Learning DNF by Decision Trees

Giulia Pagallo
Department of Computer and Information Science
University of California Santa Cruz, California 95064

Abstract

We investigate the problem of learning DNF concepts from examples using decision trees as a concept description language. Due to the replication problem, DNF concepts do not always have a concise decision tree description when the tests at the nodes are limited to the initial attributes. However, the representational complexity may be overcome by using high level attributes as tests. We present a novel algorithm that modifies the initial bias determined by the primitive attributes by adaptively enlarging the attribute set with high level attributes. We show empirically that this algorithm outperforms a standard decision tree algorithm for learning small random DNF with and without noise, when the examples are drawn from the uniform distribution.

1 Introduction

The goal of a system that learns from examples is to improve classification performance on new instances of some target concept, after observing a relatively small number of examples. A key technique is to formulate a simple explanation *(hypothesis)* of the training examples using a concept description language. Then, a new instance is classified according to the induced hypothesis.

The simplest way to describe an instance is by a measurement vector on a set of predefined attributes. An attribute comprises a (possibly) relevant characteristic of the target concept, and an attribute measurement reveals the state of the attribute in the observed instance. Such description is called *attribute-based.*

A frequently studied class of target concepts in attribute-based learning is the class of target concepts that can be represented by small Disjunctive Normal Form (DNF) formulae [Valiant, 1985, Haussler, 1988, Michalski, 1983]. While general purpose learning algorithms have been proposed for certain subclasses of simple DNF formulae [Valiant, 1985], the general problem is still open. In this paper, we investigate the problem of learning small DNF concepts using decision trees as the concept representation language [Breiman *et* a/., 1984, Quinlan, 1986].

Concepts with a small DNF description do not always have a concise decision tree representation when the tests at the decision nodes are limited to single attributes. The representational complexity is due to the fact that the tests to determine if an instance satisfies a term have to be replicated in the tree. We call this representational shortcoming the *replication problem.* The problem is reduced (and, eventually disappears), as we show in Section 3, by using high level attributes as tests in the tree. The complexity of classification rules derived from decision trees have been also addressed in [Breiman *et* a/., 1984, Quinlan, 1987b].

For learning, appropriate high level attributes may be defined a priori or dynamically by the learning algorithm. When the features are defined a priori, they are usually determined as combinations of the primitive attributes of a fixed type and size, e.g., conjunctions up to a fixed size [Breiman *et* a/., 1984, Valiant, 1985, Rivest, 1987]. In general, only a small number of features defined are meaningful for learning. A higher percentage of relevant features can be obtained, for example, by using knowledge about the concept at hand.

On the other hand, when the features are defined by the learning system, no background knowledge about the target concept is needed. The capability of a learning algorithm to enlarge adaptively the initial attribute set is called *dynamic bais* [Utgoff and Mitchell, 1982]. Genera) purpose learning systems with this capability have been proposed, for example, in [Schlimmer, 1986, Muggleton, 1987].

In this paper, we present a novel heuristic for a decision tree algorithm to modify the initial bias determined by the primitive attributes by dynamically introducing high level attributes. The learning algorithm we present (called FRINGE), builds a decision tree using the primitive attributes, and analyses this tree to find candidates for useful features. Then, it redescribes the examples using the new attributes, in addition to the primitive attributes, and builds a new decision tree. This process is iterated until no new candidate attributes are found.

The performance of FRINGE was tested in several synthetic Boolean domains. Our results show that FRINGE improves upon the classical decision tree approach when the target concept has a small DNF description, even in the presence of noise. However, when a target concept has no small DNF description, (e.g., the majority function) or when the attribute values are not correlated to the classification (e.g., the parity function), FRINGE cannot improve significantly on the poor performance of the classical approach.

2 Definitions

In this section we introduce the notation and definitions that formalize the basic notions we use in this paper. To simplify our presentation we limit ourselves to the case where all attributes are Boolean.

