

Simple Learning Algorithms Using Divide and Conquer

Nader H. Bshouty*

Department of Computer Science

The University of Calgary

Calgary, Alberta, Canada T2N 1N4

e-mail: bshouty@cpsc.ucalgary.ca

Abstract

This paper investigates what happens when a learning algorithm for a class C attempts to learn target formulas from a different class. In many cases the learning algorithm will find a “bad attribute” or a property of the target formula which precludes its membership in the class C . To continue the learning process we proceed by building a decision tree according to the possible values of this attribute (divide) and recursively run the learning algorithm for each value (conquer).

We demonstrate that the application of this idea on some known learning algorithms can both simplify the algorithm and provide additional power to learn more classes.

1 Introduction

What would happen if we were to run a learning algorithm designed to learn concepts from a class \mathcal{I} on a concept J from a class of concepts $\mathcal{J} \supset \mathcal{I}$? The algorithm may locate an attribute of J that impedes the learning process. One way to solve this problem would be to somehow eliminate this attribute and hope that this will force the target into the class \mathcal{I} . One possible elimination method would be to build a decision tree according to the possible values of this attribute and then recursively run the learning algorithm for each of these values. If the classes \mathcal{J} and \mathcal{I} are “closely” related then a polynomial size decision tree with leaves from \mathcal{I} may be all that is required to learn J . Using this divide and conquer approach, we present some learning algorithms for classes that are “closely” related to some class that is learnable.

*This research was supported in part by the NSERC of Canada.

The above idea and others are used in this paper to prove the learnability of concepts in different learning models. The models considered here are learning from membership queries (interpolation), the PAC-learning model with and without membership queries [Val84,A88], the exact learning model from membership and equivalence queries [A88], the exact learning model from equivalence queries only and the statistical query learning model [K93].

This simple approach seems to be a powerful tool for proving the learnability of many classes in different learning models. In the next section we list the results of this paper.

2 Results

The results of the paper include:

- A polynomial time learning algorithm with membership and equivalence queries for the class of unate DNF with the addition of constant number of any other terms and the addition of $O(\log n)$ nonunate variables to the terms. The hypotheses to the equivalence queries are polynomial size DNF.

This result is an improvement of the result in [Bs93] for learning almost monotone DNF, i.e., monotone DNF with the addition of constant number of non-monotone terms. In [Bs93] the hypotheses to the equivalence queries are depth 3 formula.

- A deterministic polynomial time learning algorithm with membership and equivalence queries for the class of $O(\log n)$ -term DNF. The hypotheses to the equivalence queries are polynomial size DNF.
- A polynomial time learning algorithm with membership and equivalence queries for the class of all functions of the form $g(T_1, \dots, T_k)$ where g is any monotone function, T_i are terms and $k = O(\log n)$.

The former result is proved in [BR92] and [Bs93] using (n, k) -universal sets. Our algorithm in this paper is simple and the use of universal sets is not required. The latter result is a generalization for the former.

- A polynomial time PAC-learning algorithm with membership queries under any quasipolynomial bounded uniform distribution (the distribution of each point is less than $2^{poly(\log n)}/2^n$) for the class of functions of the form $f(T_1, \dots, T_k)$ where f is any boolean function, T_i are terms or clauses and $k = O(\log n / \log \log n)$.
- A polynomial time PAC-Learning algorithm with membership queries for the class of decision trees of depth $O(\log n)$ over terms under the uniform distribution.

Decision trees of depth $O(\log n)$ over terms includes the classes $O(\log n)$ -term DNF and $O(\log n)$ -term CNF. The above is a generalization for the learnability of decision trees of constant depth over terms (see [Bs93]).

- A polynomial time PAC-learning algorithm under the uniform distribution for the class of functions of the form $g(M_1, \dots, M_k)$ where $k = O(\log n)$, T_i are clauses or terms and g is any monotone boolean function.

This result is a generalization of the result in [SM94] for learning monotone $O(\log n)$ -term DNF.

- A polynomial time learning algorithm with membership queries only for the class of functions of the form $f(M_1, \dots, M_k)$ where M_i are monotone terms, f is any boolean function and k is constant.

This result is a generalization of the result of the learnability of k -term monotone DNF [BGHM93] and the learnability of k -term multivariate polynomials [RB89] from membership queries only.

