Goal-Directed Classification using Linear Machine Decision Trees *

Bruce A. Draper Carla E. Brodley Paul E. Utgoff
Department of Computer Science
University of Massachusetts
Amherst, MA., USA. 01003
bdraper@cs.umass.edu

September 17, 1993

Abstract

Recent work in feature-based classification has focused on non-parametric techniques that can classify instances even when the underlying feature distributions are unknown. The inference algorithms for training these techniques, however, are designed to maximize the accuracy of the classifier, with all errors weighted equally. In many applications, certain errors are far more costly than others, and the need arises for non-parametric classification techniques that can be trained to optimize task-specific cost functions. This paper reviews the Linear Machine Decision Tree (LMDT) algorithm for inducing multivariate decision trees, and shows how LMDT can be altered to induce decision trees that minimize arbitrary misclassification cost functions (MCFs). Demonstrations of pixel classification in outdoor scenes show how MCFs can optimize the performance of embedded classifiers within the context of larger image understanding systems.

Keywords: Decision Trees, Non-Parametric Classification, Pattern Recognition, Object Recognition, Computer Vision, Machine Learning.

1 Introduction

Feature-based classification – in which instances are assigned to classes based on vectors of feature values – is a recurring subproblem in systems that analyze image data. In recent

*Bruce Draper's work was supported by Rome Laboratory of the Air Force Systems Command under contract F30602-91-C-0037. Carla Brodley and Paul Utgoff's work was 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.

years, many researchers have eschewed traditional parametric classification techniques (such as minimum distance classification) in favor of newer, non-parametric techniques, particularly decision trees and neural nets. These techniques learn to classify instances by mapping points in feature space onto categories without explicitly characterizing the data in terms of parameterized distributions. As a result, they avoid making assumptions about the data that, when violated, can cause parametric techniques to fail.

Decision trees classify instances by recursively subdividing feature space with hyperplanes until each subdivided region contains instances of a single type. The best known algorithm for inducing decision trees from training instances is Quinlan's ID3 [9], which uses an information-theoretic measure to choose the best hyperplane at each recursive step. Typically, the decision tree is expanded until every training instance is correctly classified, and then the tree is pruned to avoid overfitting to the training data.

One disadvantage of ID3-style decision trees is that each test is univariate. In other words, when the input features are numeric, each test is of the form $x > a$, where a is a value in the observed range of feature x. Because these tests are based on a single input variable, univariate trees can only divide feature space orthogonally to a feature's axis. This introduces a bias that may be inappropriate for problems with linearly related features[1, 11].

Utgoff and Brodley [11] overcome this problem by using the perceptron learning rule to induce decision trees in which the tests are linear combinations of features. Linear machine decision trees generalize the two-category multivariate splits permitted by the perceptron training rule to n-category multivariate splits by the use of linear machines [2]. Because the linear machine decision tree algorithm (LMDT) is unknown to most vision researchers, we begin by reviewing this powerful yet simple classification technique.

The focus of this paper, however, is on the development of *goal-directed* classifiers. In computer vision, classification is often an intermediate processing step rather than a final goal. Perhaps the best known example of this is when color and texture based classifiers are used as *focus of attention* (FOA) mechanisms for triggering the application of computationally expensive matching algorithms. When classifiers are used to focus attention, the goal of the induction algorithm should be to maximize the performance of the overall (i.e. classification plus matching) system, not just the performance of the classifier.

As an example, consider the case where a classifier is used to focus the attention of a matching algorithm. Such a system "finds" an object when the classifier labels part of the image as the object, thus generating a hypothesis, and the matching algorithm matches the model in the corresponding data, verifying the hypothesis. The success rate of such a system depends on the likelihood that the classifier will positively label the object instance, as well as the accuracy of the matcher[1] The computational cost of the system depends primarily on the frequency with which the matcher is invoked, which in turn depends on the density of objects in the data and the likelihood of the classifier generating a "false alarm". The robustness of the system is therefore primarily influenced by the false negative rate of the classifier, while the computational cost is a function of the false positive rate. The best classifier therefore depends on an application-specific utility function measuring the relative value of accuracy vs. computational cost.