A *decision tree* is a tree with labelled nodes and edges. In our setting, an internal node defines a binary test on an attribute. For example, the root node of the tree in Figure 1 defines the test is "x_1 = 1?". Each edge represents an outcome of the test. We use the convention that the left edge represents the negative outcome, and the right edge represents the positive outcome. The label of a leaf represents the class label that is assigned by the tree to any example that reaches this node.

The *size* of a concept description with respect to a representation language is the number of bits required to write down the description. For simplicity, we mear sure the size of a decision tree by the number of internal nodes.

Finally, we introduce some terminology to refer to combinations of Boolean attributes. A *literal* is an attribute or its complement. A *term* is a conjunction of literals, a *clause* is a disjunction of literals. In general, a *feature* is any Boolean combination of the attributes obtained by applying Boolean operators to the primitive attributes. We use the term *variable* to refer to a primitive attribute or to a feature. We use •, + and (bar) to denote the Boolean operators *and, or* and *not* respectively.

3 Decision trees

The common procedure to learn a decision tree from examples is based on successive subdivisions of the sample. This process aims to discover sizable subsets of the sample that belong to the same class. The tree structure is derived by a top-down refinement process on the sample.

Initially, we begin with an empty decision tree and a sample set. A test is applied to determine the best attribute to use to partition the sample. Following [Breiman *et a/.*, 1984, Quinlan, 1986] , we measure the merit of an attribute with respect to the subdivision process by the *mutual information* between the class and the attribute. Then, the best attribute according to this measure is placed at the root of the tree, and the examples are partitioned according to their value on that attribute. Each subset is assigned to one subtree of the root node, and the procedure is applied recursively. The subdivision process on one branch of the tree ends when no attribute provides information about the class label, i.e., every attribute has zero mutual information. This leaf is labelled with the common class of these examples.

To avoid overtraining, the tree grown by the subdivision process described above is pruned back. We use the *Reduce Error Pruning* method proposed by Quinlan in [Quinlan, 1987b].

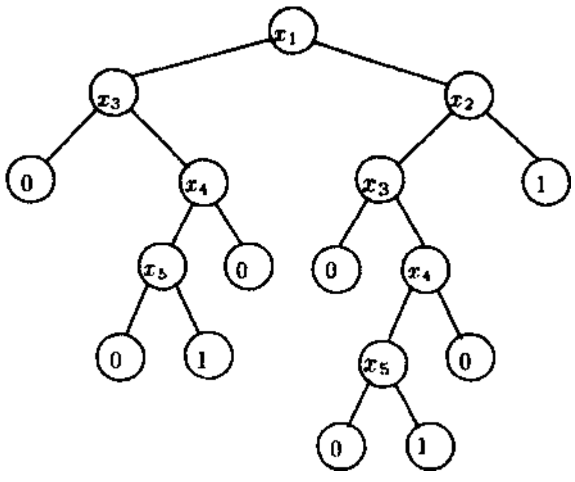


Figure 1: Smallest decision tree for $X_1 \cdot x_2 + x_3 \cdot x_4 \cdot x_5$.

4 The replication problem

A decision tree representation of a Boolean function with a small DNF description typically has a peculiar structure: the same sequence of decision tests leading to a positive leaf is replicated in the tree. To illustrate the problem, consider the Boolean function $x_1{}'X_2 + x_3\text{-}X_4\text{-}x_5$.

The smallest decision tree for the function has the following structure (see Figure 1): Each decision test in the rightmost path of the tree is an attribute of the shortest term in the formula. The path leads to a positive leaf and it corresponds to the truth setting for the first term. The left branch from each of the nodes is a partial truth assignment of the attributes that falsifies the shortest term. Then, to complete the function representation, we have to duplicate the sequence of decisions that determine the truth setting of the second term on the left branch of each of the nodes. We call this representational shortcoming the *replication problem.* In general, there are shared attributes in the terms, so some replications are not present, but the duplication pattern does occur quite frequently.

Due to the replication problem, while learning a decision tree, the partition strategy has to fragment the examples of a term into many subsets. This causes the algorithm to require a large number of examples in order to ensure that the subsets are large enough to give accurate probability estimates; otherwise the subdivision process branches incorrectly or terminates prematurely.

