Induction of Oblique Decision Trees

David Heath and **Simon Kasif** and **Steven Salzberg**
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218
(410) 516-8296
lastname@cs.jhu.edu

Abstract

This paper introduces a randomized technique for partitioning examples using oblique hyperplanes. Standard decision tree techniques, such as ID3 and its descendants, partition a set of points with axis-parallel hyperplanes. Our method, by contrast, attempts to find hyperplanes at any orientation. The purpose of this more general technique is to find smaller but equally accurate decision trees than those created by other methods. We have tested our algorithm on both real and simulated data, and found that in some cases it produces surprisingly small trees without losing predictive accuracy. Small trees allow us, in turn, to obtain simple qualitative descriptions of each problem domain.

1 Introduction

Decision trees have been used successfully for many different decision making and classification tasks. A number of standard techniques have been developed in the machine learning community, most notably Quinlan's ID3 (1986) and Breiman *et al.*'s CART (1984). Since the introduction of these algorithms, numerous variations and improvements have been put forward, including new pruning strategies (e.g., Quinlan, 1987) and incremental versions of the algorithms (Utgoff, 1989). Many of these refinements have been designed to produce better decision trees; i.e., trees that were either more accurate classifiers, or smaller trees, or both.

The main goal of our research to produce decision trees that provide small, accurate models for a set of data. Many applications, including the ones discussed in this paper, could substantially benefit from reducing the size of the tree. In particular, a small tree will provide experts with important qualitative descriptions that may be used in real–world applications.

Most decision tree algorithms use axis–parallel hyperplanes to partition a dataset. That is, if each example is characterized as a vector of numbers $(x_1, x_2, \ldots, x_d)$, the internal nodes of the tree will contain tests of the form $x_i > k$. The task of the decision tree algorithm is to choose good values for i and k at each node. These tests are equivalent to axis–parallel hyperplanes. In order to provide the best fit to a dataset, we have chosen a more general representation: our algorithm uses oblique (non–axis–parallel) hyperplanes to partition the examples. In other words, each node of the tree contains a hyperplane that can take any orientation in feature space. Because finding these hyperplanes turns out to be computationally expensive, we introduce a randomized technique for selecting good hyperplanes. The goal of our work is to find smaller decision trees that provide a better fit to the data, and therefore are likely to be more accurate. Our results show that our method finds strikingly small trees for some datasets.

In the sections that follow, we present the following results:

- A new randomized approach for computing decision trees using non–axis–parallel hyperplanes. This use of randomized algorithms is new in the context of decision tree induction.
- A new NP–completeness result for the complexity of finding oblique hyperplanes. This result also provides a complexity bound for many other learning methods.
- Experiments demonstrating that our algorithm can produce very small oblique decision trees with accuracies comparable to those of large axis–parallel trees.

Our new algorithm, which is based on simulated annealing, produces a different decision tree each time it is run on a given dataset. This property, a direct consequence of randomization, is a major advantage in light of the fact that finding the best tree is NP–complete. In order to find a good tree, we train the algorithm repeatedly on a fixed set of data, and retain only the smallest tree. As we will show in Section 5, the program frequently finds very small and accurate trees. Our experiments compare our program, which we call SADT[1], to other decision tree algorithms (including ID3) on both simulated and real data.

2 Complexity of finding good trees

Standard decision tree algorithms work recursively; i.e., given a set of examples, they generate a test which splits

[1] Simulated Annealing of Decision Trees

the examples into two subsets. They then recurse on the subsets, continuing to generate tests until some stopping criterion is satisfied.

The main difference between our problem and the generation of axis–parallel trees is that our tests can take the more general form of an oblique hyperplane. Standard decision tree algorithms use exhaustive search to find the "best" hyperplane (the criteria used to define the best test is described below). Because there are only a small number of distinct axis–parallel hyperplanes (i.e., for n examples and d attributes, there are $d(n-1)$ placements), this can be done quite efficiently. However, the number of distinct partitionings of the same set of examples that can be induced with a single oblique hyperplane is $O(n^d)$, so that the straightforward approach of considering all possible tests is too expensive for many reasonable values of n and d. For example, one of the datasets used in Section 5 has $n = 4164$ and $d = 14$. This observation led us to consider the computational complexity of finding the best test when tests are represented by oblique hyperplanes. We found that for reasonable definitions of "best," this problem is NP–complete; i.e., it probably has no polynomial time algorithm. This result implies that exhaustive search is not feasible for many datasets.