3 Preliminaries

A boolean function is a function $f : X \rightarrow B$ for some set X and $B = \{0, 1\}$. All classes considered in this paper are classes of boolean functions where $X = B^n$. We will use the variables x_1, \dots, x_n as arguments for f and for an assignment a , the i -th entry of a will be denoted by a_i or $a[i]$. For two assignments a and b we define the assignment $c = a \vee b$ (resp. $c = a \wedge b$) where $c_i = a_i \vee b_i$ (resp. $c_i = a_i \wedge b_i$). The assignment e_i satisfies $e_i[i] = 1$ and for every $j \neq i$, $e_i[j] = 0$. We will use $f|_{x_i \leftarrow \xi}$ to mean the functions obtained from f by substituting $x_i \leftarrow \xi$. We will use the order $<$ over B^n where for two assignments a and b , $a < b$ if for every $1 \leq i \leq n$, $a_i \leq b_i$ and $a \neq b$. This can be extended to functions. For two functions f and g we will write $f < g$ to mean that for all $a \in B^n$, $f(a) \leq g(a)$ and f and g are not equivalent. A boolean function f is monotone if $a > b$ implies $f(a) \geq f(b)$. A boolean function is called *unate* if there is an assignment a such that $f(x + a)$ is monotone. Here “+” is the bitwise XOR of vectors. We will say that a variable x_i is monotone (antimonotone) in f if $f|_{x_i \leftarrow 0} \leq f|_{x_i \leftarrow 1}$ (if $f|_{x_i \leftarrow 1} \leq f|_{x_i \leftarrow 0}$). We will say that a variable x_i is unate in f if x_i is monotone or antimonotone in f .

A *literal* is either a variable or a negated variable. A *term* is conjunction of literals and a *clause* is a disjunction of literals. For an assignment ξ and a variable x we will define x^ξ to be x if $\xi = 1$ and \bar{x} if $\xi = 0$. For a clause or a term u , we will denote the number of literals in u by $|u|$. A DNF (disjunction normal form) is a disjunction of terms. A CNF (conjunction normal form) is a conjunction of clauses. A k -term DNF (k -clause CNF) is a DNF (CNF) with k terms (k clauses). A k -DNF (k -CNF) is a DNF (CNF) where each term (clause) contains at most k literals. A *decision tree* is a binary tree whose nodes are labeled with variables

$\{x_1, \dots, x_n\}$ and whose leaves are labeled with constants from $\{0, 1\}$. Each decision tree T represents a boolean function f_T . To compute $f_T(a)$ we start from the root and if it is labeled with x_i we will proceed to the left child if $a_i = 1$ and to the right child if $a_i = 0$. The value of the function is the value of the leaf that will eventually be found using this procedure.

For a class of boolean functions C , a *decision tree over C* is a decision tree whose nodes are labeled with functions from C . When we reach a node in this case, the function in this node is computed and we will proceed to the left child if the function value is 1 and to the right child if the function value is 0.

4 Techniques and Results

4.1 Learning from membership and equivalence queries

What would happen if we were to run Angluin's algorithm for learning monotone boolean functions on a DNF that is not monotone? The algorithm will continue to run as long as the oracle returns only positive counterexamples. If we do get a negative counterexample b , then it must be above ($>$) some positive counterexample a . Now walking from b towards a by flipping the bits on which they disagree will give us an assignment that satisfies $f(c) = 0$ but for some $1 \leq i \leq n, c_i = 1, f(c + e_i) = 0$. This will give us the information that x_i is not monotone in f . Here we will stop the algorithm, assume that x_i is antimonotone in f and try to learn $f(x + e_i)$ using Angluin's algorithm. We can simulate membership and equivalence queries for $f(x + a)$ using membership and equivalence for $f(x)$ as described in [AHK89].

At some stage we will be trying to learn $f(x + a)$ using Angluin's algorithm which will give rise to two cases. Either we will find a new variable x_i with an $a_i = 0$ that is not monotone or we will find a variable x_i with an $a_i = 1$ that is not antimonotone. In the former we proceed as before, i.e., stop the algorithm, assume that x_i is antimonotone and run the algorithm again for $f(x + b)$ where $b = a + e_i$. For the latter we know that the variable x_i is not unate in f and we proceed by stopping the algorithm and start building a decision tree with x_i as the root. We will call this a *decision tree based on the non-unate variables*. In this decision tree the variables that label the nodes are not unate in the target but the leaves will be unate DNF formulas. We will define the *size* of the decision tree based on the non-unate variables to be the number of the leaves in the tree.

