Multivariate versus Univariate Decision Trees

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

COINS Technical Report 92-8
January 1992

Abstract

In this paper we present a new multivariate decision tree algorithm LMDT, which combines linear machines with decision trees. LMDT constructs each test in a decision tree by training a linear machine and then eliminating irrelevant and noisy variables in a controlled manner. To examine LMDT's ability to find good generalizations we present results for a variety of domains. We compare LMDT empirically to a univariate decision tree algorithm and observe that when multivariate tests are the appropriate bias for a given data set, LMDT finds small accurate trees.

1 Introduction

One commonly used approach for learning from examples is to induce a univariate decision tree (Hunt, Marin & Stone, 1966; Breiman, Friedman, Olshen & Stone, 1984; Quinlan, 1986). Each test in a univariate tree is based on one of the input variables and therefore, is restricted to representing a split through the instance space that is orthogonal to the variable's axis. Such a bias may be inappropriate for problems in which the input variables are related numerically (Breiman, Friedman, Olshen & Stone, 1984; Utgoff & Brodley, 1990). In this paper we present a new multivariate decision tree algorithm, LMDT, designed to overcome the bias of univariate splits. LMDT constructs each test in a decision tree by training a linear machine and eliminating irrelevant and noisy variables in a controlled manner. Two factors enable LMDT to find good generalizations effectively. Firstly, the method by which LMDT finds and eliminates noisy or irrelevant variables offers a computationally efficient approach to finding multivariate splits. Secondly, the linear machine training procedure enables LMDT to find a good partition of the instance space regardless if the space is or is not linearly separable.

To evaluate LMDT's ability to uncover the structure in the data and to find trees with high predictive accuracy we examine the trees found by LMDT across a variety of classification tasks. To understand under what circumstances the bias of a multivariate tree (and LMDT's search bias for finding such a tree) is more appropriate than the bias of a univariate decision tree we compare LMDT to a univariate decision tree algorithm, C4.5 (Quinlan, 1987), across these tasks. The results of this comparison show that each approach has a *selective* superiority; for some of the tasks LMDT finds significantly more accurate trees than C4.5 and for others the reverse is true. Because the hypothesis space searched by a multivariate decision tree algorithm includes the hypothesis space of a univariate decision tree algorithm we conclude that for some problems LMDT's search bias for finding a tree is inappropriate.

2 The LMDT Induction Algorithm

The LMDT algorithm builds a multiclass, multivariate decision tree using a top-down approach. For each decision node in the tree, LMDT trains a linear machine, based on a subset of the input variables, which then serves as a multivariate test for the decision node. A linear machine (Nilsson, 1965; Duda & Hart, 1973) is a multiclass linear discriminant, which itself classifies an instance. The class name is the result of the linear machine test with one branch for each possible class at the node. To classify an instance, one encodes it according to the local encoding information retained in the decision node, and follows the branch indicated by the linear machine. This process is repeated until a leaf node is reached, indicating the class to which the instance is assumed to belong.

2.1 Encoding the Input Variables

LMDT can handle instances described by numeric and/or symbolic variables, in which some of the values may be missing. The linear machine algorithm requires that all variables be numeric and therefore, LMDT encodes symbolic variables to numeric variables using the same method as PT2, ensuring that no order is placed on these variables (Utgoff & Brodley, 1990). The encoded symbolic and numeric variables are normalized automatically at each node. This is done so that it is meaningful to gauge the relative importance of the encoded variables by the magnitudes of the corresponding weights, which is essential for the variable elimination mechanism described in Section 2.3. Scaling is accomplished for each encoded variable by mapping it to standard normal form, i.e., zero mean and unit standard deviation. LMDT's approach to handling a missing value is to map it to 0, which corresponds to the sample mean of the corresponding encoded variable. All encoding information is computed dynamically at each node and is retained in the tree for the purpose of classifying instances.

2.2 Training a Linear Machine

As per Nilsson (1965), a *linear machine* is a set of R linear discriminant functions that are used collectively to assign an instance to one of the R classes. Let $\mathbf{Y}$ be an instance description (a pattern vector) consisting of a constant threshold value 1 and the numerically encoded features, which 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 adjustable coefficients, also known as weights. A linear machine infers instance $\mathbf{Y}$ to belong to class i if and only if $(\forall j, i \neq j)\ g_i(\mathbf{Y}) > g_j(\mathbf{Y})$.

