OC1: Randomized induction of oblique decision trees

Sreerama Murthy[1], Simon Kasif[1], Steven Salzberg[1], Richard Beigel[2]

[1]Dept. of Computer Science, Johns Hopkins University, Baltimore, MD 21218
[2]Dept. of Computer Science, Yale University, New Haven, CT 06520
[1]lastname@cs.jhu.edu, [2]beigel-richard@cs.yale.edu

Abstract

This paper introduces OC1, a new algorithm for generating multivariate decision trees. Multivariate trees classify examples by testing linear combinations of the features at each non-leaf node of the tree. Each test is equivalent to a hyperplane at an oblique orientation to the axes. Because of the computational intractability of finding an optimal orientation for these hyperplanes, heuristic methods must be used to produce good trees. This paper explores a new method that combines deterministic and randomized procedures to search for a good tree. Experiments on several different real-world data sets demonstrate that the method consistently finds much smaller trees than comparable methods using univariate tests. In addition, the accuracy of the trees found with our method matches or exceeds the best results of other machine learning methods.

1 Introduction

Decision trees (DTs) have been used quite extensively in the machine learning literature for a wide range of classification problems. Many variants of DT algorithms have been introduced, and a number of different goodness-of-split criteria have been explored. Most of the research to date on decision tree algorithms has been restricted to either (1) examples with symbolic attribute values [Quinlan, 1986] or (2) univariate tests for numeric attributes [Breiman *et al.*, 1984], [Quinlan, 1992]. Univariate tests compare the value of a single attribute to a constant; i.e., they are equivalent to partitioning a set of examples with an axis-parallel hyperplane. Although Breiman et al [1984] suggested an elegant method for inducing multivariate linear decision trees, there has not been much activity in the development of such trees until very recently [Utgoff and Brodley, 1991], [Heath *et al.*, 1992]. Because these trees use oblique hyperplanes to partition the data, we call them oblique decision trees.

This paper presents a new method for inducing oblique decision trees. As it constructs a tree, this method searches at each node for the best hyperplane to partition the data. Although most of the searching is deterministic hill-climbing, we have introduced randomization to determine the initial placement of a hyperplane and to escape from local minima. By limiting the number of random choices, the algorithm is guaranteed to spend only polynomial time at each node in the tree. In addition, randomization itself has produced several benefits. Our experiments indicate that it successfully avoids local minima in many cases. Randomization also allows the algorithm to produce many different trees for the same data set. This offers the possibility of a new family of classifiers: $k-$decision-tree algorithms, in which an example is classified by the majority vote of k trees (See [Heath, 1992]).

Two other methods for generating oblique trees, that have been introduced recently, are perceptron trees [Utgoff and Brodley, 1991] and simulated annealing (SADT) [Heath *et al.*, 1992]. The former shows that much smaller trees can be induced when oblique hyperplanes are used. However, theirs is a deterministic algorithm, and Heath [1992] shows that the problem of finding an optimal oblique tree is NP-Complete.[3] This work also introduces a completely randomized technique for finding good hyperplanes. The motivation for randomization is given in [Heath *et al.*, 1992], but the idea can briefly be explained as follows. Consider the hyperplane associated with the root of a decision tree. The optimal (smallest) decision tree may use non-optimal decision plane at the root. Obviously this is true for each node of the tree; this observation suggests a randomized strategy where we try to construct the smallest tree using several candidate hyperplanes at each node. This idea can be facilitated by using a randomized algorithm to find good separating hyperplanes. That is, if a randomized algorithm is executed repeatedly, it will find different hyperlanes each time. [Heath *et al.*, 1992] use an algorithm based on simulated annealing to generate good splits. Our

[3]More precisely, Heath [Heath, 1992] proves that the problem of finding an optimal oblique split is NP-Complete, using the number of misclassified examples as the error measure.

method is also randomized, but it includes a substantial directed search component that allows it to run much faster. In our experiments, our method ran much faster than SADT without sacrificing accuracy in the resulting classifier.