Now, if f is a DNF and the size of any decision tree based on non-unate variables for f is polynomial in n and the size of f then f is learnable from membership and equivalence queries by the above algorithm.

Notice that when the tree T is built we can still simulate membership queries for the functions on the leaves. Equivalence queries for the hypothesis on the leaves are simulated as follows. For each leaf we run the algorithm for learning the boolean function in that leaf. When all the algorithms want to ask equivalence queries

with hypothesis $\{h_i\}$ we build the hypothesis T with the leaves replaced by the corresponding h_i and ask one equivalence query. It is clear that a counterexample for this hypothesis will be a counterexample for exactly one of the hypotheses h_i .

In the above algorithm (and throughout the paper) the hypotheses to the equivalence queries are polynomial size decision trees with leaves that are polynomial size unate DNF. This class of hypotheses is a subset of the class of polynomial size DNF. This is because if V is the set of leaves in a decision tree with unate DNF leaves f , the assignments $x_1^v = \xi_1^v, \dots, x_{i_v}^v = \xi_{i_v}^v$ are the ones that lead to the leaf $v \in V$ in the tree and f_v is the unate DNF in the leaf $v \in V$ then

$$f = \bigvee_{v \in V} (x_1^v)^{\xi_1^v} \cdots (x_{i_v}^v)^{\xi_{i_v}^v} f_v$$

which is polynomial size DNF.

Now we will show that $O(\log n)$ -term DNF formulas are learnable from membership and equivalence queries using the above algorithm. To the best of our knowledge this is the first simple algorithm for learning $O(\log n)$ -term DNF noting that all previous learning algorithms for $O(\log n)$ -term DNF use $(n, O(\log n))$ -universal sets [BR92, Bs93]. We must first prove the following lemma.

Lemma 1. *Any decision tree based on the non-unate variables of a k -term DNF is of size at most 2^k .*

Proof. Notice that if x_i is a non-unate variable in f then we must have a term in f that contains x_i and another term that contains \bar{x}_i . Therefore if the root of the tree is x_i then both its children are $k - 1$ -term DNF's. This implies that the depth of the tree is at most k and therefore the size of the tree is at most 2^k . \square

The following is an immediate result of the preceding lemma.

Result 1 *The class of $O(\log n)$ -term DNF is learnable in polynomial time from membership and equivalence queries. The hypothesis class is polynomial size DNF.*

In the same way we prove the following more general result

Result 2 *The class of functions $g(T_1, \dots, T_k)$ where g is any monotone function T_i are terms and $k = O(\log n)$ is learnable in polynomial time from membership and equivalence queries. The hypothesis class is polynomial size DNF.*

Proof. As in lemma 1, a decision tree based on nonunate variables for $g(T_1, \dots, T_k)$ is of size at most 2^k . We need only to show that the unate DNF at the leaves are of polynomial size. Therefore we need to show that any function $g(T_1, \dots, T_k)$, where g is monotone and T_i are terms, has a polynomial size DNF representation. Write g as a monotone DNF $g = g_1 \vee \cdots \vee g_r$ where g_i are monotone terms in k variables. The number of possible monotone terms in k variables is at most 2^k so $r \leq 2^k$.

Now since g_i is a monotone term, $g_i(T_1, \dots, T_k)$ is a one term (possibly equivalent to zero). Therefore $g(T_1, \dots, T_k)$ contains at most 2^k terms which is polynomial for $k = O(\log n)$. \square

Next we will show that if the DNF is “almost” unate then it is learnable from membership and equivalence queries.

Lemma 2. *Any decision tree based on the non-unate variables for a unate DNF with $n-r$ variables with the addition of any s terms and the addition of r non-unate variables (to all the terms) is of size at most $2^r \binom{n}{s}$.*

Proof. We will call such a DNF an (n, s, r) -DNF. Let $\phi(n, s, r)$ be the maximal size of a decision tree based on non-unate variables of an (n, s, r) -DNF. We take a non-unate variable x_i and build a decision tree based on x_i . The DNF at the leaves are $(n-1, s-1, r)$ -DNF and $(n-1, s, r)$ -DNF if this variable is in the s additional terms or $(n-1, s, r-1)$ -DNF and $(n-1, s, r-1)$ -DNF if the variable is from the additional r nonunate variables. Therefore

$$\phi(n, s, r) \leq \max(\phi(n-1, s, r-1) + \phi(n-1, s, r-1) , \\ \phi(n-1, s, r) + \phi(n-1, s-1, r))$$

and $\phi(n, 0, 0) = 1$. This implies the result. \square

This gives the following.