One well known method, for training a linear machine, is the absolute error correction rule (Duda & Fossum, 1966), which adjusts $\mathbf{W}_i$, where i is the class to which the instance belongs, and $\mathbf{W}_j$, where 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 $c = \left\lceil \frac{(\mathbf{W}_j - \mathbf{W}_i)^T\mathbf{Y}}{2\mathbf{Y}^T\mathbf{Y}} \right\rceil$, is the smallest integer such that the updated linear machine will classify the instance correctly. 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. Recently, Frean (1990) has developed the notion of a "thermal perceptron", which gives stable behavior even when the instances are not linearly separable. We have applied this idea to a linear machine, though we have implemented it somewhat differently so that we can embed it within the tree induction algorithm. Frean observed that there are two kinds of errors that are problematic. First, as shown in the upper left portion of Figure 1, 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, which we achieve by using $c = \frac{\beta}{\beta + k}$, where $k = \left\lceil \frac{(W_j - W_i)^T Y}{2Y^T Y} \right\rceil$, and annealing β during training. The second kind of problematic error occurs when a misclassified instance lies very close to the decision boundary, as shown to the lower right of the decision boundary in the figure. As k approaches 0, c approaches 1 regardless of β. Therefore, to ensure that the linear machine converges, one also needs to anneal the amount of correction c that one will make regardless of k. We accomplish this by annealing c by β, giving the correction coefficient $c = \frac{\beta^2}{\beta + k}$.

Table 1 shows the algorithm for training a thermal linear machine. We have specified that β be 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, but there is no other justification for this annealing rule. 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. Here, we define

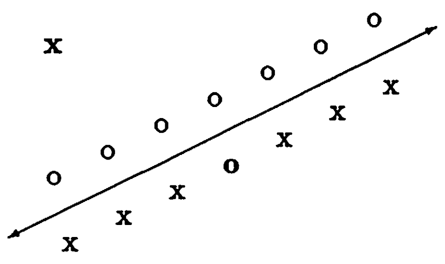

Figure 1. Nonseparable Instance Space

Table 1. Training a Thermal Linear Machine

1. Initialize β to 2.
2. If linear machine is correct for all instances or $\beta < 0.001$, then return.
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, 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 $a\beta - b$. Default values are $a = 0.999$ and $b = 0.0005$.
4. Go to step 2.

the magnitude of a linear machine to be the sum of the magnitudes of its constituent weight vectors. This 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).

2.3 Eliminating Variables

In the interest of producing an accurate and understandable tree that does not evaluate unnecessary variables, one wants to eliminate variables that do not contribute to classification accuracy at a node. Features that are noisy or irrelevant may impair classification, and LMDT finds and eliminates such features. When LMDT detects that a linear machine is near its final set of boundaries, it eliminates the variable that contributes least to discriminating the set of instances at that node, and then continues training the linear machine.

During the process of eliminating variables, the most accurate linear machine with the minimum number of variables is saved. When variable elimination ceases, the test for the decision node is the saved linear machine. There are two cases in which a linear machine based on fewer variables is preferred. The first is when the accuracy of a linear machine based on fewer variables is either higher than the best accuracy observed thus far or if the drop in accuracy is not significantly different than the best accuracy, as measured by a t-test at the .01 level of significance. In this case the linear machine based on fewer variables is saved and if the accuracy is higher, then the system updates its value for the best accuracy observed thus far. In the second case the algorithm is avoiding underfitting the data and will eliminate variables until the number of instances is greater than the capacity of a hyperplane (Duda & Hart, 1973). To this end, if the number of unique instances is not twice the dimensionality of each instance, then the linear machine with fewer variables is preferred.

