Linear Machine Decision Trees

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

COINS Technical Report 91-10
January 1991

Abstract

This article presents an algorithm for inducing multiclass decision trees with multivariate tests at internal decision nodes. Each test is constructed by training a linear machine and eliminating variables in a controlled manner. Empirical results demonstrate that the algorithm builds small accurate trees across a variety of tasks.

1 Introduction

One of the fundamental research problems in machine learning is how to learn from examples. From a sequence or set of training examples, each labeled with its correct class name, a machine learns by forming or selecting a generalization of the training examples. This process, also known as supervised learning, is useful for real classification tasks, e.g. disease diagnosis, and for problem solving tasks in which control decisions depend on classification, e.g. rule applicability. The ability to generalize is fundamental to intelligence because it allows one to reason in accordance with predictions that are often correct. This article focuses on the problem of inducing a classifier in the form of a decision tree, using a combination of methods from connectionist and decision tree approaches.

2 The Tree Induction Algorithm

The main objective in building a decision tree is to obtain an accurate classifier for instances in the domain. A decision tree is an appropriate choice when the underlying class conditional probabilities are unknown and when the class distributions are multimodal (Duda & Fossum, 1966). A secondary objective in building a decision tree is to obtain a decision procedure that is intelligible to a human, so that a person can become better at classifying instances by emulating the decision procedure. Moreover, in domains in which the cost of misclassification is high, an intelligible tree allows a person to verify that the decision procedure is indeed correct.

A tree induction algorithm is useful to the extent that it captures the structure of the underlying data (Breiman, Friedman, Olshen & Stone, 1984). Assuming that class membership depends on some combination of properties of an instance, one would like the induction algorithm to identify that combination. Accordingly, the algorithm must be able to represent such a combination and find it in an acceptable amount of time. One would like an algorithm that handles multiple classes, considers both numerical and logical combinations of variables, allows both discrete and continuous variables, and handles instances that are described imprecisely or labeled incorrectly. The rest of this section describes the LMDT algorithm, which has been designed to meet these objectives.

Table 1: Top Level of the LMDT Algorithm.

1. If all the instances are from a single class, then set TREE to be a leaf node with the class name of the single class, return.
2. Otherwise, set TREE to be a decision node containing a test constructed by training a linear machine.
3. If the test partitions the instances into two or more subsets, then for each subset build a subtree recursively, return.
4. Otherwise, set TREE to be a leaf node with the class name of the most frequently occurring class, return.

2.1 Top Level of the Algorithm

The LMDT algorithm builds a decision tree in the well known top-down manner, as indicated in Table 1. However, instead of selecting a univariate test for a decision node based on a heuristic measure, such as information gain (Lewis, 1962), the LMDT algorithm trains a linear machine, 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 the instance. The class name is the result of the linear machine test, hence there will be one branch for each possible class at the node. A linear machine decision tree is a hybrid representation, learned by a combination of two methods, one for learning decision trees and the other for learning linear discriminant functions.

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, also known as a pattern vector, consisting of a constant threshold value 1 and the numerically encoded features by which the instance is described. Then each discriminant function $g_i(\mathbf{Y})$ has the form $\mathbf{W}_i^T\mathbf{Y}$, where $\mathbf{W}$ 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 $(\forall i, i \neq j)\ g_i(\mathbf{Y}) > g_j(\mathbf{Y})$. If there is no unique maximum, then the classification is undefined.

In general, to train a linear machine, one adjusts the weight vectors $\mathbf{W}$

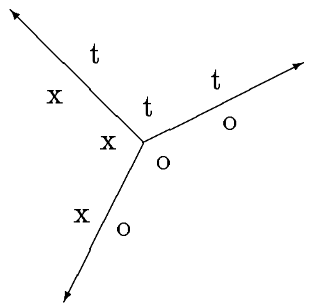

Figure 1: A Multiway Partition via a Linear Machine.

of the discriminant functions g in response to any instance that the linear machine would misclassify. This is done by increasing the weights of $\mathbf{W}_i$, where i is the class to which the instance belongs, and decreasing the weights of $\mathbf{W}_j$, where j is the class to which the linear machine erroneously classifies the instance. The specific error correction rule for the LMDT algorithm is described below in Section 2.3. If the instances are linearly separable by a linear machine, then the above training procedure, with a suitable error correction rule as described below, will find a solution machine in a finite number of steps (Duda & Hart, 1973).

