Learning from Queries and Examples with Tree-structured Bias

Prasad Tadepalli
Department of Computer Science
Oregon State University
Corvallis, OR 97331
(tadepalli@cs.orst.edu)

Abstract

Incorporating declarative bias or prior knowledge into learning is an active research topic in machine learning. Tree-structured bias specifies the prior knowledge as a tree of "relevance" relationships between attributes. This paper presents a learning algorithm that implements tree-structured bias, i.e., learns any target function probably approximately correctly from random examples and membership queries if it obeys a given tree-structured bias. The theoretical predictions of the paper are empirically validated.

1 Introduction

Mitchell defined "bias" to be any prior knowledge that the learner has about the target function that enables it to generalize the examples beyond what it has already seen (Mitchell, 1980). Incorporating declarative bias into learning is an active research topic in machine learning, and has recently given rise to many useful algorithms, e.g., (Cohen, 1992), (Pazzani and Kibler, 1992), and (Shavlik and Towell, 1989). However, most of these algorithms are "general-purpose" or "weak" in that they are not designed to take advantage of the structure of the prior knowledge, and hence, do not always give good results. This paper presents a contrasting approach, which is to design special-purpose learning algorithms that effectively exploit *particular kinds* of declarative bias. In particular, it describes a learning algorithm for "determination knowledge," which is intended to capture the notion of "relevance" (Russell, 1989).

According to Valiant's Probably Approximately Correct (PAC) learning framework, one role of bias or prior knowledge in learning is to constrain or restrict the set of functions that the learner is required to learn (Valiant, 1988). The determination knowledge identifies all the attributes relevant to making predictions about a target attribute (Russell, 1989). The property of determination knowledge that is important to us here is that by identifying the relevant features for a target function, it reduces the set of functions that the learner has to consider, which in turn reduces the number of examples sufficient for learning (Mahadevan and Tadepalli, 1988).

Russell showed that the power of determinations in reducing the number of training examples is even greater when the learner has a set of determinations structured as a tree (Russell, 1989). Tree-structured bias consists of a tree of attributes such that each attribute at a node is determined by a small number of attributes represented as its children. The root of the tree represents the target attribute whose value the learner is trying to predict from the observable input attributes represented at the leaves of the tree. Only the leaves and the root of the tree are observable during training, which makes the learning problem difficult. Previous attempts to implement tree-structured bias had only limited success and restricted scope (Getoor, 1989).

This paper presents a polynomial-time learning algorithm that efficiently learns functions which obey a given tree-structured bias. The teacher provides random examples, and also gives the outputs for any inputs queried by the learner, i.e., answers "membership queries" (Angluin, 1988). The algorithm is proved correct using the PAC learning framework, and the time and sample complexities are estimated. The paper also presents empirical results that validate the theoretical predictions, illustrating the power of declarative bias and special-purpose learning algorithms designed to exploit it. We close with a discussion of related work and future research.

2 Tree-structured Bias

Russell identified a useful notion of bias called "determinations" which specifies the set of all relevant attributes to make predictions about a target attribute (Russell, 1989).

Let an object x be described by a set of input attributes $P_1, \ldots, P_n$, whose values are denoted by $P_1(x), \ldots, P_n(x)$. The task is to predict the value of the output attribute $Q(x)$.

Out of the many types of determinations Russell defined, in this paper, we are concerned with what may be called a "functional" determination (Russell, 1989). This is defined as follows.

Definition 1 *A set of attributes* $\{P_1, \ldots, P_k\}$ *determines* Q*, written as* $\{P_1, \ldots, P_k\} \succ Q$*, if there is a k-input function* f *such that,* $\forall x, Q(x) = f(P_1(x), \ldots, P_k(x))$.

Intuitively, the above determination says that some subset of attributes $P_1, \ldots, P_k$ are sufficient to predict the value of Q for any object x.

There are many examples of such determinations: the nationality of an individual might determine his/her language; the model, year, and make of a car determine its list price; the inputs of a combinatorial boolean circuit determine its output, etc.

The most important property of determinations that is relevant to us is that they reduce the set of functions that the learner has to consider by identifying the relevant features. If nationality of a person determines his/her language, then the learner may not consider functions which map different people of the same nationality to different languages, since they are not consistent with this determination. Since learning occurs by filtering out functions which are not consistent with the training examples, the more constraining the determination, the fewer the examples needed to filter the remaining functions.