We measure the contribution of a variable to the ability to discriminate by using a measure of the dispersion of its weights over the set of classes. A variable whose weights are widely dispersed has two desirable characteristics. Firstly, a weight with a large magnitude causes the corresponding variable to make a large contribution to the value of the discriminant function, and hence discriminability. Secondly, a variable whose weights are widely spaced makes different contributions to the value of the discrimination function of each class. Therefore, one would like to eliminate the variable whose weights are of smallest magnitude and are least dispersed. To this end, LMDT's dispersion measure computes for each variable the average squared distance between the weights of each pair of classes and then eliminates the variable that has the smallest dispersion. This measure is analogous to the Euclidean interclass distance measure for estimating error (Kittler, 1986).

A thermal linear machine has converged when the magnitude of each correction to the linear machine is larger than the amount permitted by the thermal training rule for each instance in the training set. However, one does not need to wait until convergence to begin discarding variables. The magnitude of the linear machine asymptotes quickly, and it is at this point that one can make a decision about which variable to discard. To determine this point we use the following heuristic: if the ratio of the magnitude of the entire error correction to the magnitude of the linear machine with the largest magnitude observed thus far is less than α for the last n instances, where n equals the capacity of a hyperplane, then the linear machine is close to converging. Empirical tests show that setting $\alpha = .01$ is effective in reducing total training time without reducing the quality of the learned classifier.

2.4 Relationship to Other Methods for Finding Multivariate Tests

The problem of finding multivariate splits for decision trees has been studied in both pattern recognition and machine learning. Kittler (1986) describes several approaches for linear feature combination. In this framework, LMDT performs a sequential backward selection (SBS) search for a good combination of features. An SBS search is a top down search method that starts with all of the initial features and tries to remove the feature that will cause the smallest decrease in accuracy as measured by a criterion function. A criterion function is a figure of merit reflecting the amount of classification information conveyed by a feature (Kittler, 1986). However, instead of selecting the feature to remove by finding which feature causes the minimum decrease of some criterion function, LMDT removes the feature with the lowest weight dispersion. This reduces the time required to search for a set of features by a factor of n; instead of comparing n linear machines, where n is the number of features in the linear machine, LMDT compares two linear machines.

CART (Breiman, et al. 1984) and PT2 (Utgoff & Brodley, 1990) both perform a SBS search for the best set of features to use as a test in the decision tree. CART searches at each node for the linear discriminant that maximizes the reduction of a discrete impurity measure. Once a set of weights has been found, CART calculates, for each variable, the increase in the node impurity if the variable is omitted. If the smallest increase is less than a preset threshold, then the variable is omitted and the search continues. Note that

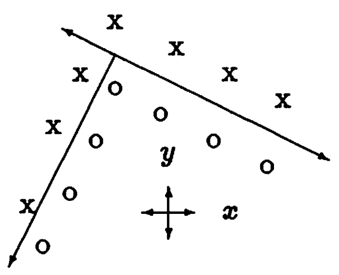

Figure 2. The "L" Problem

after each individual variable is omitted, CART searches for a new threshold, but leaves the weights for the input variables unchanged. After CART determines that further elimination is undesirable, the set of weights is recalculated for the remaining features. One problem with this approach is that after a variable is eliminated the relative importance of the remaining variables may change. Therefore, one should recalculate the weights after each variable is eliminated to avoid eliminating variables erroneously. This problem is most apparent in data sets where the variables are not linearly independent. LMDT and PT2 both recalculate the weights after each elimination. However, PT2's training procedure for finding weights can be computationally prohibitive because if one is using the absolute error correction rule without thermal training, in conjunction with the Pocket Algorithm (Gallant, 1986), it is uncertain how long it will take to find the optimal weight vector or even a good weight vector.

3 An Empirical Comparison of LMDT to C4.5

To examine LMDT's ability to find a good generalization of the examples this section provides an empirical evaluation of the LMDT algorithm. Section 3.1 demonstrates the need for multivariate tests and illustrates LMDT's ability to uncover linear structure in the data. Section 3.2 presents results for a variety of classification tasks and compares the trees produced by LMDT to a univariate decision tree algorithm, C4.5, across the dimensions of size, accuracy and time spent learning.