A multivariate test, here represented as a linear machine, makes it possible to represent decision boundaries that are not orthogonal to the axes (Breiman, Friedman, Olshen & Stone, 1984; Utgoff & Brodley, 1990). Such a capability is essential if one wants to capture and represent a concept that is best described in terms of a function of two or more variables. For example, one would want to express the concept of "an overweight person" in terms of a relationship between height and weight.

A linear machine partitions the instances into convex regions of the hyperspace, providing a multiway split, as illustrated in Figure 1. In contrast, a single linear discriminant can only represent a binary split. In the figure, one can see that no binary split separates one class from all the others. Multiple binary splits could be employed, as in a decision tree, but the splits would be inaccurate and the structure of the underlying data would be obscured.

2.2 Encoding the Instances

In order to construct a linear machine test, each instance must be represented as a vector $\mathbf{Y}$ consisting of a constant threshold value of 1, and numerically encoded features that describe the instance. The instances need to be encoded numerically, and the LMDT algorithm does this automatically at each node for all variables. All encoding information is retained in the tree for the purpose of classifying instances. 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.

When encoding a nonnumeric input variable, one needs to be careful not to impose an order on the values of the input variable. For a two-valued variable, one of the values is encoded as 1 and the other as -1. For a many-valued variable, each variable-value pair is encoded as a propositional variable, which is TRUE if and only if the variable has taken on the particular value in the instance (Hampson & Volper, 1986). The value of each propositional variable is then encoded as 1 or -1 as before. This encoding scheme makes it possible to describe instances by any mix of variable types, numeric or symbolic. There is an additional advantage to this encoding scheme; mapping many-valued variables to two-valued variables has been observed to produce trees with higher classification accuracy (Cheng, Fayyad, Irani & Qian, 1988; Mooney, Shavlik, Towell & Gove, 1989).

The encoded variables are also normalized automatically at each node as part of the encoding process. The scaling is accomplished for each encoded variable by mapping it to the mean of the observed values plus one standard deviation to 1 and the mean minus one standard deviation to -1. This is done so that weight adjustments are equally important during adjustment of a $\mathbf{W}_i$ and so that it is meaningful to gauge the relative importance of the encoded variables by the magnitudes of the corresponding weights. The ability to assess relative importance of the variables is essential for the variable elimination mechanism described below in Section 2.4.

A final consideration in encoding the input variables is how to fill in missing values during training or classification. The approach adopted here for handling a missing value is to map it to 0, which corresponds to the sample mean of the corresponding encoded variables. By using the sample

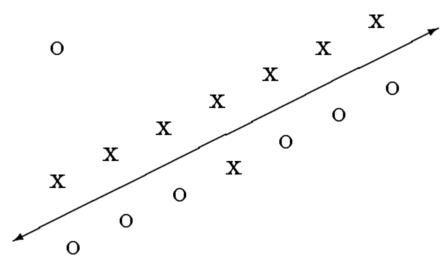

Figure 2: Nonseparable Instance Space.

means for the encoded variables, one is estimating the expected location of the instance in the encoded Euclidean n-space.

2.3 Training a Thermal Linear Machine

This section describes how a linear machine is trained, leaving the discussion of how to eliminate variables for the following section. If the training instances are linearly separable, then one can train on the instances repeatedly until the linear machine partitions the instances into separate convex regions. For each instance, the weight vectors of the linear machine are adjusted only if the instance would be misclassified by the linear machine. One well known method is the absolute error correction rule (Duda & Fossum, 1966), which adjusts $\mathbf{W}_i$, where i is the class to which the instance actually belongs, and $\mathbf{W}_j$, where j is the class to which the linear machine incorrectly assign the instance. The correction is accomplished by

$$\mathbf{W}_i \leftarrow \mathbf{W}_i + c\mathbf{Y}$$

$$\mathbf{W}_j \leftarrow \mathbf{W}_j - c\mathbf{Y}$$

with c computed to be a sufficient amount of correction to cause the linear machine to become correct for the misclassified instance.