Theorem 2.1: *Let the energy e of a hyperplane be defined by the sum–minority energy measure (defined below). Given two point sets K_1 and K_2 and a value k, the problem of determining if there is a hyperplane such that $e \leq k$ is NP–complete.*

For a proof of this theorem, see [Heath, 1992]. Note that this theorem says that finding even one optimal hyperplane is hard; clearly, then, building a whole tree is even harder.

It is worth noting that there do exist energy measures for which finding the best hyperplane is possible in polynomial time. See [Duda and Hart, 1973] for a description of different techniques of generating hyperplanes in the absence of linear separability.

2.1 Finding approximate solutions

It is clear that by abandoning the restriction to axis-parallel hyperplanes, we have made the problem of building decision trees considerably harder. Even at one particular node of the decision tree, we can not be sure of finding the *best* split, even if we define the goodness of the split in a simple, local way. What saves us from disaster is that the best tree may not use the best first split anyway! This suggests that a randomized method, one that tries a range of good (but not best) initial splits and produces a set of alternative trees, might produce good approximate solutions. This is exactly the approach we have taken.

As with many NP–hard optimization problems, our best strategy is to look for approximations or good heuristic solutions. There are several standard approaches for computationally hard optimization problems. Such possibilities include local search or gradient descent on the cost function. Typically, such methods suffer from the problem of getting stuck in local maxima. Two standard solutions to this problem are applying multiple local searches and applying a stochastic optimization approach such as simulated annealing. Perceptron trees, introduced in Utgoff and Brodley (1990) can be seen as an instance of applying multiple local searches to the problem of finding a good first split. Once one good split is found, they recursively apply the procedure (i.e., multiple local-search) to the resulting partitions of the data. Breiman et al. (1984) use a single application of a local improvement algorithm. They iterate on dimensions, finding the best perturbation in the dimension until no improvement can be made. Both algorithms are deterministic and output a single tree. Our algorithm produces a stream of different trees for a given training set. We can generate any number of these trees during training, and choose, for example, the smallest one for experimentation on the test-set.

It is well–understood that finding the best first split will not necessarily produce the smallest decision tree (i.e., the smallest tree may have a non-optimal root node). Allowing randomization in the procedure that finds the best first split, and producing many alternative trees, increases the probability of finding good trees. We believe that randomization is an important technique in the context of inductive learning procedures. Since none of the known deterministic learning procedures are guaranteed to generate good minimum-size representations (trees), the introduction of randomization gives us a higher probability of finding such representations.

Thus one of the principal claims we wished to investigate experimentally was whether, when an optimal tree cannot be found, randomized algorithms may produce better decision trees than simple deterministic algorithms. Given that any algorithm to find oblique decision trees cannot be guaranteed to find the best tree and still run in reasonable time, we decided that we needed an algorithm that could produce many different trees. Our SADT algorithm produces a different tree each time it runs. We sometimes run it hundreds of times on a given dataset, choosing the minimum size tree as the best one. Our argument is that picking the best of many solutions produced by a randomized algorithm may be preferable to using an algorithm, even a very clever one, that only produces one solution.

3 The simulated annealing algorithm

The basic outline of our algorithm is the same as that of most other decision tree algorithms. That is, we find a hyperplane to partition the training set and recurse on the two partitions. Here we describe the search for a good hyperplane.

In our implementation, d–dimensional hyperplanes are stored in the form $H(x) = h_{d+1} + \sum_{i=1}^{d} h_i x_i$, where $H = \{h_1, h_2, \ldots, h_{d+1}\}$ is the hyperplane, $x = (x_1, x_2, \ldots, x_d)$ is a point, and h_{d+1} represents the constant term. For example, in the plane the hyperplane is a line and is represented in the familiar $ax + by + c = 0$ form. Classification is done recursively. To classify an example, compare it to the current hyperplane (initially this is the root node). If an example p is at a non–leaf node labeled $H(x)$, then we follow the the left child if $H(p) \geq 0$; otherwise we descend to the right child.

The first step in our algorithm is to generate an initial hyperplane. The initial hyperplane we generate is always the same and is not tailored to the training set. We simply wanted to choose some hyperplane that was not parallel to any of the axes, so we used the hyperplane passing through the points where $x_i = 1$ and all other $x_j = 0$, for each dimension i. In particular, the initial hyperplane may be written in the above form as $h_i = 1$ for $1 \leq i \leq d$ and $h_{d+1} = -1$ since $H(x) = 0$ for each of these points. Thus in 3-D, we choose the hyperplane which passes through $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$. Many other choices for the initial hyperplane would be equally good. Once the annealing begins, the hyperplane is immediately moved to a new position, so the location of the initial split is not important.