Russell introduced "tree-structured bias" to further reduce the number of examples needed for learning (Russell, 1989). Tree-structured bias consists of a tree of attributes such that each attribute at a node is determined by at most a small constant number (k) of other attributes, which are represented as its children. The tree of attributes is called the "determination tree," and is given as prior knowledge to the learner. At the root of the determination tree is the "target attribute," whose value is to be predicted, and at the leaves are the "input attributes," whose values are given. The non-root internal nodes or "intermediate" nodes represent "intermediate" attributes, whose values are not observable during the training or testing.

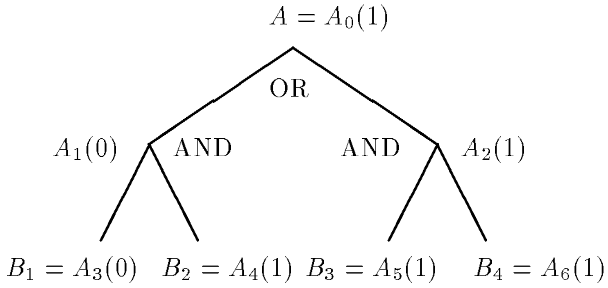

Figure 1: A Tree-structured Bias

If i is a node in the tree structure T, let A_i represent the attribute at i, and let $i_1, \ldots, i_k$ represent the children of i. $A_i(x)$ represents the value of A_i for the object x. $A = A_0$ denotes the target attribute. T_i represents the subtree of T rooted at node i. We also denote the n input attributes by $B_1, \ldots, B_n$. The input attributes at the leaves of T_i are called the *terminals* of T_i.

For each internal node i, $\{A_{i_1}, \ldots, A_{i_k}\} \succ A_i$, which means that there exist a set of k-input functions f_i such that $\forall x, i, A_i(x) = f_i(A_{i_1}(x), \ldots, A_{i_k}(x))$.

In Figure 1, the numbers in the parentheses represent the values of the attributes for a particular example when the target function is OR-OF-ANDS = (OR (AND B_1 B_2) (AND B_3 B_4)). Hence, $f_1 = f_2 = AND$, and $f_0 = OR$.

Note that without loss of generality we can ignore the internal nodes with a single child. This is because the determinations $\{P_1\} \succ Q_1$, and $\{Q_1, \ldots Q_j\} \succ R$, can be replaced with $\{P_1, Q_2, \ldots Q_j\} \succ R$, getting rid of the attribute Q_1 with a single child without increasing the number of children of any node. Hence, we assume that each internal node in the determination tree has 2 to k children.

3 The PAC Learning Framework

In this paper, we will be using a version of Probably Approximately Correct (PAC) learning, introduced by Valiant, and extended by Angluin (Valiant, 1984), (Angluin, 1988). For a broad introduction, see (Natarajan, 1991).

We restrict ourselves here to learning boolean functions. An *assignment* is a mapping of attributes $B_1, \ldots, B_n$ to their values in $\Sigma = \{0, 1\}$. Σ^n denotes the set of binary strings of length n. Each string in Σ^n represents an assignment, where the i^{th} bit of the string is the value of B_i. Let $\mathcal{F}_n$ be the set of functions from Σ^n to Σ that we are interested in learning. The set $\mathcal{F} = \{\mathcal{F}_1, \ldots, \mathcal{F}_n, \ldots\}$ is called a *function space.*

We assume an oracle called INSTANCE which takes no arguments and generates a random example (assignment) $x \in \Sigma^n$ according to some fixed, but unknown, probability distribution P. In other words, the probability of a particular example x being generated by a call of INSTANCE is $P(x)$.

We also assume another oracle called MEMBER which takes an assignment x as input and outputs $F(x)$, where $F \in \mathcal{F}_n$ is the target function. We call this a membership oracle.