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. One needs an error correction rule that gives stable behavior

Table 2: Training a Thermal Linear Machine.

1. Initialize β to 2.
2. If $\beta < 0.001$ or P(LM is correct on all instances at the node) > 0.99 then return.
3. Otherwise, select a training instance $\mathbf{Y}$ at random. If it would be misclassified by the linear machine and $k < \beta$, then
   (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 set β to $a\beta - b$. Default values are $a = 0.995$ and $b = 0.0005$.
4. Go to step 2.

even when the instances are not linearly separable. Recently, Frean (1990) has developed the notion of a "thermal perceptron", which addresses this problem, and is realized by a particular error correction rule. 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 2, if an instance is far from the decision boundary, and would be misclassified, then the decision boundary would need 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 probably counterproductive. One should pay less attention to large errors than to small errors, and one can accomplish this by

$$c = \frac{1}{1+k}, \text{ where } k = \frac{(\mathbf{W}_j - \mathbf{W}_i)^T \mathbf{Y}}{2\mathbf{Y}^T\mathbf{Y}}.$$

The expression for k is the absolute error correction needed to adjust the weight vectors so that a misclassified instance becomes correctly classified by the linear machine (Duda & Fossum, 1966). Although this will discount misclassified instances that are far from a decision boundary, it does not solve the problem completely. To achieve stability, Frean calls for paying decreasing attention to large errors, which we achieve by using $c = \frac{\beta}{\beta+k}$, and annealing (reducing) 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, so one also needs to anneal the amount of correction c that one will make regardless of k. We accomplish this by multiplying c by β because it is already annealing, giving $c = \frac{\beta^2}{\beta+k}$.

Table 2 shows the algorithm for training a thermal linear machine. 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. We define 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 much nearer to its final location (Duda & Hart, 1973). The algorithm does not anneal β when the magnitude of the linear machine is decreasing repeatedly, which can happen after a variable has been eliminated.

2.4 Variable Elimination

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. Table 3 shows the algorithm for simultaneously training a linear machine and eliminating variables from the machine. When the decision boundaries are found by thermal training, the algorithm eliminates the variable that contributes least to discriminating the set of instances at that node, and then resumes training the linear machine thermally, with β reinitialized. This eliminate-train cycle repeats until it appears that further variable elimination will only decrease classification accuracy at the node.

During the process of eliminating variables, the most accurate linear machine with the minimum number of variables is saved, and when variable elimination ceases, the test for the decision node is the saved linear machine. There are three cases in which an LM based on fewer variables is preferred.

Table 3: Training a Linear Machine with Variable Elimination.

1. Encode each instance numerically.

2. Initialize *bestaccuracy* to 0, and the weights $\mathbf{W}$ for each of the R discriminants of the linear machine to 0.

3. Train the linear machine thermally, as described in Table 2

4. Set *accuracy* to the classification accuracy of the linear machine on the training instances.

5. If $accuracy \geq bestaccuracy$ or $number\text{-}of\text{-}instances \leq 2 * number\text{-}of\text{-}variables$ then set *bestaccuracy* to *accuracy*.

6. If $accuracy \geq bestaccuracy$ or the accuracy is not significantly worse than that of the best linear machine saved, then save copy of linear machine.

7. If $accuracy \geq bestaccuracy - \delta$ and the linear machine is based on two or more encoded variables, then eliminate a least discriminating variable, go to step 3.