In goal-directed classification, the penalties for confusing two instance labels is determined by a *misclassification cost function* (MCF), and the goal of the training algorithm is to minimize the total misclassification cost. Although Bayesian techniques have long been able to minimize arbitrary MCFs[5], non-parametric techniques have not previously been able to do so. Section 3 shows how the non-parametric LMDT algorithm can be modified to reduce arbitrary misclassification cost functions, while Section 4 demonstrates empirically how MCFs alter the performance of systems with embedded classifiers.

[1] For the purposes of this discussion, we assume that the matcher is highly accurate, but much more expensive than the classifier. As a result, the possibility that the matcher will verify a false hypothesis can be ignored, and the cost of the system is dominated by the matching algorithm.

2 Linear Machine Decision Trees

As the name implies, linear machine decision trees are a marriage of two well-known classification techniques, *linear machines* [8, 5] and *decision trees* [1, 9]. The LMDT algorithm builds a multivariate decision tree from the top down. LMDT trains a linear machine to classify the initial set of training instances by partitioning the feature space into R regions, one for each of the R observed classes. If the instances in a region are from one class, the region is assigned that class label. Otherwise the algorithm is applied recursively to the region. The result is a decision tree with a linear machine at each internal node and a class label at every leaf. To classify an instance, one follows the branch indicated by the linear machine, starting at the root of the tree and working toward the leaves. When a leaf node is reached, the instance is assigned the corresponding label.

2.1 Training a Linear Machine

As per Nilsson [8], a *linear machine* is a set of R linear discriminant functions that are used collectively to assign an instance to one of R classes. Let Y be an instance description (a feature vector) consisting of a constant threshold value 1 and a set of numeric features, which describe the instance[2]. Then each discriminant function $g_i(Y)$ has the form $W_i^T Y$, where W_i is a vector of adjustable coefficients, also known as weights. A linear machine infers instance Y to belong to class i if and only if $(\forall j, i \neq j)\ g_i(Y) > g_j(Y)$.

The LMDT algorithm uses the thermal training procedure [6, 2] to find the coefficients of linear machines. Unlike the absolute error correction rule, thermal training ensures convergence to a set of coefficients regardless of whether or not the data is linearly seperable. During training, decreasing attention is paid to large errors by using the correction $c = \frac{\beta}{\beta+k}$,

[2] Throughout this discussion we will assume that all features are in standard normal form, i.e. zero men and unit standard deviation. See Brodley and Utgoff [2] for a discussion of normalization and missing feature values.

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 Y that would be misclassified by the linear machine and for which $k < \beta$, immediately
   (a) Compute correction c, and update W_i and W_j.
   (b) If the magnitude of the linear machine decreased on this adjustment, but increased on the previous adjustment, then anneal β to $a\beta - b$. ($a = 0.999$ and $b = 0.0005$)
4. Go to step 2.

where $k = \left\lceil \frac{(W_j - W_i)^T Y}{2Y^T Y} \right\rceil$, and annealing β during training[3]. This prevents large adjustments in the coefficients in response to a single value once the coefficients have begun to stabilize. In addition, to ensure that the linear machine converges even when misclassified instances lie close to the decision boundary, the thermal training procedure also anneals the correction factor c by β, giving the correction coefficient $c = \frac{\beta^2}{\beta + k}$.

Table 1 shows the algorithm for training a thermal linear machine. To allow the algorithm to spend more time training with small values of β when it is refining the location of the decision boundary, β is reduced geometrically by rate a, and arithmetically by constant b. 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, the magnitude of a linear machine is defined as the sum of the magnitudes of its constituent weight vectors. This criterion for reducing β was selected because the magnitude of a linear machine initially increases rapidly during training, and then stabilizes when the decision boundary is near its final location [5].