Definition 2 *For a function space $\mathcal{F}$, if there is an algorithm $\mathcal{M}$ that, given an error parameter ϵ, a confidence parameter δ, and the problem size n, for all target functions $F \in \mathcal{F}_n$ and all probability distributions P,*
(i) makes polynomially many calls to INSTANCE and MEMBER, in n, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$,
(ii) runs for time polynomial in n, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and the minimum length of F, and
(iii) with probability at least $1 - \delta$ outputs a function $G \in \mathcal{F}_n$ such that, $\sum_{x \in S} P(x) \leq \epsilon$, where $S = \{x | x \in \Sigma^n and F(x) \neq G(x)\}$

then $\mathcal{M}$ is said to PAC-learn *$\mathcal{F}$ from queries and examples.*

The parameter ϵ specifies the error of the function G when compared to the real function F the learner is trying to approximate. The error is measured by the probability that F and G differ on some example chosen randomly using the same distribution P that was used during the learning. Since the approximation is obtained using randomly chosen training examples, they might sometimes be unrepresentative, in which case the approximation learned from them may not be sufficiently accurate on representative test examples. A learning algorithm must ensure that the probability of this event is lower than the confidence parameter δ.

Definition 3 *A function G is* **consistent** *with the target function F on a set of examples S if $\forall x \in S, F(x) = G(x)$.*

Typically, learning algorithms work by finding a function which is consistent with a sufficiently large set of random examples. The results in PAC learning framework suggest that the sample size sufficient to learn a given space of functions using only random examples varies logarithmically with the number of functions in the function space (Blumer et al., 1987), (Natarajan, 1991). Russell used these results to derive an upper bound on the number of random examples sufficient to learn a function consistent with a given tree-structured bias (Russell, 1989). Since we also use queries in addition to random examples, we derive our algorithm-specific results from first principles, which also gives us a better upper bound on the number of random examples.

4 The Learning Algorithm

The learning algorithm works top-down on the determination tree, learning truth tables for the boolean functions that correspond to the internal nodes of the tree. As it recurs down the determination tree, it also constructs an effective membership oracle for each of the relevant internal nodes.

The main problem in learning with a tree-structured bias is that the intermediate attributes in the determination tree are not observable. The basic idea behind our algorithm is that an intermediate attribute A_i can be made effectively observable by finding inputs for all input attributes except for the terminals of T_i, such that the value of A_i for any input is always the same as or always the opposite of the value of the root attribute. Such an input vector is part of what is called *a distinguishing assignment $d(i)$* for the node i. A distinguishing assignment for a node i also makes it possible to set the value of A_i to 1 or 0 by assigning the terminals of T_i to some values.

We will see that if the algorithm is not successful in generating the distinguishing assignment for a node i using a sufficiently large sample, then, with a high probability, A_i can be ignored without losing approximate correctness in predicting the target attribute A. The rest of this section describes the algorithm and proves it correct.

We are interested in learning the set $\mathcal{F}_n$ of functions from the values of the terminals $\{B_1, \ldots, B_n\}$ of a given determination tree T to the value of its target attribute A.

The **projection** of an assignment x onto a subtree T_i, denoted by x_i, also called a **partial assignment**, is a mapping of the terminals of T_i to their values under x.

If x and y are two (partial) assignments, $x + y$ is a (partial) assignment that maps any attribute B_i to $B_i(y)$ when it is defined and to $B_i(x)$ otherwise.

Note that $x+y$ is defined exactly for those attributes which are defined for either x or y. When an attribute is defined for both x and y, the value under y overwrites that under x. Hence the operation '+' is transitive, but not commutative in general.

We use z to denote a special assignment where every input attribute is set to 0. Without loss of generality, for any target function F, we assume that the values of all intermediate attributes are 0 when the terminals of the corresponding subtrees are all set to 0, i.e., $A_i(z_i) = 0$ for all intermediate nodes i.

For a target attribute A, and an intermediate node i, an assignment x such that $A(x) \neq A(x + z_i)$, is called a **distinguishing assignment** for i, and is denoted by $d(i)$.

A distinguishing assignment $d(i)$ gives us the abilities to *control* and *observe* the value of A_i. Control, i.e., setting A_i to 1 or 0, is possible because merely changing the values of terminals of T_i can change the target attribute value only if it also changes A_i's value. Since $A_i(z_i) = 0$, $A_i(d(i)_i) = 1$.

Observing the output of A_i for any arbitrary input w_i, i.e., simulating the membership oracle for A_i, is possible since switching the value of A_i switches the target attribute value in the context of $d(i)$. To observe $A_i(w_i)$, we set the terminals of T_i according to w_i, and all other inputs according to $d(i)$. We then observe the value of A, and compare it with that obtained by changing the settings of the terminals of T_i to z_i. If the two outputs are the same, then $A_i(w_i) = A_i(z_i) = 0$, otherwise $A_i(w_i) = 1$.