Result 3 *The class of polynomial size unate DNF with the addition of $O(1)$ terms and $O(\log n)$ non-unate variables is learnable from membership and equivalence queries in polynomial time. The hypothesis class is polynomial size DNF.*

4.2 Learning from membership queries only

In this subsection we list a result using the divide and conquer technique when the learning is accomplished using membership queries only. In this learning model the learning algorithm provides the membership oracle with an assignment $a \in \{0, 1\}^n$ and the oracle returns the value of the target formula for this assignments, i.e., $f(a)$.

Let C be the class of boolean functions. Suppose for any $f \in C$ there is an algorithm that uses only membership queries for f to verify whether the target is either constant 0 or 1 and if it is not constant then the algorithm finds two assignments a and b such that $f(a) \neq f(b)$. One way to change this *constant verification* algorithm into a learning algorithm is as follows. We verify the function and find two assignments a and b such that $f(a) = 0$ and $f(b) = 1$. We then flip the bits in b that disagree with a one by one until changing one bit changes the value of the function from 1 to 0. We then know that the target function depends on the variable of this bit and we proceed as we did in the last section. We build a

decision tree based on this variable and recursively learn the two functions on the leaves. We call this type of tree a *tree based on relevant variables*. The size of this tree is the number of leaves in the tree and if the maximum possible size of such a tree is polynomial then this class is learnable in polynomial time.

Our next lemma gives us an interesting class with small decision trees that are based on the number of relevant variables.

Lemma 3. *Let $f : B^k \rightarrow B$ be a boolean function and t_1, \dots, t_k be terms. Any decision tree based on relevant variables for $f(t_1, \dots, t_m)$ has size at most*

$$\prod_{i=1}^m (|t_i| + 1).$$

Proof. Let $S(s_1, \dots, s_m)$ be the maximal possible size of a decision tree based on the relevant variables of $F = f(t_1, \dots, t_m)$ where $s_i = |t_i|$. If a relevant literal $l_i \in \{x_i, \bar{x}_i\}$ is chosen in F , say in t_1 , then substituting $l_i = 0$ will change t_1 to 0 and then the size of a decision tree based on relevant variables for $F|_{l_i \leftarrow 0}$ is at most $S(0, s_2, \dots, s_n)$. Substituting $l_i = 1$ will remove the literal l_i from t_1 and then the size of a decision tree based on relevant variables for $F|_{x_i \leftarrow 1}$ is at most $S(s_1 - 1, s_2, \dots, s_n)$. Therefore,

$$S(s_1, \dots, s_m) \leq S(0, s_2, \dots, s_n) + S(s_1 - 1, s_2, \dots, s_n)$$

and $S(0, \dots, 0) = 1$. Solving this recursion will give the result. \square

The following result is a generalization of the learnability of k -term monotone DNF [BGHM93] and the learnability of k -term multivariate polynomials from membership queries only [RB89]. It is also known from [BGHM93, RB89] that this result cannot be generalized to a non-constant k .

Result 4 *The set of all functions $f(m_1, \dots, m_k)$ are learnable in $O(n^k)$ time and membership queries where f is any function, k is a constant and m_1, \dots, m_k are monotone terms.*

By lemma 3 and the above algorithm, it is enough to show that this set is constant verifiable.

Lemma 4. *Let $g = f(m_1, \dots, m_k)$ where m_1, \dots, m_k are monotone terms. For any $\xi \in \{0, 1\}$, we have $g \equiv \xi$ if and only if $g(a) = \xi$ for all vectors of Hamming weight greater than or equal to $n - k$.*

Proof. It is enough to show that for every assignment a there is an assignment b of Hamming weight at least $n - k$ that satisfies $g(b) = g(a)$. Let (w.l.o.g)

$m_1(a) = 0, \dots, m_r(a) = 0$ and $m_{r+1}(a) = 1, \dots, m_k(a) = 1$. Since $m_j(a) = 0$ for $j \leq r$ we must have a variable x_{i_j} in m_j where $a_{i_j} = 0$. Define b to be the vector that is 0 in entries $i_j, j \leq r$ and 1 in the other entries. Then $m_j(b) = 0$ for $j \leq r$ and since $b > a$ we also have $m_{r+1}(b) = 1, \dots, m_k(b) = 1$. Therefore $g(b) = g(a)$ and the Hamming weight of b is at least $n - k$. \square

4.3 Learning from equivalence queries only

In this subsection we list some results for learning from equivalence queries only. In this learning model the learning algorithm provides the equivalence oracle with a hypothesis h that is a polynomial size circuit and the oracle answers “YES” if h is equivalent to the target formula f and answers “NO”, with a counterexample a such that $h(a) \neq f(a)$, otherwise.