The algorithmic content of this paper focusses on the question of how to partition a given sample space into homogeneous regions. A complete description of any DT building method should also include discussion of its choices regarding the pruning strategies and the stop-splitting criteria. However, we do not address these issues here, because our choices for them are quite straightforward and standard in the literature. We stop splitting when the sample space associated with the current node has zero impurity (see Section 2.4). The only pruning done by our method consists of cutting off subtrees at nodes whose impurity measure is less than a certain threshold. For a good review and comparison of pruning strategies, see [Mingers, 1989] and [Quinlan, 1992].

The problem of partitioning the sample space involves the following related issues:

- restrictions on the location and orientation of hyperplanes,
- goodness measures for evaluating a split,
- strategies to search through the space of possible hyperplanes for the best hyperplane, and
- methods for choosing a hyperplane from which the above search begins.

These issues are fundamental to the design of a DT algorithm [Breiman *et al.*, 1984], and many existing DT algorithms can be classified on the basis of how they make these choices. Section 2 elaborates our algorithm with respect to each of these issues. Section 3 presents the results of using our method to classify several real-world data sets, and compares our results to those of some existing methods. Section 4 summarizes the lessons learned from these experiments.

2 The OC1 Algorithm

In this section we discuss details of our oblique decision tree learning method. We call this algorithm OC1, for **O**blique **C**lassifier 1. OC1 imposes no restrictions on the orientation of the hyperplanes. This is the main difference between OC1 and methods such as ID3 and CART, which use only axis-parallel hyperplanes. However, OC1 cannot distinguish between two hyperplanes that have identical sets of points on both sides. In other words, if the sample space consists of n examples in d dimensions (d attributes), then our algorithm recognizes only $\binom{n}{d}$ distinct hyperplanes.

The initial hyperplane at each node in the decision tree is chosen randomly by OC1. Even if such a randomly placed hyperplane has a very poor location, it is usually improved greatly in the first few perturbations.

2.1 Search Strategies

The strategy of searching through the space of possible hyperplanes is defined by the procedure that perturbs the current hyperplane into a new location. As there are an exponential number, $\binom{n}{d}$, of possible hyperplane locations, any procedure that simply enumerates all of them will be unreasonably costly. The two main alternatives considered in the past have been to use a non-deterministic search procedure, as in SADT [Heath *et al.*, 1992], or to use a heuristic deterministic procedure, as in CART [Breiman *et al.*, 1984]. OC1 combines these two approaches, using heuristic search until it finds a local minimum, and then using a non-deterministic search step to get out of the local minimum.

We will start by explaining how we perturb a hyperplane to split the sample space P at a node of a DT. P contains n examples, each with d attributes. Each example belongs to a particular category. The equation of the current hyperplane H can be written:

$$\sum_{i=1}^{d}(a_iX_i) + a_{d+1} = 0$$

Let $P_j = (x_{j1}, x_{j2}, \ldots, x_{jd})$ be the jth example from the sample space P. If we substitute P_j into the equation for H, we get: $\sum_{i=1}^{d}(a_ix_{ji}) + a_{d+1} = V_j$, where the sign of V_j tells us whether the point P_j is above or below the hyperplane H. If H splits the sample space P perfectly, then all points belonging to the same category in P will have the same sign i.e., $sign(V_j) = sign(V_k)$ iff $category(P_i) = category(P_j)$

OC1 perturbs the coefficients of H one at a time. If we consider the coefficient a_m as a variable, and all other coefficients as constants, V_j can be viewed as a function of a_m. If U_j is defined as

$$U_j = \frac{a_m x_{jm} - V_j}{x_{jm}} \tag{1}$$

then the point P_j is above H if $a_m > U_j$, and below otherwise. Thus, by fixing the values of the coefficients $a_1 \ldots a_{d+1}$, except a_m, we can obtain n constraints on the value of a_m, using the n points in the set P (assuming no degeneracies).