3.1 Uncovering Structure

Consider the "L" problem, shown in Figure 2. One wants an algorithm to be able to find one segment of the decision boundary at the root node, and the other at the root of a subtree, thereby uncovering the linear structure of the data. The LMDT algorithm does exactly that, due to thermal training of the linear machine at each node. As the linear machine at the root begins to move toward one of the segments, misclassified instances from the other segment have decreasing effect, allowing the linear machine to find one of the segments without being misled by the distant points. A thermal linear machine avoids interaction between different linear functions. In contrast, a least-mean-squares training rule (Duda & Hart, 1973) would place a boundary through the middle of the data, thereby obscuring the structure.

In addition, the "L" problem is characteristic of situations where permitting multivariate splits enables a decision tree algorithm to induce a better generalization than using only univariate splits and is therefore the appropriate bias. A decision tree algorithm that permits

Table 2. Description of the Data Sets

Domain	Number of Classes	Number of Instances	Number of Attributes	Data Type	Missing Data
Clevland	2	303	13	N,S	yes
Glass	6	214	9	N	no
Iris	3	150	4	N	no
Letter Rec.	26	20,000	16	N	no
Pixel Seg.	7	3210	19	N	no
Votes	2	435	16	B	yes

only univariate splits would require a large number of tests to classify the training instances correctly. Indeed, given 40 instances from this domain, C4.5 induces a tree of 21 nodes and 22 leaves. The size of a univariate tree for problems like the "L" problem is dependent on the *grainsize* of the problem; an increase in the number of instances, clustered near a separating hyperplane, increases the number of splits necessary to classify the data correctly. However, the increase in the number of splits does not ensure an increase in accuracy for previously unseen examples.

3.2 Results for a Variety of Domains

We present results for six data sets. These data sets were chosen to represent a mix of symbolic and/or numeric attributes, missing values, binary class tasks and multiclass tasks. A description of each of the data sets can be found in Table 2. The Clevland data set consists of 303 patient diagnoses (presence or absence of heart-disease) described by 13 attributes (Detrano, et al. 1989). The Glass domain involves identifying glass samples taken from the scene of an accident as one of six classes. The Iris data set, Fisher's classic data set, contains 50 examples of three different types of iris plants. One class is linearly separable from the other two, and the latter two are not linearly separable from each other. For the Letter Recognition task the objective is to identify a black-and-white rectangular pixel display as one of the 26 capital letters in the English alphabet (Frey & Slate, 1991). In the pixel segmentation domain the task is to learn to segment an image into seven classes. Each instance is the average of a 3x3 grid of pixels represented by 19 low-level, real-valued image features. In the Votes domain the task is to classify each member of Congress, in 1984, as Republican or Democrat using their votes on 16 key issues.

Various performance measures for each of the tasks are reported in Table 3. Each reported measure is the average of ten runs. To achieve an estimate of the true error rate, for five of the tasks, we performed a ten-fold crossvalidation for each run (Weiss & Kulikowski, 1991). The data were split randomly for each run, with the same split used for both algorithms. For the letter-recognition task, we used the traditional training set of 16,000 instances and the test set of 4,000 instances (Frey & Slate, 1991). The measures reported are: the number of the original input attributes that ever need to be evaluated somewhere in the tree (Unique Attrs.); the number of test nodes in the tree (Nodes); the total number of leaves in the tree (Leaves); the average number of *encoded* variables per linear machine (Avg vars/LM); the number of epochs need to converge to a tree that classifies the training instances correctly (an epoch is equal to the number of instances in the training set.); the number of bits needed

Table 3. Comparison of LMDT to C4.5

Domain	Alg	Unique Attrs.	Nodes	Leaves	Avg. vars/LM	Epochs	Bits	Accuracy
Clevland	LMDT	6.1	2.8	3.8	3.4	78(1180)	282	**77.55**
Clevland	C4.5	10.9	45.8	46.8		576	1273	72.29
Glass	LMDT	7.2	3.6	13.5	4.3	32(206)	974	54.46
Glass	C4.5	8.6	38.9	39.9		469	1330	**67.75**
Iris	LMDT	2.0	1.3	3.3	1.8	5(99)	124	93.93
Iris	C4.5	3.7	8.0	9.0		53	202	93.80
Letter Rec.	LMDT	16.0	516	1673	3.1	33(1859)	109,404	**88.40**
Letter Rec.	C4.5	16.0	1147	1148		780	49,895	86.90
Pixel Seg.	LMDT	15.9	8.3	23.5	5.5	32(1001)	2751	95.10
Pixel Seg.	C4.5	10.5	40.2	41.2		231	2009	96.80
Votes	LMDT	4.6	1.6	2.6	3.4	29(1376)	176	94.75
Votes	C4.5	6.7	7.4	8.4		85	224	**96.30**