We start with the following simple observation

Observation 5 *If a class of formulas C is learnable from membership and equivalence queries from hypothesis class H and C is learnable from equivalence queries then C is learnable from equivalence queries with an output from H .*

Proof. Learn f with any output hypothesis using equivalence queries only then run the algorithm that learns from membership and equivalence queries and answer membership queries using the hypothesis that was just learned. \square

Observation 6 *For a constant k there is an equivalence query algorithm that learns k -term DNF and outputs a k -term DNF.*

The next result is proven in [KLPV87] for the PAC learning model (in fact all the results in the table in [KLPV87] are also true for the equivalence query model).

Observation 7 *Let C_1 and C_2 be classes of concepts that are learnable in polynomial time from only equivalence queries where the hypothesis to an equivalence query is always a subset of the target concept. Then*

$$C_1 \vee \bar{C}_2 = \{f_1 \vee \bar{f}_2 \mid f_1 \in C_1 \text{ and } f_2 \in C_2\}$$

is learnable in polynomial time from equivalence queries only.

Proof. Run the algorithms that learn C_1 and C_2 with hypothesis $h_1 \vee h_2$ where h_1 and h_2 are the hypotheses of C_1 and C_2 , respectively. If a positive counterexample is returned then it is for h_1 and if a negative counterexample is returned then it is for h_2 . \square

The following result follows from Observation 7.

Result 8 *The class k -DNF \vee k -term DNF is polynomial time learnable from equivalence queries only.*

4.4 PAC Learning with membership queries

In this subsection we list some results for PAC-learning with membership using the divide and conquer technique. During PAC learning the learner only sees examples $(a, f(a))$ according to some distribution D and the learning algorithm should run in time polynomial in n , $1/\epsilon$ and $1/\delta$ and output a polynomial sized circuit hypothesis h such that $\text{Prob}(D(f\Delta h) < \epsilon) > 1 - \delta$. It is known [A88] that learning from equivalence and membership queries implies PAC-learning with membership queries under any distribution.

We say that a distribution D supports $r(n)$ -terms if $r(n)$ is the minimal integer such that for any term T of size $\omega(r(n))$ the distribution $D(T)$ is less than any $1/\text{poly}(n)$. The uniform distribution over $\{0, 1\}^n$ supports $O(\log n)$ -terms.

The first result is

Result 9 *The class of functions of the form $f(T_1, \dots, T_k)$ where f is any boolean function, T_i are clauses or terms and $k = O(\log n / \log \log n)$ is PAC-learnable in polynomial time with membership queries under any distribution that supports $\log^c n$ -terms for any constant c .*

Proof. The algorithm builds a decision tree based on relevant variables for f . If at some stage we have l leaves we draw examples until we have enough examples for each leaf. If for each leaf v , with probability greater than $1 - \delta/l$, the function in v is constant with error $\epsilon/2$ (using Chernoff bounds) then we replace the leaf by that constant. If a leaf cannot be reached with probability less than $\epsilon/(2l)$ then we replace it with any constant. This implies that with probability at least $1 - \delta$ the tree is an ϵ approximation of the target. If some leaf is not an ϵ approximation of a constant then we choose two points a and b that have different values for that leaf and use them to find a new relevant variable for that leaf. To show that the above algorithm will work we need to show that the size of the tree will never grow more than $\text{poly}(n)$.