The problem then is to find a value for a_m that satisfies as many of these constraints as possible. (If all the constraints are satisfied, then we have a perfect split.) This problem is easy to solve; in fact, it is just an axis parallel split in 1-D. The value a_{m_1} obtained by solving this one dimensional problem is a good candidate to be used as the new value of the coefficient a_m. Let H_1 be the hyperplane obtained by changing a_m to a_{m_1} in H. If H has better (lower) impurity than H_1, then H_1 is discarded. If H_1 has lower impurity, H_1 becomes the new location of the hyperplane. If H and H_1 have identical impurities, and different locations, then H_1 is accepted with probability *stag_prob*.

```
Perturb(H,m)
{
  for j = 1 to n
      Compute U_j (Eq. 1)
  Sort U_1 ... U_n in nondecreasing order.
  a_{m_1} = best univariate split of the sorted U_j s.
  H_1 = result of substituting a_{m_1} for a_m in H.
  If (impurity(H) < impurity(H_1))
      { a_m = a_{m_1} ; stagnant = 0 }
  Else if (impurity(H) = impurity(H_1))
      { a_m = a_{m_1} with probability
                    stag_prob = e^{-stagnant}
      stagnant = stagnant + 1 }
}
```

Figure 1: Perturbation Algorithm

The parameter *stag_prob*, denoting "stagnation probability", is the probability that a hyperplane is perturbed to a location that does not change the impurity measure. To prevent the impurity from remaining stagnant for a long time, *stag_prob* decreases exponentially with the number of "stagnant" perturbations. It is reset to 1 every time the global impurity measure is improved. Pseudocode for our perturbation procedure is given in Fig. 1.

Now that we have a method for locally improving a coefficient of a hyperplane, we need a method for deciding which of the $d+1$ coefficients to pick for perturbation. We experimented with three different orders of coefficient perturbation, which we labelled Seq, Best, and R-50:

Seq : Repeat until none of the coefficient values is modified in the **for** loop:
  For $i = 1$ to $d+1$, Perturb(H, i)

Best: Repeat until coefficient m remains unmodified :
  m = coefficient which when perturbed, results in the maximum improvement of the impurity measure.
  Perturb(H,m)

R-50: Repeat a fixed number of times : (50 in our experiments)
  m = random integer between 1 and $d+1$
  Perturb(H,m)

As will be shown in our experiments (Section 3), the order of perturbation of the coefficients does not affect the classification accuracy as much as other parameters, especially the number of iterations (see Section 2.2.2). But if the number of iterations and the impurity measure are held constant, the order can have a significant effect on the performance of the method. In our experiments, though, none of these orders was uniformly better than any other.

A sequence of perturbations stops when the split reaches a local minimum (which may also be a global minimum) for the impurity measure. Our method uses randomization to try to jump out of local minima. This randomization technique is described next.

2.2 Local Minima

A big problem in searching for the best hyperplane (and in many other optimization problems, as well) is that of local minima. The search process is said to have reached a local minimum if no perturbation of the current hyperplane, as suggested by the perturbation algorithm, decreases the impurity measure, and the current hyperplane does not globally minimize the impurity measure.

We have implemented two ways of dealing with local minima: perturbing the hyperplane in a random direction, and re-running the perturbation algorithm with additional initial hyperplanes. While the second technique is a variant of the standard technique of multiple local searches, the first technique of perturbing the hyperlane in a random direction is novel in the cont ext of decision tree algorithms. Notably, moving the hyperlane in a random direction rather than modifying one of the coefficients one at a time does not modify the time complexity of the algorithm.

2.2.1 Perturb coefficients in a random direction When a hyperplane $H = \sum_{i=1}^{d} a_i * x_i + a_{d+1}$ can not be improved by deterministic perturbation, we do the following.

- Let $R = (r_1, r_2, \ldots, r_{d+1})$ be a random vector. Let α be the amount by which we want to perturb H in the direction R. i.e., Let $H_1 = \sum_{i=1}^{d} (a_i + \alpha r_i) x_i + (a_{d+1} + \alpha r_{d+1})$ be the suggested perturbation of H.
- The only variable in the equation of H_1 is α. Therefore each of the n examples in P, depending on its category, imposes a constraint on the value of α (See Section 2.1). Use the perturbation algorithm in Fig. 1 to compute the best value of α.
- If the hyperplane H_1 obtained thus improves the impurity measure, accept the perturbation. Continue with the coefficient perturbation procedure. Else stop and output H as the best possible split of P.

We found in our experiments that a single random perturbation, when used at a local minimum, proves to be very helpful. Classification accuracy improved for every one of our data sets when such perturbations were made.

2.2.2 Choosing multiple initial hyperplanes Because most of the steps of our perturbation algorithm are deterministic, the initial randomly-chosen hyperplane determines which local minimum will be encountered first. Perturbing a single initial hyperplane deterministically thus is not likely to lead to the best split of a given dataset. In cases where the random perturbation method may have failed to escape from local minima, we thought it would be useful to start afresh, with a new initial hyperplane.