One way to solve the replication problem is to use conjunctions of the primitive literals at the decision nodes. To illustrate this solution, consider the two term formula discussed above. Assume we use the conjunction X_1 - x_2 as the test at the root node. Now, the right branch leads to a positive leaf that corresponds to the examples that satisfy the first term. The left branch leads to a representation of the second term (see Figure 2). We can represent this term by single attributes as before, but now the replication problem has disappeared. If, in addition, we have a feature for the second term, $x_3 \cdot x_4 \cdot x_5$, we obtain a decision with only two internal nodes.

But, how to define appropriate features while learning

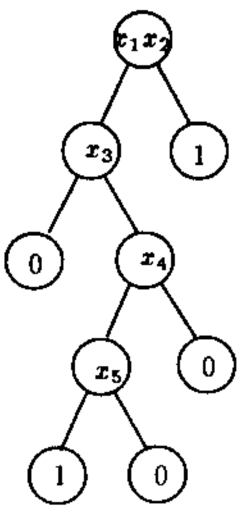


Figure 2: A representations for $x_1 \cdot x_2 + x_3 \cdot \bar{x}_4 \cdot x_5$ using features.

from examples? One approach is to determine them a priori as combinations of the primitive attributes of a fixed type and size [Breiman *et* a/., 1984, Rivest, 1987]. A second approach is to discover them after learning a decision tree. This schema has been used by Quinlan to infer production rules from a decision tree [Quinlan, 1987a]. Quinlan's method generates a production rule from each path in the decision tree, and then simplifies them by removing irrelevant conditions (attributes) in the rule.

There are two main differences between this approach and the FRINGE algorithm we define in the next section. First, Quinlan\s method is a simplification procedure, so to find an accurate set of production rules the initial decision tree has to already contain all the relevant patterns. In view of the replication problem, this may require a large tree and a large training sample. For FRINGE only some patterns have to be present in the initial tree, other patterns may surface later as the iteration proceeds. Second, Quinlan's method maintains rules for both classes when learning a single concept. This representation may not be as efficient as FRINGE'S representation for concepts with a small DNF description.

5 Learning Features

The FRINGE learning algorithm uses the following iterative schema to define appropriate features. The algorithm begins with a set *V* of primitive attributes, and creates a decision tree for a set of examples, choosing its decision variables from the set *V*. Then, a *find-feature* procedure generates new features as Boolean combinations of the variables that occur near the fringe of the tree. We describe this heuristic in more detail below. The set of new features is added to the variable set, the examples are reexpressed using this expanded set of variables, and the execution of the decision tree algorithm and the find-feature procedure is repeated. We will call a single execution of both processes an *iteration.* The iterative process terminates when no new features are added to the variable set, or a maximum number of variables is reached.

The find-feature heuristic was originally designed to discover relevant conjunctions of the target concept. To see this, observe that a decision tree defines an equivalent DNF expression. Each path from the root node to a positive leaf defines a term as follows: initially, set the term to the constant value 1; then, for each node in the path form the conjunction of the current term and the attribute at the node if the path proceeds to the right of the node, otherwise form the conjunction of the current term and the negation of this attribute.