For the target function OR-OF-ANDS in Figure 1, the assignment 0111 is a distinguishing assignment for node 2. This is because $A(0111) \neq A(0100) = 0$. Hence, A_2 can be set to 1 by setting its terminals to 11. Its value on any input can be observed because it is always the same as the value of A after setting the terminals of A_1 to 01.

All internal nodes i need not have distinguishing assignments. A node is called *live* for a given target function, if it has a distinguishing assignment. If a node i is not live, then, $A(x) = A(x + z_i)$ for all assignments x, which means that replacing A_i with the constant function **0** would not effect the value of A for any input. A non-live node is called *dead.*

The recursive algorithm Learn-Tree (see Table 1) takes a random example set S, a subtree T_i of the determination tree, and a distinguishing assignment for node i as inputs. It returns an approximation G_i to the target function A_i. Given a distinguishing assignment $d(i)$ for the node i, the algorithm searches for distinguishing assignments $d(i_j)$ among all possible $d(i) + c_i$, where c_i's are obtained by setting the terminals of some subtrees T_{i_j} of T_i to the corresponding projections of some example e, and setting the terminals of all other subtrees to 0's. All such assignments for a given node i and example e are called the *corners* C_i of e at node i. In other words, $C_i = \{d(i) + c_{i_1} + \ldots + c_{i_k} \mid c_{i_j} \in \{e_{i_j}, z_{i_j}\}\}$.

If Learn-Tree cannot find a distinguishing assignment for some node i_j, it is assumed to be dead, and the attribute A_{i_j} is set to the constant function **0**. If it can find a distinguishing assignment for a node i_j, then it has, in effect, a membership oracle for the attribute A_{i_j}. Hence, it can call itself recursively on each node i_j with a known distinguishing assignment.

For example, consider the random training example 0111 for the target function OR-OF-ANDS of Figure 1. The corners of 0111 at node A_1 are $\{0111, 0011, 0100, 0000\}$. Since $A(0111) \neq A(0100) = 0$, 0111 (or 0011) is recognized as a distinguishing assignment for node 2. Given no other training example, node 1 does not have a distinguishing assignment, and is assumed to be dead. If another example, say 1100, is provided, then it will be recognized as a distinguishing assignment for node 1.

After finding distinguishing assignments, at each node i, the algorithm constructs a partial truth table for the function A_i, denoted by g_i, by generating all possible bit vectors $b_1, \ldots, b_k$, at nodes $i_1, \ldots, i_k$, such that $b_j = 0$ whenever $d(i_j)$ is not known. It uses the partial assignment $c_{i_j} = d(i_j)_{i_j}$ to make the bit $b_j = A_{i_j}(c_{i_j}) = 1$, and uses $c_{i_j} = z_{i_j}$ to make it 0. Then, $g_i(b_1, \ldots, b_k)$ is assigned to $A_i(c_{i_1} + \ldots + c_{i_k})$, which is inferred using a simulated membership oracle for A_i.

In the above example, assuming that no distinguishing assignment has been found for A_1, the first input of g_0 is always assumed to be 0. The truth table of g_0 will only consist of two entries: $g_0(00) = A(0100) = 0$. and $g_0(01) = A(0111) = 1$.

It then calls itself recursively on each node i_j with a known distinguishing assignment by passing itself the distinguishing assignment $d(i_j)$ and the projections of the examples in S onto T_{i_j}. The recursion terminates at the leaves of the tree and returns the values of the input attributes.

In our example, Learn-Tree will be called on node 2 with the distinguishing assignment 0111 and the single input 11. 0111 is found to be the distinguishing assignment for both the nodes 5 and 6. All possible inputs are generated at nodes 5 and 6 to generate the truth table for A_2. The output of A_2 on any input is the same as the output of A with the first two input bits set to 01. For example, $g_2(00) = A(0100) = 0$; $g_2(11) = A(0111) = 1$; Finally, the program terminates at the leaf nodes 5 and 6 by returning the values of the input attributes.

Hence, given a single example 0111, the program learns the function (AND B_3 B_4) for A. This gives incorrect output on 3/16 cases. Given two examples $\{0111, 1100\}$, it will be able to distinguish node 1 as well, and learns the OR-OF-ANDS function.

