		Segment	
	MV	UV	MV	UV	MV	UV	MV	UV
CART (β=0.0)	2.9	1.3	0.9	1.1	5.3	8.2	7.2	6.5
CART (β=0.1)	2.8	1.9	0.5	1.8	4.0	9.9	6.6	7.7
CART (β=0.2)	2.8	1.9	0.5	1.8	4.0	9.9	6.6	7.7
GSBE-G + Uni	4.4	1.3	1.5	0.2	5.1	1.6	9.3	4.7
GSBE-I + Uni	4.2	1.2	1.4	0.3	5.3	2.0	9.9	5.9
HSS-G + Uni	3.8	0.7	1.3	0.2	5.8	1.1	8.8	2.5
HSS-I + Uni	3.9	1.1	1.3	0.3	6.8	1.6	9.2	3.5
SBE-G + Uni	3.8	0.9	1.2	0.2	5.2	1.2	8.5	2.7
SBE-I + Uni	4.2	1.0	1.3	0.3	6.0	1.9	9.2	3.5
SFS-G + Uni	4.0	0.8	1.2	0.2	5.5	1.2	8.1	1.7
SFS-I + Uni	4.3	0.6	1.1	0.2	6.5	1.5	9.0	2.5

Table 19: Multivariate (M) versus Multivariate + Univariate (MU) Tests

Data Set	HSS		GSBE		SBE		SFS	
	Best	Sig.	Best	Sig.	Best	Sig.	Best	Sig.
Breast	M	0.311	M	0.585	M	0.110	M	0.920
Bupa	M	0.283	M	0.163	M	0.005	M	0.008
Cleveland	M	0.010	M	0.073	M	0.073	M	0.012
Glass	MU	0.047	MU	0.690	MU	0.460	MU	0.297
Hepatitis	M	0.048	M	0.035	M	0.035	M	0.015
Iris	MU	0.702	M	0.941	M	0.159	M	0.066
LED	M	0.656	MU	0.896	MU	0.958	no dif	0.376
Segment	M	0.005	MU	0.005	M	0.038	no dif	0.988

Table 20: Comparison of Merit Criteria (Gini vs. Info): Paired t-test

Selection	Breast	Bupa	Clev	Glass	Hep.	Iris	LED	Segment
GSBE	0.309	0.782	0.064	0.202	**0.033**	0.449	0.898	0.129
HSS	0.509	0.160	**0.008**	0.959	0.734	0.766	0.765	0.173
SBE	0.659	0.160	**0.008**	0.983	0.734	0.085	0.775	0.685
SFS	0.755	**0.064**	**0.001**	0.973	0.373	0.266	**0.025**	0.438

errors between the three settings of β 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 Iris data where CART with β set to 0.0 or 0.1 is better by .2 and .3 respectively). There are two differences between CART with β 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

Table 21: Comparison of Feature Selection Methods: CPU Seconds

Algorithm	Breast	Bupa	Clev	Glass	Hep.	Iris	LED	Seg.
C4.5	1.0	0.1	0.0	0.0	0.0	0.0	0.0	25.5
RP-C4.5	1.0	0.0	0.0	0.1	0.0	0.0	0.0	25.8
CART (Uni)	3.0	1.2	1.3	1.2	1.0	0.0	5.0	10.6
CART ($\beta = 0.0$)	5.4	4.0	4.6	3.4	1.0	1.0	8.4	41.1
CART ($\beta = 0.1$)	6.1	4.2	5.0	3.8	1.0	1.0	8.8	44.3
CART ($\beta = 0.2$)	6.1	4.3	5.2	3.9	1.1	1.0	8.9	44.4
GSBE-G	9.8	16.5	56.8	6.7	55.0	0.8	29.1	296.7
GSBE-I	9.0	13.3	45.3	6.6	47.3	0.8	28.0	286.1
HSS-G	23.9	45.6	325.0	23.9	513.0	1.0	104.3	654.5
HSS-I	21.3	37.7	250.6	20.3	363.5	1.0	97.7	614.7
SBE-G	17.5	42.4	398.6	15.8	609.9	1.0	47.6	467.6
SBE-I	14.3	33.0	276.0	10.9	415.6	1.0	41.5	366.3
SFS-G	18.4	32.6	230.9	18.8	432.2	1.0	78.5	733.8
SFS-I	17.4	26.4	166.9	17.0	219.7	0.9	74.9	696.3

the feature selection method we ran a paired t-test for each data set to determine if the difference in the errors between merit criteria were significant. The results are shown in Table 20 (differences at the .05 level are shown in bold-face type). From the table we see that there is no interaction between the merit criteria and the feature selection methods. However, there was some interaction between choice of merit criteria and data set. For the Cleveland data set the choice of merit criteria was very important; independent of the selection algorithm, trees formed using the gain ratio criterion made fewer errors.

Table 21 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 one second. 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 feature selection process. By far the most time consuming part of forming a multivariate test is finding the coefficients of the test.

From this experiment we draw the following conclusions:

1. In general, adding multivariate tests improves the performance over univariate decision trees.

2. Adding multivariate tests decreases the number of nodes in the tree, but increases the number of features tested per node. It does not increase the number of the input features that are tested somewhere in the tree.

3. Adding a multivariate test to the set of possible univariate tests and then selecting

Table 22: Difference in Pruning Methods for the Hepatitis Data Set

Selection	Errors	Test Size	Time
HSS	-1.0	-1.2	+0.9
GSBE	-0.6	-1.2	+1.1
SBE	-0.2	-1.7	+1.1
SFS	0.0	-1.3	+0.5

the best did not perform as well as multivariate tests only.

4. CART is not particularly sensitive to the choice of β.

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 time than the multivariate methods. Of the multivariate methods, CART required the least amount of CPU time.

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 22 reports the difference (MTP - BTP) between the two methods for the Hepatitis data set. We report the reduction in errors, 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 pruning a node 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 paper began by asserting that multivariate tests alleviate a representational limitation of univariate decision trees. The experimental 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 paper compared various well-known and new methods for constructing multivariate decision trees. Of the four coefficient learning algorithms for linear combination tests described, 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 use RLS for 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. A surprising result of this research was that the addition of multivariate tests to the set of possible univariate tests did not always lead to better decision trees. For one data set, including multivariate tests increased the number of errors. This result suggests that the method for choosing between a multivariate and a univariate test using the merit criterion does not necessarily produce the best partition of the data. This observation coupled with the result that HSS was the best feature selection method points to a new direction for future research: how to determine automatically for each test node in a tree whether a univariate or a multivariate test is a better representation.

Acknowledgments

This material is based upon work supported 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, Iris and LED 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. We thank Jeffery Clouse, Jamie Callan, Tom Fawcett, Margie Connell, Gila Kamhi and Wray Buntine for their helpful comments.

References

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1987). *Learnability and the Vapnik-Chervonenkis dimension*, (UCSC-CRL-87-20), Santa Cruz, CA: University of California.

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 disese. *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. (1990). What should be minimized in a decision tree? *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 749-754). Boston, MA: Morgan Kaufmann.

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.

Fisher, R. A. (1936). Multiple measures in taxonomic problems. *Annals of Eugenics, 7,* 179-188.

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.

Kittler, J. (1986). Feature selection and extraction. In Young & Fu (Eds.), *Handbook of pattern recognition and image processing.* New York: Academic Press.

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.

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. *Internation 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.