Let g be a function that is induced from f by substituting 0 for all terms $T_i, i \in I$ with $|T_i| > \log^{2c+1} n$ and 1 for all clauses $T_j, j \in J$ with $|T_j| > \log^{2c+1} n$. Obviously, g is a good approximate for f (for any $\epsilon = 1/\text{poly}(n)$) because the distribution supports $\log^c n$ -terms and hence for any random example a with probability at least $1 - 1/\omega(\text{poly}(n))$ we have

$$T_i = 0 \text{ for } i \in I \text{ and } T_j = 1 \text{ for } j \in J. \quad (1)$$

We will show that with probability at least $1 - 1/\omega(\text{poly}(n))$ all the membership queries asked on a in the above algorithm will satisfy $f(a) = g(a)$. It is clear that with probability $1 - 1/\omega(\text{poly}(n))$ all of the random examples in the algorithm will satisfy (1). This does not necessarily imply that every membership queries satisfies (1). We need a stronger condition so we claim that, with probability $1 - \omega(\text{poly}(n))$, for every two random examples a and b , $a \vee b$ and $a \wedge b$ satisfy (1). This implies that all points between $a \vee b$ and $a \wedge b$ in the $\{0, 1\}^n$ lattice satisfy (1). Now since

membership queries are asked for points in that range we conclude that all of the assignments in the membership queries satisfy (1).

To prove the above, we show that for $i \in I$

$$\Pr_{a,b}[T_i(a \vee b) = 1] < \frac{1}{\omega(\text{poly}(n))}$$

and the same bound holds for $i \in J$ and $a \wedge b$. This is shown in the appendix. Since the number of examples and $|I \cup J|$ is polynomial, we get the result.

Therefore our algorithm will not distinguish between membership queries to f and membership queries to g . Now since, by lemma 3, the size of decision tree based on relevant variables for g is at most

$$(\log^{2c+1} n + 1)^{O\left(\frac{\log n}{\log \log n}\right)} = \text{poly}(n),$$

so the size of the tree in the algorithm will not grow more than polynomial size. \square

When the distribution is uniform we can prove a stronger result. Using the divide and conquer approach we will prove that decision trees of depth $O(\log n)$ over terms (the nodes of the decision tree contains terms) can be ϵ -approximated by a decision tree of depth $O(\log n + \log(1/\epsilon))$. This, with the results in [KM93], immediately implies the following.

Observation 10 *The class of decision trees of depth $O(\log n)$ over terms is PAC-learnable with membership queries under the uniform distribution.*

Notice that the class decision trees of depth $O(\log n)$ over terms contains the class of $O(\log n)$ -term DNF, $O(\log n)$ -term CNF and the class in result 9.

Lemma 5. Any decision tree of depth $O(\log n)$ over terms can be ϵ -approximated by a decision tree of depth $O(\log n + \log(1/\epsilon))$.

Proof. Given a decision tree of depth d over terms T , we construct a decision tree T' (over variables) for T as follows. Choose a variable x_i from the term of the root of T and put it in the root of T' . For one of the values $\xi \in \{0, 1\}$ of x_i the term at the root is 0 and therefore the projection $T|_{x_i \leftarrow \xi}$ is a decision tree of depth $d - 1$ over terms. We then recursively build the tree for $T|_{x_i \leftarrow 0}$ and $T|_{x_i \leftarrow 1}$ in the children of x_i in T' . After building T' we truncate T' at depth $r = O(\log n + \log(1/\epsilon))$, i.e., delete all nodes of depth greater than equal to $r + 2$ and replace all nonleaves at depth $r + 1$ by leaves labeled with 0.

Now randomly uniformly choose x from $\{0, 1\}^n$. Computing T' at a random uniform x is equivalent to randomly walking in T' . At each random step in T' , if we observe the computation in T we notice that with probability $1/2$ the computation in T is moving to the children of the current node. This is because substituting a random value for a variable in a term will cause the term to be 0 with probability $1/2$. Therefore to have a $1 - \epsilon$ probability that the computation in T reaches the leaves, we need (by Chernoff bound) T' to be of depth $r = O(\log n + \log(1/\epsilon))$. \square

4.5 Statistical query learning

In this subsection we list a result for statistical query learning using the divide and conquer technique. In the statistical query model the learning algorithm can ask *statistical query* (g, η) where g is a polynomial time computable boolean function over the domain $\{0, 1\}^n$ and $\eta = \frac{1}{\text{poly}(n, 1/\epsilon)}$. The statistical oracle returns an η -approximation of the expectation of $g(x, f(x))$ under a distribution $D, E_D[g_i(x, f(x))]$. The learning algorithm should ask a polynomial number of statistical queries, run in polynomial time and outputs a hypothesis h that is an ϵ -approximation of the target function under the distribution D .