[3] k is the well-known absolute error correction coefficient [4]

2.2 Eliminating features

In order to produce accurate and understandable trees that do not evaluate unnecessary features, one wants to eliminate features that do not contribute to classification accuracy at a node. Noisy or irrelevant features 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 feature that contributes least to discriminating the set of instances at that node, and then continues training the linear machine. Elimination proceeds until only one feature remains or further elimination will significantly reduce the accuracy of the linear machine.

During feature elimination, the best linear machine with the minimum number of features is saved. When feature elimination ceases, the test for the decision node is the saved linear machine. The best linear machine is the one with the fewest features whose accuracy is not statistically significant from the highest accuracy observed during elimination, or the linear machine underfits the data[4]

A feature's discriminability is measured by the dispersion of its weights over the set of classes. A feature that has not been eliminated from a linear machine has a weight in each class's discriminant function. A feature whose weights are widely dispersed has two desirable characteristics: a weight with a large magnitude causes the corresponding feature to make a large contribution to the value of the function, and hence discriminability, and a feature whose weights are widely spaced, across the 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, LMDT computes, for each feature, the sum of the squared differences in the weights for each pair of classes; for each feature, F, LMDT computes

[4]If there are fewer instances than the capacity of a hyperplane (twice the dimensionality of the feature vector), then there is insufficient data to pick the best hyperplane orientation, and the linear machine is said to underfit the data. [5]

$dispersion = \sum_{i,j=1}^{nclasses} (weight_{F,i} - weight_{F,j})^2$. The feature with the smallest dispersion is then eliminated.

The weights of a thermal linear machine have 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 features. The magnitude of the linear machine asymptotes quickly, and it is at this point that one can make a decision about which features to discard. When to begin feature elimination is determined by a heuristic: if for the past n instances, the ratio of the magnitude of the linear machine to the maximum magnitude observed thus far is less than α, then the linear machine is close to converging, where n is the capacity of a hyperplane and α set to be 1%. Empirical tests across a variety of data sets have shown that setting $\alpha = 1\%$ is effective in reducing total training time without reducing the quality of the learned classifier.

2.3 Error-Reduction Pruning

Overfitting becomes a problem in domains (such as vision) that contain *noisy* instances, i.e. instances for which the class label or some number of the feature values are incorrect. Overfitting occurs when the learning algorithm induces a classifier that classifies all instances in the training set, including the noisy ones, correctly. Such a classifier will perform poorly for previously unseen instances. To avoid overfitting, the LMDT classifier is pruned back to reduce the estimated classification error, as computed for an independent set of instances [1, 10]. A subtree is pruned if its error rate is higher than the error rate that would occur if the subtree were replaced with a leaf containing the most frequently observed class at the root of the subtree.

3 Goal-Oriented Classification

As discussed earlier, a goal-oriented classification system is one in which the performance of the classifier is tailored to maximize overall system performance rather than classification accuracy. More formally, we define a *misclassification cost function* (MCF) as

$$MCF : L \times L \rightarrow Z^+$$

where L is the set of possible labels, such that $MCF(l,l) = 0$ for all $l \in L$. A goal-oriented classifier is one that minimizes the total cost $\sum MCF(p,t)$ of its confusions, where p is the classifier's predicted label for an instance and t is the instance's "true" label.

Given this definition, a goal-oriented classifier can be trained to maximize classification accuracy by assigning an equal weight to all confusions in the MCF. Conversely, a classifier can be trained as an FOA mechanism for instances of type t by making $MCF(q,t) > MCF(t,q)\ \forall q \neq t$. In the extreme case, $MCF(t,q) = 0$, implying that there is no cost whatsoever for a false positive. Such an MCF will lead a classifier to assign all pixels the label t, thereby minimizing the number of false negatives and the total cost. To avoid this, a small cost should be associated with false positives, where the ratio of costs between false negatives and false positives is an explicit statement of the system's reliability/cost tradeoff.