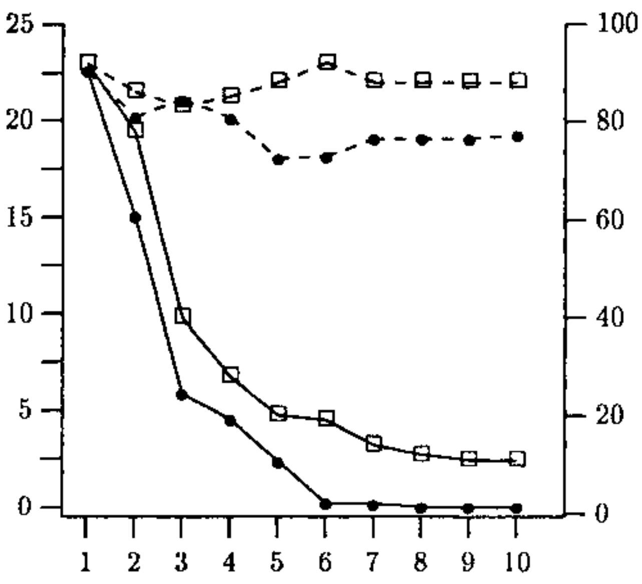
The *find-feature* procedure defines a feature for each positive leaf as follows: initialize the feature the constant value 1, and consider the path of length two from the leaf. Then, for each node on the path, form the conjunction of the current feature and the attribute at the the node if the path proceeds to the right, otherwise form the conjunction of the current feature and the negation of the attribute. A leaf node with distance less than two from the root does not define a feature. So, the find-feature heuristic simply defines features by applying the term formation rule to the fringe of the tree. For example, the find-feature heuristic applied to the decision tree in Figure 1 would produce the features: x_4 • x_5, x_4 • x_5, x_1 - X_2 (one feature appears twice). They correspond to the positive leaf nodes taken from left to right.

In each iteration, the find-feature heuristic forms very simple combinations: namely, conjunctions of two literals. The creation of longer terms, or more complex features, occurs adaptively through the iterative process. Initially, the variable set contains only the attributes. After the *k-i\\* iteration, the variable set may contain features of size up to 2*. For example, after the second iteration, a feature is either a term of size up to 4, or a conjunction of two clauses, each of size 1 or 2. Negated features include clauses of size up to 4, and disjunctions of two terms of size 1 or 2. In the limit, the find-feature procedure has the capability of generating any Boolean function of the literals, since the *negation* and *and* operators are a complete set of Boolean operators.

Figure 3 illustrates the learning performance results for a single execution of FRINGE on a small random DNF (dnf4, see section 6) and how it compares to a strategy where features are proposed at random. The random proposal heuristic works as follows. A feature is defined as the conjunction of two elements chosen at random from the current variable set. Moreover, the random proposal heuristic adds in each iteration the same number of features as FRINGE did. So, both methods work with the same number of variables in each iteration.

The graph shows the change in percentage error and in size of the hypothesis generated by the two methods as the iteration proceeds. The error is measured on a sample drawn from the uniform distribution independently of the training sample. More details on the experimental design are given in section 6.

The shape of the error and size graphs for FRINGE, given in Figure 3, are typical. After the first few iterations, a very accurate hypothesis is usually developed,



Iteration

Figure 3: Performance comparison between FRINGE and random as defined by: •, solid line: % error for FRINGE; •, dashed line : % error for random; D, solid line: hypothesis size for FRINGE; □, dashed line : hypothesis size for random. The right scale measures % error, the left scale measures the hypothesis size by the number of internal nodes

and the remaining few steps are used to reduce the size of the representation by introducing more meaningful features. Eventually, the process ends because the find-feature procedure cannot discover any new features, and any further iterations would produce the same hypothesis.

The random guess heuristic was not successful at all. The small fluctuations occur because by chance a few of the random combinations are significant.

Another interesting aspect of FRINGE'S learning behavior is the form of the final hypothesis. In this example, the test target concept has 10 terms in its smallest DNF description and the final tree has 10 decision nodes. The decision variable at each node is exactly one conjunction in the original target concept. Hence, the complexity of the final decision tree is the same as the complexity of the smallest DNF' representation for the concept. A similar behavior was observed for all DNF concepts, in the absence of noise, whose representation uses terms of approximately the same length. For DNF concepts with terms much longer than the average, the final hypothesis tends not to include them. However, the final hypothesis is still very accurate because the examples that only satisfy these long terms are rare when the examples are drawn from the uniform distribution.

To end this section, we discuss briefly how the find-feature procedure can be generalized to accommodate continuous attributes, and multiple concepts. In a decision tree a test for a continuous attribute defines a binary partition of the attribute range [Breiman *et a*/., 1984]. Therefore, it can be thought of as a binary attribute, and the find feature procedure can be applied without modifications.