to represent the classifier (Bits); and the percentage of the test instances classified correctly (Accuracy). If the difference in the accuracy for the test set is statistically significantly different for the two algorithms, then we highlight this difference by reporting the higher accuracy in bold-face type. The test for significance is a t-test at the .01 level of significance.

The time required to find an LMDT tree is naturally greater than the time required for a C4.5 tree because the hypothesis space of multivariate decision trees is larger than the hypothesis space of univariate decision trees. To compare the difference, we report the number of epochs for each of the algorithms. For the LMDT trees we report both the number of instances used to update the linear machine and the number of instances observed (reported in parentheses). We cannot give a theoretical bound for the time LMDT requires to learn a decision tree as the algorithm for training a linear machine is nondeterministic. For both algorithms we count the number of times each training instance is examined. All of the training instances are examined at the root of the tree, however, at each subtree, the algorithm examines only a portion of the instances. The number reported in Table 3 is the sum of the number of instances observed at each node in the tree divided by the size of the training set. This count is fair because although C4.5, while searching for a test at a subtree, may only examine part of each instance, the same is true of LMDT.

It is not meaningful to compare the size of the two types of trees using measures such as the number of nodes or the number of leaves; the size of an LMDT node can be of greater complexity than a C4.5 node. To compare the size of the trees, we use the Minimum Description Length Principle (MDLP) (Rissanen, 1989), which states that the best "hypothesis" to induce from a data set is the one that minimizes the length of the hypothesis plus the length of the data when coded using the hypothesis to predict the data. Here a hypothesis is a decision tree and the data is the training set. The best hypothesis is the one that can represent the data with the fewest number of bits. To represent the data we must code both the tree and the error vector. The details of the coding procedure are given in Appendix A.

The results in Table 3 show that LMDT finds trees for the Clevland and Letter Recognition tasks that are statistically significantly more accurate than those C4.5 finds, whereas

C4.5 finds more accurate trees for the Glass and Votes tasks. The difference in the accuracies for the Iris and the Pixel Segmentation tasks are not significant. The size of the trees, as measured by the number of bits required to code the tree, is not consistent with the MDLP. We conjecture that this is due to the fact that our codings are not provably optimal.

4 Conclusion and Directions for Future Work

The objective of creating a multivariate decision tree algorithm is to overcome the restriction of univariate trees to tests that represent splits that are orthogonal to the variables' axes. However, the results in Section 3 demonstrate that for some data sets the bias of a univariate decision tree is more appropriate. Although a univariate tree is a special case of a multivariate tree, LMDT's bias for finding such a tree may be inappropriate for some tasks, because it may not find a univariate test when it should. LMDT's variable elimination method is a greedy search procedure and suffers from a problem inherent in any hill-climbing procedure; it can get stuck on local maxima. Therefore, although the hypothesis space LMDT searches includes univariate decision trees, the heuristic nature of LMDT's search may result in selecting a test from an inappropriate part of the hypothesis space.

A solution to this problem would be to determine the appropriate bias dynamically for each test in the tree. The perceptron tree algorithm (Utgoff, 1989) is one example of a system that tries to determine the appropriate representational bias for the instances automatically. Specifically, the algorithm first tries to fit a linear threshold unit(LTU) to the space of instances. If the space is not linearly separable, then the bias of an LTU is inappropriate and the system searches for the best univariate test. However, for some instance spaces, the best test may be based on a subset of the variables. From our results we conclude that a multivariate decsion tree algorithm should employ a dynamic control strategy for finding the appropriate representational bias for each test in the decision tree. Specifically, rather than search the space of multivariate tests using a fixed bias (like LMDT), such a system would have the capability to focus its search using heuristic measures of the learning process (Brodley, 1992).