8. Final linear machine is the one saved by step 6, return.

The first is when the accuracy of an LM based on fewer variables is higher than the best accuracy observed thus far. The second is when the accuracy of the LM drops when a variable is eliminated. A t-test with $\alpha = .01$, is used to measure whether the difference in the mean classification accuracies of the two LMs is significant or due to chance. The third case occurs when there are too few instances for the number of variables (Duda & Hart, 1973). The δ parameter is included for the sake of efficiency, and its default value is 0.10. The algorithm can stop trying to eliminate variables if the chance of finding an LM based on fewer variables with higher accuracy is remote. Note that when the accuracy of an LM increases when a variable is eliminated, it is not clear how much of the increase should be attributed to additional training time.

We measure the contribution of a variable to the ability to discriminate as the dispersion of the weights for each of the classes. There are two desirable characteristics of a weight with a large dispersion. First, a weight with a large magnitude causes the corresponding variable to have a large effect on the value of the discrimination function and hence discriminability. Second, a variable whose weights are spaced far apart makes different contributions to the value of the discrimination function of each class. Therefore, one would like to eliminate the variable whose weights have small magnitudes and are evenly spaced. To this end, our dispersion measure computes, for each variable, the average squared distance between the weights of each pair of classes. The variable with the smallest dispersion is eliminated. This basis for comparing the discriminating ability of the variables depends on the variables having been normalized, as discussed above in Section 2.2.

2.5 Discriminability of a Linear Machine

It happens that some unusual distributions of instances lead to a linear machine that places all the instances in just one region, e.g. an instance from one class located near the center of a cloud of instances from another class. The LMDT algorithm does not attempt to partition such distributions, on the assumption that such poorly located instances are indicative of noise or poor representation. The algorithm simply replaces the nondiscriminating decision node with a leaf node indicating the most frequently occurring class, as per step 4 of Table 1. We have tried several methods for forcing the algorithm to find a discriminating linear machine, but without any apparent

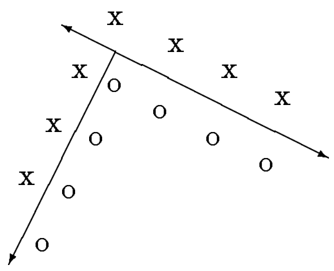

Figure 3: The "L" Problem.

benefit in classification performance. One such method is to select one instance from each class at random, and train a linear machine to discriminate just those instances.

3 Illustrations

This section illustrates that the LMDT algorithm is capable of uncovering the underlying structure of the data, and that it finds good tests for decision nodes. The algorithm finds small trees across a variety of classification tasks.

3.1 Uncovering Structure

In order to see that the algorithm can uncover underlying structure, consider the "L" problem shown in Figure 3. One would want 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. 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. It may be that such an artificial boundary is unlikely in practice, but it is essential that an algorithm be able to find segments of the decision boundary wherever they occur.

Table 4: LMDT on a Variety of Tasks.

Domain	R	Invars	LMs	Vars/LM	Totvars	%Train	%Test
DNF	2	5	2	2.5	5	100.00	
chess	2	39	7	11.2	37	99.46	
LED-10%	10	7	8.6	4.5	7	79.64	70.20
Soybean	15	35	4.8	8.3	23	97.59	84.88
Segment	7	17	1	5.8	5.8	98.86	94.25

3.2 Results for a Variety of Domains

Table 4 show various performance measures for the LMDT algorithm on a variety of tasks. Pessimistic pruning (Quinlan, 1987) is used to avoid overfitting. Each measure is an average for five runs. Column "Invars" shows the number of input variables. Column "Vars/LM" indicates the average number of variables per linear machine, and column "Totvars" lists the number of unique input variables tested somewhere in the tree.

The DNF task illustrates the ability of LMDT to find multivariate splits and to eliminate variables in order to capture the underlying structure of the instance space. Pagallo (1989) used this task to illustrate the ability of FRINGE to find multivariate tests at a node, though by a much different mechanism from that of LMDT. The concept to be learned is the Boolean function $ab \vee c\bar{d}e$. The tree found by LMDT has two tests and three leaves and is logically equivalent to that found by FRINGE. The root node has an LM based on variables a and b and the second node has an LM based on c, d and e.