Next, the hyperplane is repeatedly perturbed. If we denote the current hyperplane by $H = \{h_1, h_2, \ldots, h_{d+1}\}$, then the algorithm picks one of the h_i's randomly and adds to it a uniformly chosen random variable in the range $[-0.5, 0.5)$. Using our goodness measure (described below), we compute the energy of the new hyperplane and the change in energy ΔE.

If ΔE is negative, then the energy has decreased and the new hyperplane becomes the current split. Otherwise, the energy has increased (or stayed the same) and the new hyperplane becomes the current split with probability $e^{-\Delta E/T}$ where T is the *temperature* of the system. The system starts out with a high temperature that is reduced slightly with each move. Note that when the change in energy is small relative to the temperature, the probability of accepting the new hyperplane is close to 1, but that as the temperature becomes small, the probability of moving to a worse state approaches 0.

In order to decide when to stop perturbing the split, we keep track of the split that generated the lowest energy seen so far at the current node. If this minimum energy does not change for a large number of iterations (we used numbers between 3000 and 30000 iterations in our experiments), then we stop making perturbations and use the split that generated the lowest energy. The recursive splitting continues until each node is pure; i.e., each leaf node contains only points of one category.

3.1 Goodness criteria

Our SADT algorithm can work with any goodness criterion, and we have experimented with several criteria. For detailed discussions of these measures, see Heath [1992]. In these earlier studies, the *sum–minority* goodness measure generated smaller, more accurate trees than the others we tested. As a result, we chose the sum–minority goodness measure for the experiments described here. This measure is defined as follows.

Consider a set of examples X, belonging to 2 classes, u and v. A hyperplane divides the set into two subsets X_1 and X_2. For each subset, we find the class that appears least often. We say that these are the *minority* categories. If X_1 has few examples in its minority category C_1, then it is relatively *pure*. We prefer splits that are pure; i.e., splits that generate small minorities. Let the number of examples in class u (class v) in X_1 be u_1 (v_1) and the number of examples in class u (class v) in X_2 be u_2 (v_2). To force SADT to generate a relatively pure split, we define the sum–minority error measure to be $\min(u_1, v_1) + \min(u_2, v_2)$. For a description of this and other error measures, see [Heath, 1992].

3.2 Taking advantage of randomization

We believe randomization in learning algorithms can easily be used to advantage. Because SADT constructs different trees every time, it is possible to run it multiple times and pick a tree or set of trees that has some advantage over a more "average" tree. For example, to minimize tree size, one can run SADT several times and choose the smallest tree it creates.

Another way we can benefit from randomization is to take a set of SADT trees, and combine their classifications by taking the plurality. In bi-classification problems, this reduces to taking the majority. For example, if we have 5 trees, and 3 classify an example as "0," and the other two classify it as "1," then we predict the example belongs to class "0."

The premise behind this idea is that any one tree may not capture the target concept completely accurately, but will approximate it with some error. This error differs from tree to tree. By using several trees and taking the plurality, we hope to overcome this type of error.

4 An example in the plane

To illustrate the possible advantages that SADT offers, we demonstrate its use on an artificial dataset containing 200 examples in two classes, chosen uniformly from a two-dimensional space. We labeled the points according to a partitioning with three non-axis-parallel lines. Thus, the optimal partitioning (and the optimal tree) required three oblique "splits."