So far, only the positive leaves determine candidate features. A dual heuristic can be defined for CNF (Conjunctive Normal Form) concepts by looking at the negative leaves, and by using disjunctions instead of conjunctions. Moreover, a symmetric heuristic could be obtained by applying the original find-feature procedure to the positive as well as to the negative leaves. This last heuristic can also be used in the multiple class case, by applying the feature formation rule to all leaves, regardless of their class label.

6 Experiments

The performance of a learning algorithm can be measured in terms of the classification accuracy on unseen instances from the target concept and in terms of the size of the final hypothesis. The goal of our experiments was to test how well the FRINGE algorithm does using these two criteria, and how well it compares with a decision tree algorithm with respect to the same criteria.

The algorithms were tested on five domains. They are: small random DNF, multiplexor, parity, majority and prob-disj. In addition, we tested the performance of the algorithms in the presence of random noise in both attributes and classification.

We present in Table 1 a concise description of the test functions by listing the total number of attributes, the number of terms, the average term length and the standard deviation from the average of the term length, in the smallest DNF description of the target concepts. The first four test functions (dnfl - dnf4) are small random DNF functions [Pagallo and Hausslcr, 1988], mult11 is multiplexor on 11 attributes [Wilson, 1987], par4 and par5 are parity four and five attributes respectively [Pagallo and Hausslcr, 1988], majll is majority out of 11 attributes [Subutai and Tesauro, 1988], and prdsj is a short hand for prob-disj [Quinlan, 1987b]. For all test functions except prdsj additional irrelevant attributes have been added.

We executed ten independent runs for each test function. The following design criteria for the sample sets were used for all domains except prdsj. For this domain we used the specifications given in [Quinlan, 1987b] to facilitate the comparison of results.

In each execution, the learning and testing tasks were performed on two sets of examples independently drawn from the uniform distribution. The *learning set* was randomly partitioned into two subsets, *training* and *pruning* sets, using the ratios 2/3 and1/3 [Breiman *et a*/., 1984]. The training set was used to generate a consistent hypothesis. The pruning set was used to reduce the size of the hypothesis and (hopefully) to improve its classification performance on new instances of the target concept.

Let *N* be the number of attributes and *K* be the number of literals needed to write down the smallest DNF description of the target concept. Let *e* be the percentage error that we wish to achieve during the testing task. The number of learning examples we used is given by the

Table 1: DNF description for test target concepts

function			*term length*	
id	*attributes*	*terms*	*avg.*	*dev.*
dnf1	80	9	5.8	0.4
dnf2	40	8	4.5	1.2
dnf3	32	6	4.8	1.0
dnf4	64	10	4.1	0.7
mult11	32	8	4.0	0.0
par4	16	8	4.0	0.0
par5	32	5	5.0	0.0
maj11	32	462	5.0	0.0
prdsj	10	3	3.0	0.0

following formula:

$$\frac{K * \log_2(N)}{\epsilon} \quad (1)$$

This formula represents roughly the number of bits needed to express the target concept, times the inverse of the error. This is approximately the number of examples given in [Vapnik, 1982, Blumer *et al.*, 1987] which would suffice for an ideal learning algorithm that only considers hypotheses that could be expressed with at most the number of bits needed for the target, concept, and always produces a consistent hypothesis. Qualitatively, the formula indicates that we require more training examples as the complexity of the concept increases or the error decreases. In our experiments we set e = 10%. We used 2000 examples to test classification performance.

Table 2 presents the results we obtained with the decision tree algorithm and FRINGE for each target concept. The table reports the average percentage error and the average tree size obtained over ten executions of the algorithms. The percentage error is the number of classification errors divided by the size of the test sample. The deviation of the actual results from the average error is within 7.3% for the decision tree algorithm and within 4.5%. for FRINGE. The size of the tree is measured by the number of internal nodes.