3.1 Inducing a Minimum-Cost Classifier

We have altered the LMDT algorithm to form a classifier with the explicit goal of reducing the total misclassification cost of the errors. Section 3.1.1 describes the changes made to the tree building algorithm and Section 3.1.2 describes the change made to the pruning algorithm.

3.1.1 Building a Tree with the Goal of Reducing Misclassification Cost

Table 2: Misclassification-Cost Training

1. Initialize the training $proportion_i = 1.0$ for each class, i;

2. If linear machine has converged (by the thermal training rule) then return.

3. Otherwise, train linear machine thermally in the proportions specified by the training *proportion* for n objects, where $n = \sum_{i=1}^{nclasses} proportion_i$. Specifically, for each observed object Y with class label i:

   - Increment $observed_i$.
   - If Y would be misclassified by the linear machine as class p, then update $cost_i = cost_i + MCF(p, i)$.

4. For each class, i, recalculate $proportion_i = \frac{cost_i / observed_i}{\sum_{j=1}^{nclasses} (cost_j / observed_j)}$

5. Go to step 2.

We changed LMDT's weight learning algorithm and feature elimination strategy to meet the new goal of reducing misclassification cost. LMDT's original method for training the weights of a linear machine sought to reduce overall error by randomly sampling instances to update the weights of the linear machine. In the modified version of LMDT, the training routine samples objects based on the cost of misclassifications that the *current* classifier makes. This focuses training on objects proportional to their contribution to the current total misclassification cost, as specified by the MCF.

The misclassification cost training routine is shown in Table 2. Specifically, the training proportions are initialized to 1 (Step 1), ensuring that at the start of training all classes are sampled evenly. Using the training proportions, objects are sampled from each class in proportion to the current misclassification cost for the class (Step 3). Each class's cost, which is the sum of all misclassification error costs made in the past, is updated. The error cost of each class is updated each time the linear machine misclassifies an instance, while the number of observed objects for the class is incremented each time an instance from the class is observed. After LMDT trains using instances in the specified proportions, it uses the updated error costs to recalculate the proportions (Step 4). To compute the sample proportion for class i the error cost rate of class i is weighted by the total error cost, which is the sum of the error cost rate over all observed classes. LMDT now uses the updated sampling proportions and continues training (Step 5).

The second change to the LMDT algorithm alters the feature elimination method to take into consideration misclassification costs. The original LMDT algorithm preferred linear machines with fewer features over more complex ones, unless the simpler machine was significantly less accurate or the larger linear machine underfit the training data. The new version changes this preference criteria to take into consideration the misclassification costs of the two linear machines. Instead of preferring a linear machine that has a higher classification accuracy we now prefer the linear machine that has a lower misclassification cost. We retain the criterion that the linear machine not underfit the data.

3.1.2 Cost Reduction Pruning

Given a misclassification cost function, pruning schemes that seek to reduce overall classification error can be adapted to reduce overall cost of the misclassifications. There are many different tree pruning methods, but all use some estimate of the true error to prune back the tree [1]. LMDT uses reduced-error pruning, which computes the estimate of the true error using a set of instances that is independent from the training set [10]. When deciding whether to prune back a subtree, the cost of keeping the subtree is compared to the cost of replacing the subtree with a leaf node. To compute the cost, we weight each classification error by the cost of that error (as given by the MCF) and then sum the computed costs across all of the errors.

4 A Demonstration of Minimum-cost Classification

To demonstrate minimum misclassification cost learning, we considered a scenario in which the classifier's task is to focus attention on roads in outdoor scenes. In particular, the

Table 3: Varying the Ratio of False Negatives to False Positives

Ratio FN : FP	False Negatives	False Positives	Accuracy Test Set
1 : 1	44.6	26.8	83.6
2 : 1	27.0	44.2	83.2
5 : 1	17.8	53.0	83.3
10 : 1	10.8	125.4	80.0
20 : 1	9.2	190.8	79.3
200 : 1	8.4	296.0	75.0

classifier is part of a potential road-following algorithm for the UMass Mobile Perception Laboratory (MPL), an unmanned, outdoor autonomous vehicle [3]. The classifier is applied to the RGB pixel values from one-half of a stereo pair of color images, and the resulting labeling is used as a mask for a stereo-disparity algorithm which scans for obstacles. In general, the idea is that the vehicle should only drive over territory that is flat (as determined by the stereo algorithm), and roads are likely to be flat.