Theorem 4 *The function output by Learn-tree is consistent with the target function on the input example set S.*

Proof: The proof is by contradiction. Note that if an algorithm finds a distinguishing assignment for a node, it must be correct. However, the algorithm assumes that any node for which it has not found a distinguishing assignment to be dead and stops the recursion at that level. The only inconsistencies between the target function and the learned function are due to this assumption.

Assume that the output function is not consistent with some example $e \in S$. Hence, some node in the determination tree which is assumed to be dead must be responsible for the inconsistency. Let i be one of the farthest nodes from the root such that the distinguishing sequence $d(i)$ has been found, but some of its children are assumed dead. Without loss of generality, let $Q_i = \{i_1, \ldots, i_l\}, l \leq k$ be the set of children of i assumed dead. Since i is one of the farthest such nodes, all the live children of i would be free of dead descendents, and hence their values always match those of the target function.

When called on the subtree T_i, Learn-Tree would have looked for distinguishing assignments for nodes in Q_i in the set $C_i = \{d(i) + e_i + c_{i_1} + \ldots + c_{i_l} \mid c_{i_j} \in \{e_{i_j}, z_{i_j}\}, 1 \leq j \leq l\}$, and failed. This means that

$A(d(i)+e_i) = A(d(i)+e_i+z_{i_1}) = A(d(i)+e_i+z_{i_1}+z_{i_2}) = \ldots = A(d(i)+e_i+z_{i_1}+\ldots+z_{i_l})$.

Since changing the value of A_i always changes the target attribute value in the context of $d(i)$, it follows that $A_i(e_i) = A_i(e_i + z_{i_1} + \ldots + z_{i_l})$.

In other words, assuming that the values of A_{i_1} through A_{i_l} are 0 does not change the value of A_i, and hence, none of the attribute values of ancestors of i. We can now remove nodes i_1 through i_l from possible reasons for the inconsistency, and repeat the same argument for the remaining such nodes. This means that none of the nodes assumed dead is responsible for the inconsistency, which contradicts our initial observation. •

Procedure **Learn-Tree** (S, T_i, $d(i)$)

; Takes the random example set S,
; a subtree T_i of the determination tree, and
; a distinguishing assignment $d(i)$ of i as inputs, and
; finds an approximation G_i to A_i.

; Terminal condition

  If i is a leaf node, Then return a function $G_i = A_i$

; Find distinguishing assignments for i's children

  For each example $e_i \in S$
    $C_i := \{d(i) + c_{i_1} + \ldots + c_{i_k} \mid c_{i_j} = e_{i_j}$ if $d(i_j)$
      has already been found, else $c_{i_j} \in \{e_{i_j}, z_{i_j}\};\}$
    For each i_j s.t. $d(i_j)$ has not been found
      If $\exists x, y \in C_i$ s.t. $y = x + z_{i_j}$, and $A(x) \neq A(y)$*
        Then $d(i_j) := x$;
    End For;
  End For;

; Generate the truth table for i, denoted by g_i

  For each possible bit vector $\langle b_1, \ldots, b_k \rangle$ s.t.
      $b_j = 0$ whenever $d(i_j)$ is not defined
    $g_i(b_1, \ldots, b_k) := A_i(c_{i_1} + \ldots + c_{i_k})$*,
      where, $c_{i_j} = d(i_j)_{i_j}$ if $b_j = 1$, and
        $c_{i_j} = z_{i_j}$ if $b_j = 0$;
  End For;

; Recursively call Learn-Tree on subtrees of T_i

  For each child i_j of i
    If $d(i_j)$ is defined Then
      G_{i_j} := Learn-Tree($\{e_{i_j} \mid e_i \in S\}$, T_{i_j}, $d(i_j)$);
    Else G_{i_j} := The constant function **0**;
  End For;

Output $G_i := \langle g_i, \langle G_{i_1}, \ldots, G_{i_k} \rangle \rangle$;
End **Learn-Tree**;

* $A_i(w_i)$:= If A_i is the target attribute A
      Then MEMBER(w_i)
      Else If MEMBER($d(i) + w_i$) =
          MEMBER($d(i) + z_i$)
        Then 0 Else 1;

Table 1: Learning Algorithm for Tree-structured Bias

Theorem 5 *If the number of input attributes in a determination tree is n, then Learn-Tree PAC-learns the function class consistent with a determination tree from queries and examples if it is called with a random example set of size $m \geq \frac{1}{\epsilon}(\frac{n}{2}\ln 2 + \ln\frac{1}{\delta})$.*

Proof: Note that we are only interested in finding a function G such that the probability of its disagreeing with the target function on a randomly chosen example $\sum_{G(x)\neq F(x)} P(x) < \epsilon$. Call all other functions 'bad'.