FRINGE discovered the exact concept for dnfl, dnf3, dnf4, multill and par4. The small error for dnP2 and dnf3 results from the fact that one or two terms in the concepts are much longer than the average length. So, examples that satisfy only the longer terms are very rare, when the examples are drawn from the uniform distribution. FRINGE did not include a description for the longer terms in the final hypothesis.

Majority and parity concepts, with a large number of inputs, are hard for decision trees. Majority is hard because there are a large number of terms in the smallest DNF representation of the concept, and hence there are a large number of terms is the smallest decision tree for the concept. Parity is hard when the examples are drawn from the uniform distribution because the attributes are not correlated with classification [Pagallo and Haussler, 1988]. FRINGE found an exact representation for par4, but it could not improve significantly on the poor performance of the decision tree algorithm for par5. Both methods performed poorly on majll.

Table 2: Performance comparison between FRINGE an Decision Tree

function	*avg.% error*		*avg. tree size*	
id	*DTree*	*FRINGE*	*DTree*	*FRINGE*
dnf1	12.4	0.0	48.7	9.0
dnf2	10.4	0.5	52.4	7.6
dnf3	7.4	0.3	45.4	6.1
dnf4	24.9	0.0	93.9	10.0
mult11	13.1	0.0	97.0	11.6
par4	38.3	0.0	125.3	4.9
par5	36.5	22.1	324.6	120.7
maj11	18.7	15.4	132.6	72.6
prdsj	22.8	11.4	21.9	7.1

Table 3: Performance comparison on noisy dnf2

	noise on attr		*noise on class*	
noise	*avg.% error*		*avg. % error*	
level	*DTree*	*FRINGE*	*DTree*	*FRINGE*
5%	15.8	10.3	13.9	6.0
10%	19.3	16.7	20.6	11.3
20%	27.8	27.1	32.7	28.5
40%	29.4	28.6	49.0	49.0

The results for prdsj are, to within experimental error, comparable to the results reported in [Quinlan, 1987b].

As an example of the sensitivity of the algorithms to noise we test them on the concept dnf2. Figure 3 summarizes the performance results for the noise experiments with different levels of class and attribute noise. The average percentage error was taken over five execution of each algorithm. A noise level of *n* percent means that, with probability *n* percent the true value was flipped. For attribute noise, each attribute was corrupted independently by the same noise level. In all experiments, the learning and testing sets were generated as for the noise free case, and the results are an average over five executions. The learning and testing sets were corrupted using the same type and level of noise. The table shows that FRINGE is more sensitive to attribute noise than class noise, as is the decision tree algorithm.

Another informative measure of the performance of a learning system is how the classification accuracy varies as a function of the number of learning instances. This variation is described by a learning curve. Figure 4 compares the learning curves for the decision tree algorithm and FRINGE on dnf4. The percentage error is measured on an independent test set, and it is the average of five executions for each algorithm. FRINGE finds an exact representation for the target concept using a learning set of 1980 (or more) examples. A sample size of 1980 examples is the size of the learning set predicted by formula (1) for t = 10%.

7 Conclusions

In this paper we view the problem of learning from examples as the task of discovering an appropriate vocabulary for the problem at hand. We present a novel algorithm based on this approach. The algorithm uses a decision

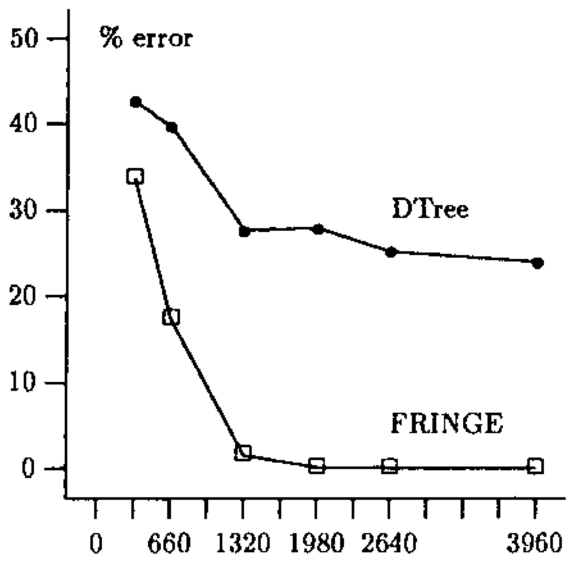


Figure 4: Learning Curves for dnf4

tree representation and it determines an appropriate vocabulary through an iterative process. The method accommodates target concepts that are described by discrete as well as continuous attributes, and it can be used to learn multiple concepts simultaneously.