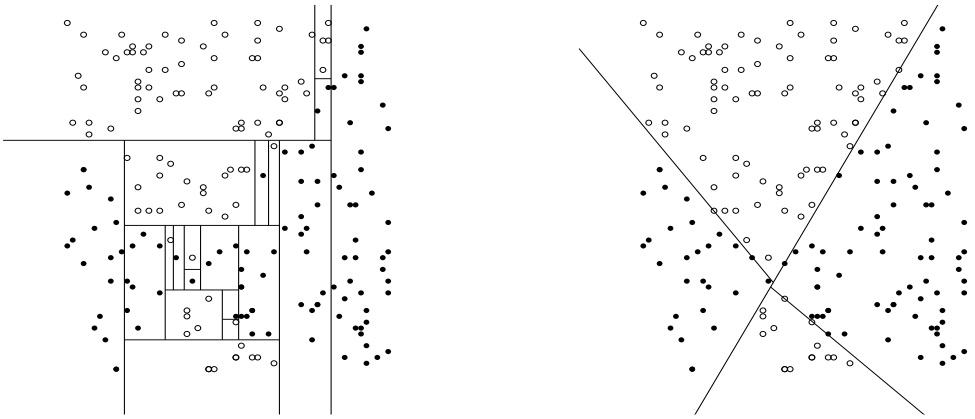
Figure 1 shows the tree generated by ID3 and one of the trees generated by SADT. The tree generated by ID3 did not fit the data very well, because the correct partitions were oblique lines. We ran one hundred trials with each of our goodness measures, and kept the smallest tree as the best. This example shows that, at least for simple problems, SADT can perform very well.

5 Experiments

5.1 Applying SADT to cancer diagnosis

For our first experiment, we ran SADT on a real dataset that has been the subject of other machine learning studies; in particular, it was the subject of experiments that partitioned the data using oblique hyperplanes (Mangasarian et al., 1990). This dataset contains 470 examples of patients with breast cancer, and the diagnostic task is to determine whether the cancer is benign or malignant. Each example has 9 numeric attributes, hence our decision trees used hyperplanes in 9-D.

Mangasarian's method uses linear programming to find pairs of hyperplanes that partition the data. The algorithm finds one pair of parallel hyperplanes at a time, and each pair can be oriented at any angle with respect to all other pairs. The resulting model is a set of oblique hyperplanes, similar though not identical to an oblique decision tree.



(A) (B)

Figure 1: Tree generated by (A) ID3 and (B) SADT

Algorithm	*Accuracy on Test Data (%)*	*Number of Hyperplanes*
SADT	97	1
ID3	83	30
Mangasarian et al.	97	16
EACH	95	
Nearest Neighbor	94	

Table 1: Breast cancer results

Algorithm	*Accuracy (%)*	*Std Dev*	*Tree Size*	*Std Dev*
SADT	94.9	3.47	4.58	2.0
ID3	88.1	9.89	27.8	4.7

Table 2: Cross-validated breast cancer results

Because Mangasarian et al. received the data as they were collected in a clinical setting, their experimental design was very simple. They trained their algorithm on the initial set of 369 examples. Of the 369 patients, 201 (54.5%) had no malignancy and the remainder had confirmed malignancies. On the next 70 patients to enter the clinic, they used their algorithm for diagnosis, and found that it correctly diagnosed 68 patients. We used $68/70 = 0.97$ as a rough estimate of the accuracy of Mangasarian et al.'s method. They then re-trained their algorithm using the 70 new patients, and reported that it correctly classified all of the next 31 patients to enter the clinic. Mangasarian reported that his program's output was being used in an actual clinical setting. Using the same dataset with a more uniform experimental design, Salzberg reported that the EACH hyper-rectangle program produced 95% classification accuracy, and 1-nearest-neighbor had 94% accuracy (Salzberg, 1991).

We imitated Mangasarian et al.'s experiment by using the same 369 examples from their initial trial to produce decision trees, which we then tested on the final 101 patients. It is not possible to show a nine-dimensional partitioning here, but our results and some comparisons are summarized in Table 1.

Mangasarian's model used 8 pairs of hyperplanes – 16 hyperplanes – to model the data. With SADT we were able to determine that the same accuracy can be attained with a single hyperplane. Our method uses a significantly smaller number of oblique planes, and produces comparable accuracy. The axis-parallel hyperplanes produced by ID3 made a much poorer classifier for this dataset — the tree was much larger and was less accurate than the SADT tree. As a side note, some of the larger trees generated by SADT scored 100% on the test set. The smallest such "perfect" tree contained 6 hyperplanes.

In addition, we ran a ten-fold cross-validation trial on the same set of data. In an x–fold cross-validation trial, we divide the dataset into x approximately equal sized subsets and perform x experiments. For each set s, we train the learning system on the union of the remaining $x - 1$ sets and test on set s. The results are averaged over these x runs. The results of these tests are shown in Table 2. For each of the ten training– and test–set pairs, we performed thirty–six trials. That is, each value in Table 2 is the average of 360 values and should be more indicative of the true accuracy on the problem. The SADT trees are somewhat more accurate than the axis-parallel trees. The size of the trees generated by SADT is considerably less that that of the trees generated by ID3. We note that there are different ways in which the size of trees can be measured. In this paper, we equate tree size with the number of hyperplanes, regardless of the description complexity of the hyperplanes.