Since the only discrepancies between the target function and the learned function are due to assuming that some nodes are not live, for a given target function, the function output by *Learn-Tree* is determined by the set of nodes in the determination tree assumed to be dead.

Denote the set of internal nodes which are parents of leaf nodes in the determination tree by L. Since each internal node has at least degree 2, $|L| \leq \frac{n}{2}$. Note that the number of potential subsets of internal nodes assumed to be dead by *Learn-Tree* is bounded by the number of possible subsets of L. A node which is not in L being dead is equivalent to all its descendents in L being dead.

Let $b \subseteq L$ be a set of internal nodes of the determination tree which are assumed to be dead by *Learn-Tree* while learning a target function F. Let G be the function learned. Assume that G is 'bad,' so that for any example e, $P(G(e) = F(e)) \leq (1-\epsilon)$. However, by Theorem 4, G is consistent with all m examples in S. Since the training examples are chosen independently randomly, the probability of this event is $\leq (1-\epsilon)^m$. Since there is potentially a function G for each subset $b \subseteq L$, the probability of some bad function being consistent with all the random examples is upper bounded by $2^{|L|}(1-\epsilon)^m$.

If we choose m such that $2^{|L|}(1-\epsilon)^m < \delta$, we can ensure that our algorithm does not output a bad function with probability greater than δ. This can be done by making $m \geq \frac{1}{\epsilon}(|L|\ln 2 + \ln\frac{1}{\delta})$.

Since $|L| \leq \frac{n}{2}$, it suffices to bound the number of examples by $\frac{1}{\epsilon}(\frac{n}{2}\ln 2 + \ln\frac{1}{\delta})$.

In the worst case, to find the distinguishing assignments, one query is generated for each possible corner of each example at each internal node. In addition, a query may be generated to fill each entry in the truth table at each internal node. Hence the number of queries is bounded by $(n-1)2^k(1 + \frac{1}{\epsilon}(\frac{n}{2}\ln 2 + \ln\frac{1}{\delta}))$. Since this is also the most expensive part of the algorithm, its complexity is polynomial in n, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, satisfying all our requirements. •

5 Experimental Results

The above algorithm was implemented as a program called TSB. There were two previous implementations of tree-structured bias (Getoor, 1989). However, both of them are inadequate to compare with TSB. One of them assumes that the boolean functions at each node are monotone. The other is a neural network implementation which uses Backprop for learning and is too slow to be used on trees of a few dozen nodes. Hence TSB is compared with with ID3, which is knowledge-free and hence cannot be expected to perform as well (Quinlan, 1986). However, the comparison is useful to demonstrate the power of tree-structured bias and special-purpose learning algorithms designed to exploit it.

TSB was tested on a determination tree with $k = 3$. For concreteness, we can think of the target concept as a combinatorial boolean circuit, with a maximum fan-in $k = 3$. The circuit has $n = 20$ inputs. Each training episode consisted of generating a random circuit consistent with the determination tree, training it on a set of random examples (selected using the uniform distribution), and testing it on 25, separately generated, test examples. Results were averaged over 50 randomly generated circuits for each random training sample size, which varied from 0 to 50 with increments of 5.

Figure 2 shows the relationship between the size of the random training sample and the number of queries generated by TSB. The number of queries increased with the size of the random sample, but not too sharply. The total number of queries raised from 8, with no random examples, to 557, with 50 random examples. However, using a hash table that caches the results of previous queries reduced the number of queries considerably. The number of unique queries raised to 158 with 45 random examples and then declined slightly to 148 with 50 random examples. This shows that the number of unique queries, although larger than the number of random examples, is still quite reasonable.

Figure 3 shows the relationship between the number of unique queries and the accuracy on the test sample for TSB. It also shows the performance of ID3 on a training example set which was collected from the unique queries asked by TSB. Hence ID3 and TSB used the same training data, although ID3 did not have the determination knowledge.