It is known from [K93] that polynomial time learning in this model implies error tolerant polynomial time PAC-learning.

The following result is a generalization of the result of [SM94] for PAC-learning monotone $O(\log n)$ -term DNF with respect to the uniform distribution. This result cannot be extended to non-monotone functions [BFJ+94].

Result 11 *The class of functions $M(g_1, \dots, g_k)$ for monotone functions M and monotone terms or clauses g_1, \dots, g_k with $k = O(\log n)$ is SQ-learnable under the uniform distribution in polynomial time.*

Proof. The function $f = M(g_1, \dots, g_k)$ can be represented as a decision tree of depth k over the terms g_1, \dots, g_k . Any one of the terms can be the term in the root of this decision tree. Therefore by lemma 5 any decision tree based on relevant variables for f of depth $O(k + \log(1/\epsilon))$ is an ϵ -approximation for f . Notice that when $k = O(\log n)$ the size of this tree is polynomial.

Now if we can find a relevant variable using some expectation of the form $E[g(x, f(x))]$ then we can build a decision tree for f . To find a relevant variable we use the result from [KV94] which shows that for any monotone boolean function f , if f is not an ϵ -approximation of a constant then there exists a variable that correlates well with f under the uniform distribution, i.e., there is an x_i such that

$$E[(1 - 2x_i)(1 - 2f)] \geq \frac{1}{\text{poly}(n, 1/\epsilon)}.$$

Notice that $E[(1 - 2x_i)(1 - 2f)] = 4E[x_i f] - 2E[f]$ and so it can be found using two statistical queries. Obviously, if a variable correlates well with the target then this variable is relevant. We use this fact to build the decision tree based on the relevant variables for f . We do this recursively until the depth of the leaf exceeds $O(\log n + \log(1/\epsilon))$ or until the leaf is an ϵ -approximation of some constant.

Notice that since the depth of the tree is at most $O(\log n + \log(1/\epsilon))$, it is possible to simulate statistical queries for the functions on the leaves by statistical queries of f as follows. If $x_{i_1} = \xi_{i_1}, \dots, x_{i_l} = \xi_{i_l}$ are the values that leads to some leaf v in the tree where $l < O(\log n + \log(1/\epsilon))$ and $\xi_j \in \{0, 1\}$, and f_v is the function in the leaf v then

$$E[g(x, h(x))] = \Pr[g(x, f(x)) = 1 | x_{i_1} = \xi_{i_1}, \dots, x_{i_l} = \xi_{i_l}]$$

$$= \frac{\Pr[g(x, f(x))x_{i_1}^{\xi_{i_1}}, \dots, x_{i_l}^{\xi_{i_l}}]}{2^l}$$

where $x^0 = \bar{x}$ and $x^1 = x$. Now since $2^l < \text{poly}(n/\epsilon)$ it is possible to get an answer for $SQ(g(x, h(x)), \eta)$ by asking the statistical query $SQ(g(x, f(x))x_{i_1}^{\xi_{i_1}}, \dots, x_{i_l}^{\xi_{i_l}}, \eta/2^l)$ and then multiplying by 2^l . \square

acknowledgment. I would like to thank David Wilson for proof reading the paper.

References

- [A88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.
- [AHK89] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. Technical report, Report No. UCB/CSD 89/528, Computer Science Division, University of California Berkeley, 1989. To appear, *J. ACM*.
- [Bs93] N. H. Bshouty. Exact learning via the monotone theory. In *Proceeding of the 34th Symposium on Foundations of Computer Science*. pages 302–311, November 1993.
- [BR92] A. Blum and S. Rudich. Fast learning of k -term DNF formulas with queries. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 382–389, May 1992.
- [BS90] A. Blum and M. Singh. Learning functions of k terms. In *Proceeding of the Third Annual Workshop on Computational Learning Theory*, pages 144–153, August 1990.
- [BFJ+94] A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, S. Rudich. Weakly learning DNF and characterizing statistical queries learning using fourier analysis. In *Proceedings of the Twenty Sixth Annual ACM Symposium on Theory of Computing*, pages 253–262, May 1994.
- [K93] M. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proceeding of the 25th Annual ACM Symposium on Theory of Computing*, pages 392–401, May 1993.
- [KM93] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM J. Computing*, 22(6):1331–1348, 1993.
- [KV94] M. Kearns and L. Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata, *Journal of the ACM*, 41, 1994.