We use the word *iteration* to denote one run of the perturbation algorithm, at one node of the decision tree, using one random initial hyperplane; i.e., one attempt using either Seq, Best, or R-50 to cycle through and perturb the coefficients of the hyperplane. One iteration also includes perturbing the coefficients randomly once at each local minimum, as described in Section 2.2.1. One of the input parameters to OC1 tells it how many iterations to use. If it uses more than one iteration, then it always saves the best hyperplane found thus far.

In all our experiments, the classification accuracies increased with more than one iteration. Accuracy seemed to increase up to a point and then level off (after about 20–50 iterations, depending on the domain). Our conclusion was that the use of multiple initial hyperplanes substantially improved the quality of the best tree found.

2.3 Comparison to Breiman et al.'s method

Breiman et al [1984, pp. 171–173] suggested a method for inducing multivariate decision trees that used a perturbation algorithm similar to the deterministic hill-climbing method that OC1 uses. They too perturb a coefficient by calculating a quantity similar to U_j (Eq. 1) for each example in the data, and assign the new value of the coefficient to be equal to the best univariate split of the U_js. In spite of this apparent similarity, OC1 is significantly different from the above algorithm for the following reasons.

- Their algorithm does not use any randomization. They choose the best univariate split of the dataset as their only choice of an initial hyperplane. When a local minimum is encountered, their deterministic algorithm halts.
- Their algorithm modifies one coefficient of the hyperplane at a time. One step of our algorithm can modify several coefficients at once.
- Breiman et al. report no upper bound on the time it takes for a hyperplane to reach a (perhaps locally) optimal position. In contrast, our procedure only accepts a limited number of perturbations. The number of changes that reduce the impurity is limited to n, the number of examples. The number of changes that leave impurity the same is limited by the parameter *stag_prob* (Section 2.1). Due to these restrictions, OC1 is guaranteed to spend only polynomial time on each hyperplane in a tree.[4]

In addition, the procedure in [Breiman *et al.*, 1984] is at best an outline: though the idea is elegant, many details were not worked out, and few experiments were performed. Thus, even without the significant changes to the algorithm we have introduced, there was a need for much more experimental work on this algorithm.

2.4 Goodness of a hyperplane

Our algorithm attempts to divide the d-dimensional attribute space into homogeneous regions, i.e., into regions that contain examples from just one category. (The training set P may contain two or more categories.) The goal of each new node in the tree is to split the sample space so as to reduce the "impurity" of the sample space. Our algorithm can use any measure of impurity, and in our experiments, we considered four such measures: information gain [Quinlan, 1986], max minority, sum minority, and sum of impurity (all three defined in [Heath, 1992]). Any of these measures seem to work well for our algorithm, and the classification accuracy did not vary significantly as a function of the goodness measure used. More details of the comparisons are given in Section 3 and Table 2.

2.4.1 Three new impurity measures The impurity measures max minority, sum minority, and sum of impurity were all very recently introduced in the context of decision trees. We will therefore briefly define them here. For detailed comparisons, see [Heath, 1992]. For a discussion of other impurity measures, see [Fayyad and Irani, 1992] and [Quinlan and Rivest, 1989].

Consider the two half spaces formed by splitting a sample space with a hyperplane H, and call these two spaces L and R (left and right). Assume that there are only two classes of examples, though this definition is easily extended to multiple categories. If all the examples in a space fall into the same category, that space is said to be *homogeneous*. The examples in any space can be divided into two sets, A and B, according to their class labels, and the size of the smaller of those two sets is the *minority*. The *max minority* (MM) measure of H is equal to the larger of the two minorities in L and R. The *sum minority* measure (SM) of H is equal to the sum of the minorities in both L and R.