Each successive point on the two curves of Figure 3 represents a random training sample of size 5 more than the previous point. With no random examples at all, both the programs learn a function which al-

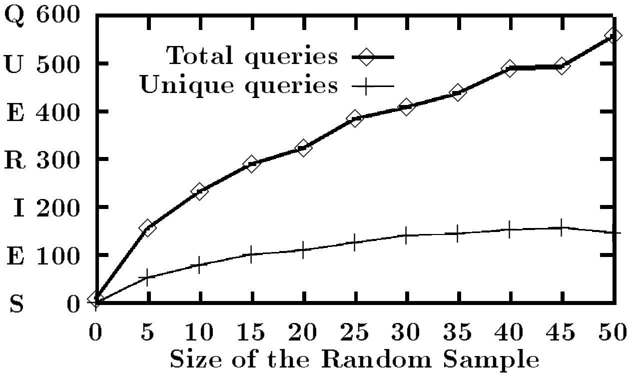

Figure 2: Growth of queries with random examples

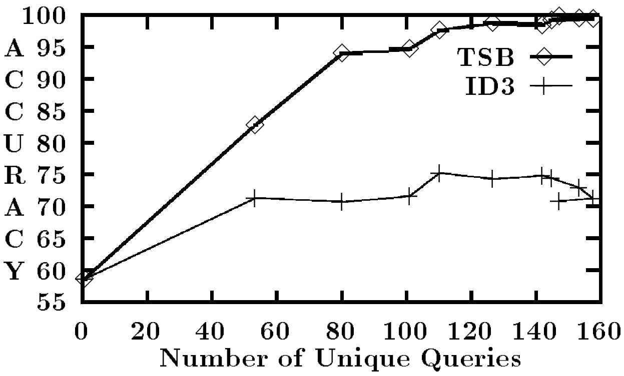

Figure 3: Learning curves: TSB vs. ID3

ways outputs a 0. That gives 60% accuracy in this domain. In TSB, the accuracy reached 94% with approximately 80 (unique) queries (corresponding to 10 random examples). The prediction on the test sample was more than 99% accurate after making 145 queries (corresponding to 35 random examples). In contrast, ID3's performance only increased to a maximum of 75% with 111 examples and then declined with more training. ID3 was run without pruning. C4.5 with information gain heuristic and with pruning performed similarly (not shown here). This shows that TSB effectively exploits its tree-structured bias, and ID3 suffers from not being able to take advantage of this knowledge.

As can be expected, the experimental results of TSB were also much better than the worst-case theoretical predictions. For the above domain, to achieve a 90% accuracy with 90% probability ($\epsilon = \delta = 0.1$), the theory predicts upper bounds of 92 random examples and 13,984 queries, which are much higher than the experimentally determined values of ≈ 10 and ≈ 230 respectively.

6 Discussion and Related Work

It appears that both the oracles used by our algorithm are necessary to implement tree-structured bias. Learning with tree-structured bias from classified random examples is reducible to learning arbitrary boolean formulas (Pitt and Warmuth, 1990), which in turn is reducible to some apparently hard cryptographic problems (Kearns and Valiant, 1989). To see that tree-structured bias cannot be implemented with membership queries alone, consider the class of boolean functions which output 1 on exactly one input and output 0 on all others. This function class can be represented with a tree-structured bias. However, it cannot be learned with membership queries alone, because the program does not get any information unless it chances upon the single input that gives rise to an output of 1. In the worst case, this involves trying each possible input, which is not allowed.

Our algorithm shares some features with the read-once formulas studied in the Computational Learning Theory literature, e.g., (Hancock and Hellerstein, 1991). Read-once formulas are boolean formulas where each variable appears exactly once. If we write down the function described by a determination tree using the functions $f_1, \ldots, f_n$ that correspond to the internal nodes of the tree, each input feature appears exactly once in such a formula. However, read-once formulas only allow a 'base' set of functions such as AND, OR and NOT (and perhaps a few more) to appear in them, and do not allow arbitrary functions described by truth tables. The algorithms for learning read-once formulas first learn the tree structure, and then adopt methods similar to Learn-Tree to learn the functions at the nodes in the tree.