5.2 Identifying stars and galaxies

In order to study the performance of SADT on larger datasets, we ran several experiments using astronomical image data collected with the University of Minnesota Plate Scanner. This dataset contains several thousand astronomical objects, all of which are classified as either

Algorithm	*Dataset*	*Accuracy (%)*	*Std Dev*	*Tree Size*	*Std Dev*
SADT	Bright	99.0	0.48	7.03	2.2
ID3		99.0	0.63	37.7	2.7
SADT	Dim	94.0	1.2	27.6	13
ID3		94.7	1.1	260	150

Table 3: Star/Galaxy results

stars or galaxies. Odewahn et al. [1992] used this dataset to train perceptrons and backpropagation networks.

Although we did not have access to the exact training and test set partitions used by Odewahn et al., we approximated the data using the same filtering of data from the same source. Although our results may not be completely comparable to theirs, we include them to show that both learning methods produce similar accuracies. Our results were generated by averaging nineteen 10–fold cross–validation trials.

We experimented on two different subsets of the astronomical data. One set is comprised of *dim* objects, the other of *bright* objects. Bright objects tend to be easier to classify. The bright dataset consists of 4164 examples; there are 4652 examples in the dim dataset. Each example has fourteen real-valued attributes and a label of either "star" or "galaxy." Approximately 35% of the examples are galaxies.

Classification results are shown in Table 3. Odewahn et al. [1992] obtained accuracies of 99.7% using backpropagation on the bright dataset and 92.2% on the dim dataset. We obtained a cross-validated accuracy of 99.1% with a perceptron on the bright dataset. Note that a perceptron can be considered a tree of size 1. This implies that SADT trees, despite their small size, could still benefit from pruning. The accuracy of the trees generated by SADT and those generated by ID3 was approximately the same in both cases. However, SADT was able to find much smaller trees than ID3.

5.3 Classifying irises

Fisher's [1936] iris data is a well known dataset, and many common learning techniques have been applied to it. The data consists of 150 examples, 50 each of three different types of irises: setosa, versacolor, and virginica. Each example is described by numeric measurements of width and length of the petals and sepals.

We performed thirty-five 10-fold cross-validation trials using SADT. Averaged results are shown in Table 4, along with 10-fold cross-validation trials for ID3. Weiss and Kapouleas [1989] obtained accuracies of 96.7%, 96.0%, and 95.3% with backpropagation, nearest neighbor, and CART, respectively, using leave-one-out trials, i.e., 150–fold cross validation.

Once again, we note that the accuracy of SADT is comparable to that of other learning techniques. The trees generated by SADT tend to be somewhat smaller and more accurate on average than those of ID3.

Algorithm	*Accuracy (%)*	*Std Dev*	*Tree Size*	*Std Dev*
SADT	94.7	5.99	4.24	1.5
ID3	90.0	8.5	7.8	1.7

Table 4: Iris results

Dataset	*Technique*	*Classification Accuracy*	*Tree Size*
Cancer	average	94.9	4.58
malignancy	smallest	95.1	1.90
Star/galaxy	average	99.0	7.03
Bright	smallest	99.1	3.80
Star/galaxy	average	94.0	27.6
Dim	smallest	94.2	15.2
Iris	average	94.7	4.24
	smallest	96.6	2.20

Table 5: Prediction using smallest trees

6 Pruning SADT trees

The problem of generating small decision trees through choice of a stopping criterion or energy measure has been considered difficult. See, for example, [Breiman, *et al.*, 1984]. The approach commonly used for standard decision trees is to first generate a tree that correctly classifies every training example, and use another procedure, *pruning*, to reduce the size of the tree. See [Mingers, 1989] for an overview of several decision tree pruning methods. The results we present in our paper are for unpruned trees. It turns out that some pruning methods work well on our oblique trees. A comparison of pruning strategies for SADT can be found in [Heath, 1992].