Of relevance to this paper is how the performance of the suggested road-following algorithm will be affected by errors in classification. As a rule, false positives, in which the classifier labels a non-road pixel as road, will force the stereo algorithm to match a pixel it should not have had to match. This will not endanger MPL, since the vehicle will only drive over a piece of ground if the stereo algorithm determines that it is flat, but it may force it to slow down to allow extra time for the stereo algorithm. False negatives also do not endanger MPL; however they may force it to stop completely if it is unable to find the road. As a rule, therefore, false negatives are more damaging to MPL's performance than false positives. (Note that if the pixel labels were used to generate a steering direction without stereo, then the opposite might be true: false positives might be more costly than false negatives. This only emphasizes the point that the relative costs associated with false positives and false negatives are application-specific.)

To test our modifications to the LMDT algorithm, we systematically varied the misclassification cost function across a set of otherwise identical experiments. The data consisted of 8,211 randomly selected pixels from four rural New England road scenes, with each pixel having a red, green and blue value. (No texture or positional information was used.) For training purposes, hand-segmentations of the images divide the pixels into nine classes (foliage, sky, foliage-sky boundary, road, roadline, gravel, tree trunk, dirt, or grass). On each trial, a classifier was induced from the training and pruning sets and applied to the test set.

We began by assigning every confusion an equal weight, a strategy that leads LMDT to optimize overall classification accuracy. Averaged over five trials, LMDT correctly classified 83.6% of all pixels, as shown in table 3. We then began to alter the MCF, steadily increasing the cost for false negative road hypotheses relative to the cost for false positive ones. In so doing, we shifted LMDT into an FOA mode, in which the classifier is willing to create a few "false alarms" in order to avoid missing part of the road. (All confusions not involving the "road" category were assigned equal weight of 0.1.) At a ratio of $2:1$, the system begins to label more pixels as "road", creating more false positives but fewer false negatives. At a ratio of $10:1$, LMDT creates almost no false negatives, albeit with an increase in false positives. Beyond $10:1$ system performance degrades, as the false negative count, which is already negligible, cannot decrease significantly, while the number of false positives continues to climb.

Table 3 shows the false negative and false positive counts of the road pixels (combined over the four images) for each MCF. The first column shows the ratio of the cost of classifying a road pixel as something else (a false negative) to the cost of classifying a non-road pixel as road (a false positive). The next two columns report the number of false negatives and false positives in the set of test instances, averaged over five runs; each run uses a different random partition of the data set into training (50%), pruning (25%) and test (25%) instance sets[5]. The final column reports the overall classification accuracy for the test instances. As

[5]Note that the instances for the test set are sampled randomly and are not sampled according to cost.

expected, the overall classification accuracy decreases as the cost of mislabelling the road is increased. However, by reducing the number of false negatives, the system's performance as a FOA mechanism improves, at least until the cost ratio approaches $10:1$.

Table 3 gives a concise, quantitative summary of LMDT's performance under different MCFs. To show the impact changing the MCF can have on the classification of a single image, Figure 1 shows one of the images and the pixels labeled as "road" under three different MCFs. The upper right hand corner shows the road pixels under a $1:1$ cost ratio between false negatives and false positives. Although the overall labeling is quite good, many road pixels on the left side of the foreground are missed. As the ratio of costs between false negatives and false positives is raised to $5:1$ (lower left corner), these false negatives are removed and the structure of the road becomes clearer. As the ratio is further increased to $20:1$ (lower right corner), the benefit of removing a few additional false negatives is overwhelmed by the clutter introduced by the increase in false positives.