Angluin studied the problem of exact identification of a target concept from membership and equivalence queries (Angluin, 1988). Equivalence queries are in the form of "is the target function the same as G?" If G is the target function, the teacher answers 'yes'; otherwise the teacher responds with a counter-example, i.e., an example where G and the target function disagree. We can design a learning algorithm that exactly identifies the target function for tree-structured bias from membership and equivalence queries using Learn-Tree as a subroutine. In each iteration, the top-level algorithm runs Learn-Tree on all the counter-examples provided by the teacher thus far. It then asks an equivalence query on the learned function. Either the function is correct, or the teacher provides yet another counter-example, and the cycle repeats. Each iteration makes at least one non-live node live, because otherwise the counter-example would have been classified correctly by the previously learned function. Hence the process terminates in at most n iterations.

7 Conclusions and Future Work

This paper is another attempt to bridge the gap between the theoretical tools developed in the computational learning theory, and the problems addressed and ideas developed in the experimental machine learning. We saw that the idea of learning from examples and membership queries can be utilized to learn effectively when the function space is constrained by declarative knowledge in the form of a tree of determinations. This illustrates the power of tree-structured bias in constraining learning, and the effectiveness of special-purpose learning algorithms in exploiting this bias.

One limitation of our results is that they do not generalize to determination *graphs*, where a node might have multiple parents. Multiple parents make it difficult or impossible to independently control their attribute values and find their distinguishing assignments. It is also unclear, whether the determination tree itself can be learned, and under what conditions. The work on read-once formulas is interesting in this context. Another generalization is to allow restricted classes of non-boolean functions at the nodes of the tree. It would also be interesting to look for practical applications of tree-structured bias to a real world task.

Acknowledgments

This research was supported by the National Science Foundation under grant number IRI:9111231. I thank Hussein Almuallim, Sridhar Mahadevan, Balas Natarajan, Stuart Russell, and Manfred Warmuth for interesting discussions on this topic. I thank Sridhar Mahadevan, Chandra Reddy, and the reviewers of this paper for many useful comments on previous drafts.

References

Angluin, D., Queries and concept learning, *Machine Learning*, 2, 1988.

Blumer, A. Ehrenfeucht, A., Haussler, D., and Warmuth, M. K., "Learnability and the Vapnik-Chervonenkis Dimension," *Journal of the ACM*, 36:4, 1989.

Cohen, W. W., Compiling Prior Knowledge Into an Explicit Bias, In Proceedings of Machine Learning Conference, Morgan Kaufmann, 1992.

Getoor, L., *The instance description: How it can be derived and the use of its derivation.* MS report, Computer Science Division, University of California, Berkeley, CA, 1989.

Hancock, T. and Hellerstein, L., Learning Read-Once Formulas over Fields and Extended Bases, In Proceedings of the Fourth Annual Workshop on Computational Learning Theory, Morgan Kaufmann, 1991.

Kearns, M. and Valiant, L. G., Cryptographic Limitations on Learning Boolean Formulae and Finite Automata, In Proceedings of 21^{st} Annual ACM Symposium on Theory of Computing, Assoc. Comput. Mach., 1989.

Mahadevan, S. and Tadepalli, P., On The Tractability of Learning from Incomplete Theories, In Proceedings of the Fifth International Machine Learning Conference, Morgan Kaufmann, 1988.

Mitchell, T. The Need for biases in learning generalizations. In *Readings in Machine Learning* (Ed) Shavlik, J. and Dietterich, T. G., Morgan Kaufmann, 1990.

Natarajan, B., *Machine Learning: A Theoretical Approach*, Morgan Kaufmann, 1991.

Pazzani, M. and Kibler, D., The Utility of Knowledge in Inductive Learning, *Machine Learning*, Kluwer Academic, 9:1, 1992.

Pitt, L. and Warmuth, M. K., Prediction Preserving Reducibility, *Journal of Computer and System Sciences*, Academic Press, 41:3, 1990.

Quinlan, R., Induction of Decision Trees, *Machine Learning*, Kluwer Academic, 1:1, 1986.

Russell, S., *The Use of Knowledge in Analogy and Induction.* Morgan Kaufmann, 1989.

Shavlik, J. W. and Towell, G. G., An Approach to Combining Explanation-Based and Neural Learning Algorithms, *Connection Sciences*, 1:3, 1989.

Valiant, L. G., A Theory of the Learnable, *Communications of the ACM*, 27:11, August 1984.