7 Predicting using smallest trees

One advantage of randomization is that SADT can generate many different trees to model a dataset, and then use some additional criterion to decide which tree to use for classification. We believe that smaller trees provide a qualitatively better description of a domain, and we therefore used tree size as a criterion for picking trees. Table 5 shows a comparison of predictive accuracy and tree size for all four of our datasets using two criteria. Our default method, **average**, was to use SADT to generate approximately 200 different trees, and the results in the table give the average accuracy of all these trees. However, if we simply pick the **smallest** tree generated by the algorithm, performance improves slightly, as shown in the table. The benefit to this technique is that we can obtain smaller trees with no loss (and perhaps even an increase) in accuracy. (Each line in the table represents a 10–fold cross–validation study; therefore, accuracy and tree size are averages over 10 runs.)

8 Summary of Results

We have developed an algorithm for generating oblique decision trees, in which decisions are linear threshold

functions of example attributes. We have shown that for reasonable goodness measures, finding best splits is probably not possible in polynomial time.

This result differs from most previous NP–completeness results addressing learning complexity. Hyafil and Rivest (1976) showed that finding a decision tree that minimizes expected classification time is NP–hard. Blum and Rivest (1988) show that learning is difficult even when trying to train a neural network with only 3 nodes. Several other published results have considered the complexity of exact learning, namely, finding a circuit, formula or neural network consistent with a dataset. (See, e.g., Baum and Haussler, 1989; Blum and Rivest, 1988; Judd, 1988; and Lin and Vitter, 1991.) All of these results require at least a two level circuit or neural network, since the problem of learning with a single layer can be solved using linear programming. Unlike the previous results, which apply to exact learning, our result addresses approximate learning. We show that the approximate learning problem is NP-complete even for the simplest model of a neural mechanism: the perceptron.

This has forced us to examine methods of finding good (as opposed to best) splits. Our experiments have shown that simulated annealing works well as a method for generating oblique decision trees, in that it generates trees with fewer tests, while maintaining classification accuracy. We have compared our methods to others on both real and artificial data, in order to give a broad overview of how well the methods work. We plan to explore these techniques further by using different annealing methods and different goodness criteria. We are also developing randomized geometric algorithms that will be more efficient than annealing. In this paper, our goal was to test the thesis that smaller trees generate good decisions. This justifies the effort to seek efficient algorithms for finding good oblique decision surfaces in a decision tree.

If construction of a good classifier is more important than producing a concise representation, then one might use SADT to generate a set of trees, and then use the majority vote for classification. We have experimented with this and other ideas [Heath, 1992], but these results are not included in the interest of brevity.

To summarize, SADT was able, for the datasets we used in our experiments, to generate quite small trees. We consider this to be its main advantage over other decision tree algorithms.

Acknowledgements

The authors wish to thank David Aha for providing comments and relevant references. This research was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-89-0151, and by the National Science Foundation under Grant IRI-9116843.

References

[Baum and Haussler, 1989] E. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1:141–160, 1989.

[Blum and Rivest, 1988] A. Blum and R. L. Rivest. Training a 3-node neural network is np-complete. In *Proceedings of the First ADM Workshop on the Computational Learning Theory*, pages 9–18, Cambridge, MA, 1988.

[Breiman *et al.*, 1984] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. 1984.

[Duda and Hart, 1973] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[Heath, 1992] D. Heath. *A geometric framework for machine learning*. PhD thesis, The Johns Hopkins University, Baltimore, MD, 1992.

[Hyafile and Rivest, 1976] L. Hyafile and R. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[Judd, 1988] J. S. Judd. On the complexity of loading shallow neural networks. *Journal of Complexity*, 4:177–192, 1988.

[Lin and Vitter, 1991] J. H. Lin and J. S. Vitter. Complexity results on learning by neural nets. *Machine Learning*, 6:211–230, 1991.

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

[Mingers, 1989] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.

[Odewahn *et al.*, 1992] S. C. Odewahn, E. B. Stockwell, R. L. Pennington, R. M. Humphreys, and W. A. Zumach. Automated star/galaxy discrimination with neural networks. *Astronomical Journal*, 103(1):318–331, 1992.

[Quinlan,] J.R. Quinlan. Simplifying decisoin trees. *International Journal of Man-Machine Studies*, 27:221–234.

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

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

[Utgoff and Brodley,] P. Utgoff and C. Brodley. An incremental method for find multivariate splits for decision trees. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 56–65, Los Altos, CA. Morgan Kaufmann.

[Utgoff, 1989] P. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.

[Wiess and Kapouleas, 1989] S. Wiess and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proceedings of the Eleventh IJCAI*, Detroit, MI, 1989. Morgan Kaufmann.