- [SM94] Y. Sakai and A. Maruoka. Learning monotone log-term DNF formula. In *Proceeding of the Seventh Annual ACM Workshop on Computational Learning Theory*, pages 165–172, July 1994.
- [Val84] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

Appendix

Lemma A1. Let D be a distribution on $\{0, 1\}^n$ that supports $\log^c n$ terms. Let T be a term of size $|T| \geq \log^{2c+1} n$. For two random assignments a and b we have

$$\Pr_{a,b}[T(a \vee b) = 0] \leq \frac{1}{\omega(\text{poly}(n))}.$$

Proof. We may assume without loss of generality that $T = x_1 \wedge \cdots \wedge x_r$ where $r = \log^{2c+1} n$. For an assignment $a \in \{0, 1\}^r$ we define T^a to be the term

$$\bigvee_{a_i=0} x_i.$$

The Hamming weight of vector a will be denoted by $wt(a)$ and we will write $s(T)$ for $|T|$. We have

$$\begin{aligned} \Pr_{a,b}[T(a \vee b) = 1] &= E_{a,b}[T(a \vee b)] \\ &= E_a E_b[T^a(b)] \\ &= E_a[E[T^a] | s(T^a) \leq \log^{c+1} n] \Pr_a[s(T^a) \leq \log^{c+1} n] + \\ &\quad E_a[E[T^a] | s(T^a) > \log^{c+1} n] \Pr_a[s(T^a) > \log^{c+1} n]. \end{aligned}$$

If $s(T^a) \geq \log^{c+1} n = \omega(\log^c n)$ then since the distribution supports $\log^c n$ -terms we have

$$E[T^a] = \Pr_b[T^a(b) = 1] = \frac{1}{\omega(\text{poly}(n))}$$

and therefore

$$E_a[E[T^a] | s(T^a) > \log^{c+1} n] = \frac{1}{\omega(\text{poly}(n))}.$$

Now if we show that $\Pr_a[s(T^a) = \log^{c+1} n] < 1/\omega(\text{poly}(n))$ then from the above we get $\Pr_{a,b}[T(a \vee b) = 1] = 1/\omega(\text{poly}(n))$ and the result of the lemma follows. Notice that $s(T^a) \leq \log^{c+1} n$ is equivalent to $wt(a) \geq \log^{2c+1} n - \log^{c+1} n$. The next lemma shows that there is a polynomial size DNF, $g = T_1 \vee \cdots \vee T_s$, $s = \text{poly}(n)$ with $|T_i| = \log^{c+1} n$, such that $g(a) = 1$ for all a with $wt(a) \geq \log^{2c+1} n - \log^{c+1} n$. Therefore

$$\begin{aligned} \Pr_a[s(T^a) \leq \log^{c+1} n] &= \Pr_a[wt(a) \geq \log^{2c+1} n - \log^{c+1} n] \\ &\leq \Pr[g = 1] \\ &\leq \sum_i \Pr[T_i = 1] \\ &\leq \frac{s}{\omega(\text{poly}(n))} = \frac{1}{\omega(\text{poly}(n))}. \square \end{aligned}$$

Lemma A2. Let $A \subset \{0, 1\}^{\log^{2c+1} n}$ be the set of all assignments of Hamming weight at least $\log^{2c+1} n - \log^{c+1} n$. There is a polynomial size DNF with terms of size $\log^{c+1} n$ that is 1 for all the points in A .

Proof. We give a probabilistic proof. Randomly uniformly choose m terms T_1, \dots, T_m of size $\log^{c+1} n$ and define $g = T_1 \vee \dots \vee T_m$. We have

$$\begin{aligned}
\Pr[(\exists a \in A) g(a) = 0] &\leq \binom{\log^{2c} n}{\log^{c+1} n} \Pr[g(a) = 0] \\
&\leq 2^{\log^{c+2} n} (\Pr[T_i(a) = 0])^m \\
&= 2^{\log^{c+2} n} \left(1 - \left(1 - \frac{1}{\log^c n} \right)^{\log^{c+1} n} \right)^m \\
&\approx 2^{\log^{c+2} n} \left(1 - \frac{1}{n^{\log e}} \right)^m
\end{aligned}$$

and for $m = \text{poly}(n)$ we can get the probability of $[(\exists a \in A) g(a) = 0]$ to be less than 1. \square