The chess task (Quinlan, 1983) demonstrates that LMDT finds small trees. The tree found by LMDT contains 7 linear machines, each with an average of 11.2 variables. The smallest tree produced by an ID3 variant (Utgoff, 1989) contains 62 univariate tests. A linear machine is more complex than a univariate test, so a strict comparison is unfair.

The LED (Breiman, Friedman, Olshen & Stone, 1984) data set shows LMDT's performance on noisy data sets. The domain consists of seven Boolean attributes and ten classes, one for each of the decimal digits. The seven attributes are the seven light emitting diodes of a LED display. Each attribute has a 10% percent probability of having its value inverted. Breiman et al. have shown that for a noise rate of 10% the optimal Bayes classification rate is 74%.

The soybean task illustrates LMDT's faculty to handle multivalued attributes. The soybean data, first introduced by Michalski (Michalski & Chilausky, 1980), consists of 35 symbolic attributes with 4% of the values missing.

The segmentation task illustrates the ability of LMDT to handle real-valued variables and shows how variable elimination can reduce the number of variables tested. In the segmentation domain, the task is to learn to segment an image into one of seven classes. Each instance is a pixel described by low level image features. LMDT reduced the number of input variables that ever need to be evaluated from 17 to 5. The values of the variables in this domain are expensive to generate. Therefore, reducing the number that need to be evaluated reduces the amount of processing required to obtain a description of a pixel.

4 Discussion

The LMDT algorithm handles multiclass problems, finds multivariate tests, and eliminates variables in a justified manner. Through thermal training of a linear machine at a node, the algorithm is able to find a good decision boundary without being misled by misclassified instances. Through automatic encoding of numeric and symbolic variables, the algorithm is applicable to a wide variety of classification learning tasks.

Two problems remain that we are in the process of addressing. First, there are several fast forms of conjugate gradient weight adjustment (Møller, 1990), and we would like to incorporate one. These methods are typically one to two orders of magnitude faster than gradient descent methods. Typically, conjugate gradient weight adjustment methods seek to minimize mean-squared error, so we would need to substitute a different error function to be minimized, retaining the ability of the thermal linear machine to uncover the structure of the data.

Finally, trees with linear machine tests can be difficult to understand because a linear machine is itself difficult to understand. A linear machine can be converted to a set of classification rules, which are more palatable. For example, for a three-class linear machine, one would produce three rules, one for each class. The rule for class 0 would be of the form: if $g_0(x) > g_1(x) \wedge g_0(x) > g_2(x)$ then x belongs to class 0. Standard arithmetic transformations may allow further simplification of such a rule. Of course, the variable elimination mechanism aids in simplifying the LM itself, which will lead to simpler rules. In general, multivariate tests are essential, so we will be seeking methods for making them understandable, rather than limiting the algorithm to weaker kinds of tests. A method similar to that employed by MACIE (Gallant, 1988) may be applicable for explaining decisions made by an individual linear machine.

References

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

Cheng, J., Fayyad, U. M., Irani, K. B., & Qian, Z. (1988). Improved decision trees: A generalized version of ID3. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 100-106). Ann Arbor, MI: Morgan Kaufman.

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.

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

Gallant, S. I. (1988). Connectionist expert systems. *Communications of the ACM, 31*, 152-169.

Hampson, S. E., & Volper, D. J. (1986). Linear function neurons: Structure and training. *Biological Cybernetics, 53*, 203-217.

Lewis, P. M. (1962). The characteristic selection problem in recognition systems. *IRE Transactions on Information Theory, IT-8*, 171-178.

Michalski, R. S., & Chilausky, R. L. (1980). Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Policy Analysis and Information Systems, 4*, 125-160.

Møller, M. F. (1990). *A scaled conjugate gradient algorithm for fast supervised learning*, (DAIMI PB - 339), Denmark: Aarhus University, Computer Science Department.

Mooney, R., Shavlik, J., Towell, G., & Gove, A. (1989). An experimental comparison of symbolic and connectionist learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 775-780). Detroit, Michigan: Morgan Kaufmann.

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

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, Carbonell & Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-machine Studies, 27*, 221-234.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning, 4*, 161-186.

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.