We also show empirically that this method compares very favorably on Boolean domains to an implementation of a standard decision tree algorithm for noise-free as well as noisy problems. Since the difficulties of the decision tree method arise from a representational limitation, we expect that any reasonable implementation of this algorithm will give similar results.

Currently we are testing FRINGE on natural domains. So far, we have compared the methods on the mushroom [Schlimmer, 1986], hypothyroid [Quinlan, 1987b], and LED domains [Breiman *et ai,* 1984], the latter with 10% attribute noise. The hypotheses generated by the decision tree method were already very accurate (less than 0.78% error) in the first two domains, and they were close to optimal (26.0% error, [Breiman *et al,* 1984]) in the LED domain. The hypotheses generated by FRINGE were more concise and at least as accurate.

Acknowledgments

I would like to thank David Haussler for helpful discussions of the material presented here, and improvements to an earlier version of this paper. I also thank R. Buntine, T. Dietterich, J. Schlimmer and P. Utgoff for helpful comments to a technical report that preceded this paper. The support of the ONR grant N00014-86-K-0454 is gratefully acknowledged.

References

[Blumer *et al,* 1987] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letters,* 24:377-380, 1987.

[Breiman *et ai,* 1984] L. Breiman, J.H. Friedman, R.A. Olsen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth Statistic/Probability Series, 1984.

[Haussler, 1988] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence,* (36):177-221, 1988.

[Michalski, 1983] R. Michalski. A theory and methodology of learning. In *Machine Learning: An Artificial Intelligence Approach,* pages 83-134. Morgan Kaufmann, 1983.

[Muggleton, 1987] S. Muggleton. Duce, an oracle based approach to constructive induction. In *Proc. IJCAI-81,* pages 287-292. Morgan Kaufmann, 1987.

[Pagall o and Haussler, 1988] G. Pagallo and D. Haussler. Feature discovery in empirical learning. Technical Report UCSC-CRL-88-08, Dept. of Comp. Science, Univ. California, Santa Cruz, 1988.

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

[Quinlan, 1987a] J.R. Quinlan. Generating production rules from decision trees. In *Proc. IJCAI-87,* volume 1, pages 304-307. Morgan Kauffman, 1987.

[Quinlan, 1987b] J.R. Quinlan. Simplifying decision trees. *International Journal of Man-machine Studies,* 27:221 234, 1987.

[Rivest, 1987] R. Rivest. Learning decision lists. *Machine Learning,* 2:229 246, 1987.

[Schlimmer, 1986] J. C. Schlimmer. Concept acquisition through representational adjustment. *Machine Learning,* 1:81 106, 1986.

[Subutai and Tesauro, 1988] A. Subutai and G. Tesauro. Scaling and generalization in neural networks: A case study. In *Proc. IEEE-88 Conf. on Neural Inf. Proc. Systems - Natural and Synthetic,* 1988.

[Utgoff and Mitchell, 1982] P. Utgoff and T. M. Mitchell. Acquisition of appropriate bias for inductive concept learning. In *Proc. AAA 1-82,* pages 414-417. Morgan Kaufmann, 1982.

[Valiant, 1985] L. G. Valiant. Learning disjunctions of conjunctions. In *Proc. IJCAJ-85,* pages 560-566. Morgan Kaufmann, 1985.

[Vapnik, 1982] V. N. Vapnik. *Estimation of Dependencies Based on Empirical Data.* Springer Verlag, 1982.

[Wilson, 1987] S. W. Wilson. Classifier systems and the animat problem. *Machine Learning,* 2:199228, 1987.