The problem of bias is not restricted to decision trees. It is a well known problem that the ability of a chosen algorithm to induce a good generalization depends on how well the hypothesis space underlying the learning algorithm and the bias for searching that space fit the given task. Given that different algorithms search different hypothesis spaces, it is not surprising then that one algorithm finds better hypotheses than others for *some*, but not all tasks. Given a task for which there is no *a priori* knowledge as to what the appropriate hypothesis space should be, a learning algorithm should itself determine what is the appropriate bias.

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 segmentation data was donated by B. Draper and R. Collins from the Visions Lab at UMASS. The Clevland, Glass, Iris, Letter Recognition and Votes1 data sets are from the UCI machine learning database. J. Ross Quinlan provided us with a copy of C4.5. Jeffery Clouse, Jamie Callan, Tom Fawcett, and Margie Connell provided helpful comments.

References

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees.* Belmont, CA: Wadsworth International Group.

Brodley, C. E. (1992). *Dynamic automatic model selection,* (Coins Technical Report 92), Amherst, MA: University of Massachusetts, Department of Computer and Information Science.

Cover, T. M. (1973). Enumerative source coding. *IEEE Transactions on Information Theory, IT-19,* 73-77.

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 Hournal 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.

Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory, IT-21,* 194-203.

Frean, M. (1990). *Small nets and short paths: Optimising neural computation.* Doctoral dissertation, Center for Cognitive Science, University of Edinburgh.

Frey, P. W., & Slate, D. J. (1991). Letter recognition using holland-style adaptive classifiers. *Machine Learning, 6,* 161-182.

Gallant, S. I. (1986). Optimal linear discriminants. *Proceedings of the International Conference on Pattern Recognition* (pp. 849-852). IEEE Computer Society Press.

Hunt, E., Marin, J,, & Stone, P. (1966). *Experiments in induction.* Academic Press Inc..

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

Nilsson, N. J. (1965). *Learning machines.* New York: McGraw-Hill.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1,* 81-106.

Quinlan, J. R. (1987). Decision trees as probabilistic classifiers. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 31-37). Irvine, CA: Morgan Kaufmann.

Rissanen, J. (1989). *Stochastic complexity in statistical inquiry.* New Jersey: World Scientific.

Utgoff, P. E. (1989). Perceptron trees: A case study in hybrid concept representations. *Connection Science, 1,* 377-391.

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.

Weiss, S. M., & Kulikowski, C. S. (1991). *Computer systems that learn.* Palo Alto: Morgan Kaufmann.

Appendix A

To determine the number of bits required to code a consistent hypothesis we need to code both the hypothesis induced by the learning algorithm and the errors that the hypothesis makes. To code the error-list we use Cover's (1973) enumerative encoding scheme. To code each type of decision tree we use a recursive, top-down depth first procedure making the assumption that the receiver of the code knows the order of the attributes. For the following discussion let k equal the number of classes, let n equal the number of attributes and let $V(a_i)$ equal the number of distinct values for attribute a_i. For both algorithms we need one bit to indicate if the node is a leaf or a test. To code a leaf we need $log_2(k)$ bits.

Coding a C4.5 Test: We use one bit to indicate if the attribute tested is discrete or continuous and $log_2(n)$ bits to specify the attribute. If the tested attribute is discrete, then we code the value of each branch, which takes $log_2(V(a_i))$ bits. Note that when coding the current node's children we need only use $log_2(n-1)$ bits, to specify an attribute, because a discrete attribute is never re-tested in the subtree. We code the value of a continuous test as a real number. We assume that the left branch corresponds to *less-than* and that the right branch corresponds *greater-than-or-equal-to* range value.

Coding an LMDT Test: If the node is a linear machine, then we need k bits to specify which classes have a linear discriminant at this node in the tree. We code each linear discriminant as a vector of real numbers To represent an attribute that has been eliminated we code its weight as 0.

Coding Real Numbers: We first reduce the precision necessary to retain the same accuracy. We code the integer part of the number separately from the fractional part, using Elias' (1975) asymptotically optimal prefix code.