The *sum of impurity* measure requires us to give the two classes numeric values, 0 and 1. Let $P_1, .., P_L$ be the points (examples) on the left side of H. Let C_{P_i} be the category of the point P_i. We can define the average class avg of L as $avg = \frac{\sum_{i=1}^{L} C_{P_i}}{L}$. The *impurity* of L is then defined as $\sum_{i=1}^{L}(C_{P_i} - avg)^2$ The sum of impurity (SI) of H is equal to the sum of the impurity measures

[4] The theorethical bound on the amount of time OC1 spends on perturbing a hyperplane is $O(dn^2 \log n)$. To guarantee this bound, we have to reduce *stag_prob* to zero after a fixed number of changes, rather than reducing it exponentially to zero. The latter method leaves an exponentially small chance that a large number of perturbations will be permitted. In practice, however, hyperplanes were never perturbed more than a small (< 12) times. The expected running time of OC1 for perturbing a hyperplane appears to be $O(kn \log n)$, where k is a small constant.

Table 1: Comparisons with other methods

Data	Method	Accuracy (%)	Tree Size	Impurity Measure
Star Galaxy (Bright)	OC1	99.2	15.6	SI
	CSADT	99.1	18.4	SI
	ID3	99.1	44.3	SI
	1-NN	98.8	—	—
	BP	99.8	—	—
Star Galaxy (Dim)	OC1	95.8	36.0	SI
	1-NN	95.1	—	—
	BP	92.0	—	—
IRIS	OC1	98.0	3.0	SI
	CSADT	94.7	4.2	SM
	ID3	94.7	10.0	MM
	1-NN	96.0	—	—
	BP	96.7	—	—
Cancer	OC1	97.4	2.4	SI
	CSADT	94.9	4.6	SM
	ID3	90.6	36.1	SI
	1-NN	96.0	—	—

on both L and R.

3 Experiments

In this section, we present results of experiments we performed using OC1 on four real-world data sets. These results, along with some existing classification results for the same domains, are summarized in Table 1. All our experiments used 10-fold cross-validation trials. We built decision trees for each data set using various combinations of program parameters (such as the number of iterations, order of coefficient perturbation, impurity measure, impurity threshold at which a node of the tree may be pruned). The results in Table 1 correspond to the trees with the highest classification accuracies.

The results for the CSADT and ID3 methods are taken from Heath [Heath, 1992]. CSADT is an alternative approach to building oblique decision trees that uses simulated annealing to find good hyperplanes. These prior results used identical data sets to the ones used here, although the partitioning into training and test partitions may have been different. In each case, though, we cite the best published result for the algorithm used in the comparison.

Star/galaxy discrimination. Two of our data sets came from a large set of astronomical images collected by Odewahn et al [Odewahn *et al.*, 1992]. In their study, they used these images to train perceptrons and back propagation (BP) networks to differentiate between stars and galaxies. Each image is characterized by 14 real-valued attributes and one identifier, viz., "star" or "galaxy". The objects in the image were divided by Odewahn et al. into "bright" and "dim" data sets based on the image intensity values, where the "dim" images are inherently more difficult to classify. The bright set contains 3524 objects and the dim set contains 4652 objects.

Heath [Heath, 1992] reports the results of applying the SADT and ID3 algorithms only to the bright images. We ran OC1 on both the bright and dim images, and our results are shown in Table 1. The table compares our results with those of CSADT, ID3, 1-nearest-neighbor (1-NN), and back propagation on bright images, and with 1-NN [Salzberg, 1992] and back propagation on the dim images.

Table 2: Effect of parameters on accuracy and DT size

Iter	Imp. Meas.	Order	Prune Thresh.	Acc. (%)	Tree Depth & Size
1	SI	R-50	10	96.4	3.0,4.9
10	SM	Best	4	97.0	3.3,4.3
10	SM	Seq	10	96.6	2.3,3.3
20	SM	R-50	8	96.8	3.1,4.3
50	MM	Best	6	97.1	1.9,2.8
100	SI	Best	8	96.9	1.9,2.3
1	MM	Seq	0	93.7	6.2,19.6
1	MM	Seq	2	93.8	4.9,14.3
1	MM	Seq	10	92.5	2.9,5.6
1	MM	Best	10	89.2	3.9,6.7
1	MM	R-50	10	92.3	2.8,5.0

Classifying irises. The iris dataset has been extensively used both in statistics and for machine learning studies [Weiss and Kapouleas, 1989]. The data consists of 150 examples, where each example is described by four numerical attributes. There are 50 examples in each of three different categories. Weiss and Kapouleas [Weiss and Kapouleas, 1989] obtained accuracies of 96.7% and 96.0% on this data with back propagation and 1-NN, respectively.