An even more compelling example is shown in Figure 2 where the false negatives produced by the $1:1$ classification are bunched near the top, so that the road "disappears" before it recedes into the background (see the upper-right hand corner of Figure 2). When the ratio of false negative to false positive costs is raised to $5:1$, however, the spatial extent of the road becomes clear. In essence, the $5:1$ MCF creates an FOA that allows the system to see more of the road, increasing the distance ahead an autonomous navigation system can see. (As in Figure 1, the $20:1$ ratio adds more to clutter than to clarity.)

5 Conclusion

In many computer vision applications, classification is a means to an end, rather than the goal. Consequently, classification learning algorithms should be judged by how well they support the system's final goal, rather than by their classification accuracy. One way to measure how well a classifier supports a given goal is through a misclassification cost function (MCF), which assigns a cost (weight) to every type of confusion based on the impact the confusion would have on later processing. The ideal classifier is then defined as the one that minimizes the total misclassification cost. This paper presents a modification of the LMDT algorithm that minimizes the total misclassification cost for arbitrary misclassification cost functions (MCFs), allowing multivariate decision trees to be tailored to specific goals.

References

[1] Breiman, L., Freidman, J.H., Olshen, R.A., and Stone, C.J. *Classification and Regression Trees,* Wadsworth Inc., Belmont, CA., 1984.

[2] Brodley, C.E. and Utgoff, P.E. "Multivariate Decision Trees", *Machine Learning,* to appear.

[3] Draper, B., Buluswar, S., Hanson, A. and Riseman, E. "Information Acquisition and Fusion in the Mobile Perception Laboratory," *Proc. of Sensor Fusion VI,* Boston, MA., Aug. 1993, pp. 175-187.

[4] Duda, R.O., and Fossum, H. "Pattern Classification by Iteratively Determined Linear and Piecewise Linear Discriminant Functions," *IEEE Trans. on Electronic Computers,* 15(2):220-232. 1966.

[5] Duda, R.O., and Hart, P.E. *Pattern Classification and Scene Analysis.* Wiley & Sons, New York. 1973.

[6] Frean, M. *Small nets and short paths: Optimising Neural Computation.* Ph.D. thesis, Center for Cognitive Science, University of Edinburgh. 1990.

[7] Mingers, J. "An Empirical Comparison of Pruning Methods for Decision Tree Induction," *Machine Learning,* 4:227-243. 1989.

[8] Nilsson, N.J. *Learning Machines.* McGraw-Hill, New York, 1965.

[9] Quinlan, J.R. "Induction of Decision Trees," *Machine Learning,* 1:81-106, 1986.

[10] Quinlan, J.R. "Simplifying Decision Trees", *International Journal of Man-machine Studies,* 27:221-234. 1987.

[11] Utgoff, P.E. and Brodley, C.E. "An Incremental Method for Finding Multivariate Splits for Decision Trees," *Proc. of the Seventh International Conference on Machine Learning,* Austin, TX., 1990. Morgan-Kaufman.

Figure 1: A color image and the pixels classified as "road" under three different MCFs. The classification in the upper left-hand corner optimizes a MCF with a 1 : 1 cost ratio between false negative road errors and false positive road errors. The lower left optimizes a 5 : 1 false negative to false positive ratio, while the lower right optimizes a 20 : 1 ratio. In the 1 : 1 classification, overall accuracy is optimized but false negative road pixels are tolerated, as can be seen in the lower foreground. At 5 : 1, false negatives are discourages, leading to more road detection. At 20 : 1 false negatives are almost completely suppressed, but the number of false positives has grown sharply.

Figure 2: A color image and the pixels classified as "road" under three different MCFs. The classification in the upper left-hand corner optimizes a MCF with a $1:1$ cost ratio between false negative road errors and false positive road errors. The lower left optimizes a $5:1$ false negative to false positive ratio, while the lower right optimizes a $20:1$ ratio. Notice how the receding portions of the road are missed by the $1:1$ MCF, but captured by the two biased MCFs.