Breast cancer diagnosis. A method for classifying using pairs of oblique hyperplanes was described in [Mangasarian *et al.*, 1990]. This was applied to classify a set of 470 patients with breast cancer, where each example is characterized by nine numeric attributes plus the label, benign or malignant. The results of CSADT and ID3 are from Heath [Heath, 1992], and those of 1-NN are from Salzberg [Salzberg, 1991].

Table 2 shows how the OC1 algorithm's performance varies as we adjust the parameters described earlier. The table summarizes results from different trials using the cancer data. We ran similar experiments for all our data sets, but due to space constraints this table is shown as a representative. The most important parameter is the number of iterations; we consistently found better trees (smaller and more accurate) using 50 or more iterations. There was no significant correlation between pruning thresholds and accuracies, and the sum minority (SM) impurity measure almost always produced the smallest (though not always the most accurate) trees. We did not find any other significant sources of variation, either in the impurity measure or the order of perturbing coefficients.

4 Conclusions

Our experiments seem to support the following conclusions:

- The use of multiple iterations; i.e., several different initial hyperplanes, substantially improves performance.
- The technique of perturbing the entire hyperplane in the direction of a randomly-chosen vector is a good means for escaping from local minima.
- No impurity measure has an overall better performance than the other measures for OC1. The nature of the data determines which measure performs the best.
- No particular order of coefficient perturbation is superior to all others.

One of our immediate next steps in the development of OC1 will be to use the training set to determine the program parameters (e.g., number of iterations, best impurity measure for a dataset, and order of perturbation).

The experiments contained here provide an important demonstration of the usefulness of oblique decision trees as classifiers. The OC1 algorithm produces remarkably small, accurate trees, and its computational requirements are quite modest. The small size of the trees makes them more useful as descriptions of the domains, and their accuracy provides a strong argument for their use as classifiers. At the very least, oblique decision trees should be used in conjunction with other methods to enhance the tools currently available for many classification problems.

Acknowledgements

Thanks to David Heath for helpful comments. S. Murthy and S. Salzberg were supported in part by the National Science Foundation under Grant IRI-9116843.

References

Breiman, L.; Friedman, J.H.; Olshen, R.A.; and Stone, C.J. 1984. *Classification and Regression Trees.* Wadsworth International Group.

Fayyad, U. and Irani, K. 1992. The attribute specification problem in decision tree generation. In *Proceedings of AAAI-92*, San Jose CA. AAAI Press. 104–110.

Heath, D.; Kasif, S.; and Salzberg, S. 1992. Learning oblique decision trees. Technical report, Johns Hopkins University, Baltimore MD.

Heath, D. 1992. *A Geometric Framework for Machine Learning.* Ph.D. Dissertation, Johns Hopkins University, Baltimore MD.

Mangasarian, O.; Setiono, R.; and Wolberg, W. 1990. Pattern recognition via linear programming: Theory and application to medical diagnosis. In *SIAM Workshop on Optimization.*

Mingers, J. 1989. An emperical comparison of pruning methods for decision tree induction. *Machine Learning* 4(2):227–243.

Odewahn, S.C.; Stockwell, E.B.; Pennington, R.L.; Humphreys, R.M.; and Zumach, W.A. 1992. Automated stargalaxy descrimination with neural networks. *Astronomical Journal* 103(1):318–331.

Quinlan, J.R. and Rivest, R.L. 1989. Inferring decision trees using the minimum description length principle. *Information and Computation* 80:227–248.

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

Quinlan, J.R. 1992. *C4.5 Programs for Machine Learning.* Morgan Kaufmann.

Salzberg, S. 1991. Distance metrics for instance-based learning. In *Methodologies for Intelligent Systems: 6th International Symposium, ISMIS '91.* 399–408.

Salzberg, S. 1992. Combining learning and search to create good classifiers. Technical Report JHU-92/12, Johns Hopkins University, Baltimore MD.

Utgoff, P.E. and Brodley, C.E. 1991. Linear machine decision trees. Technical Report 10, University of Massachusetts, Amherst MA.

Weiss, S. and Kapouleas, I. 1989. An emperical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proceedings of Eleventh IJCAI*, Detroit MI. Morgan Kaufmann.