

Online Learning

Nader H. Bshouty

Eyal Kushilevitz

Abstract

We give Online learning model.

1 Learning from Membership Queries Only – Positive Results

So far we showed that learning with equivalence queries only (Section ??) and learning with membership queries only (Section ??) are not very powerful. Instead, we considered the combination of equivalence queries and membership queries (Sections ?? – ??) and we showed that in this case there are very powerful learning algorithms. In this section we re-consider again learning from membership queries only. We will show two positive results. The first is that the class of all functions which are arbitrary “combinations” of k monotone terms is learnable in time polynomial in n^k . This class includes for example monotone DNF with k terms. The second result is that the class of decision tree of depth k is learnable in time polynomial in 2^k .

1.1 Learning Combinations of Monotone Terms

Let $F_k(\text{MTERMS})$ be the class of all functions of the form $f = g(M_1, \dots, M_k)$ where M_1, \dots, M_k are monotone terms and g is an arbitrary boolean function. To learn this class, we use the divide and conquer technique. The idea is to find a variable x_j that the target function f depends on. We then write f as

$$f = x_j f_1 + \bar{x}_j f_0$$

as in Section ?? . Observe that in this case both f_0 and f_1 are in $F_{k-1}(\text{MTERMS})$ and so we can recursively learn the two functions f_0 and f_1 . Notice that if $\phi(n, k)$ is the maximal number of “divide” steps of $f \in F_k(\text{MTERMS})$ then since $f_0 \in F_{k-1}(\text{MTERMS})$ we have $\phi(n, k) \leq \phi(n-1, k-1) + \phi(n-1, k)$. It follows that $\phi(n, k) \leq \binom{n}{k} = O(n^k)$. Therefore, the problem is reduced to the problem of finding a variable that f depends on. We use the following fact:

Fact 1.1 *A function $f \in F_k(\text{MTERMS})$ is not constant (0 or 1) if and only if f is not constant over the domain*

$$W_k = \{a \in \{0, 1\}^n \mid wt(a) \geq n - k\}$$

where $wt(a)$ is the number of ones in a .

Proof: Obviously if f is not constant over W_k then it is not a constant function. For the other direction, suppose $f = g(M_1, \dots, M_k)$ where M_1, \dots, M_k are monotone terms. Since f is not a constant function then there exist two assignments $c^{(0)}$ and $c^{(1)}$ such that $f(c^{(0)}) = 0$ and $f(c^{(1)}) = 1$. Now, for each $\xi \in \{0, 1\}$ we do the following: let $c = c^{(\xi)}$. Denote by $I \subseteq \{1, \dots, k\}$ the set of all terms such that $M_i(c) = 0$. For every $i \in I$, since $M_i(c) = 0$ then there must be a variable $x_{j(i)}$ in M_i such that $c_{j(i)} = 0$. Now, define c' to be the assignment that is 1 in all entries except the entries $j(i)$ where $i \in I$. First, notice that $c' \geq c$ and that $wt(c') \geq n - k$. Now, if $i \in I$ then since $c'_{j(i)} = 0$ and $x_{j(i)}$ is a variable in M_i we have $M_i(c') = 0$. If $i \notin I$ then since M_i is monotone, $c' \geq c$ and $M_i(c) = 1$ then $M_i(c') = 1$. Now, since $M_i(c') = M_i(c)$ for all i we have $f(c) = f(c') = \xi$. To conclude, we got two assignments in W_k on which f have different value. \square

This lemma shows that by asking membership queries on $|W_k| \leq n^k$ assignments we can decide whether a function in $f \in F_k(\text{MTERMS})$ is constant. Also, given two assignments a and b such that $g(a) = 0$ and $g(b) = 1$ we can find a variable x_j that f depends on as follows. Flip bits in a that disagree with b and observe (using membership queries) the value of the function on the modified assignment until flipping a bit, say c_j , will change the function value from 0 to 1. Then, we know that the function depend on x_j .

Exercise 1.1.1 Write in details an algorithm for learning the class $F_k(\text{MTERMS})$ and analyze its complexity.

1.2 Learning Decision Trees of Small Depth

In this section we consider the class $\text{XOR}(k\text{-TERMS})$ of all functions that can be expressed as the exclusive-or of terms of size at most k . We show that this class is learnable from membership queries in time polynomial in n and 2^k . It can be shown that decision trees of depth k are in this class and therefore decision trees of depth $k = O(\log n)$ are efficiently learnable from membership queries only. First, we observe that we may assume that all the terms are monotone. This is because any term of size k can be transformed into an exclusive-or of (at most 2^k) monotone terms of size k . For example:

$$x_1\bar{x}_2x_3\bar{x}_4 = x_1(x_2 \oplus 1)x_3(x_4 \oplus 1) = x_1x_2x_3x_4 \oplus x_1x_3x_4 \oplus x_1x_2x_3 \oplus x_1x_3.$$

Our algorithm works by assigning values to some of the variables and considering the resulted function. We will use the following fact:

Fact 1.2 Let $f = M_1 \oplus M_2 \oplus \dots \oplus M_t$ where M_1, \dots, M_t are distinct monotone terms. Assume that M_1 is a term with a maximal size in f . Then, assigning values to the variables not in M_1 and keeping the variables in M_1 as they are yields a nonzero function of the variables in M_1 .

A basic step in the algorithm is a test denoted **test-zero**, that checks for a function $f \in \text{XOR}(k\text{-TERMS})$, whether it is identically 0. This is done by picking a “small” number of assignments and checking whether f is 0 on all of them or not. The following lemma is used for analyzing this test.

Lemma 1.2.1 *Let $f \in \text{XOR}(k\text{-TERMS})$. If $f \not\equiv 0$ then*

$$\Pr_a[f(a) = 1] \geq \frac{1}{2^k},$$

where the probability is over a which is chosen uniformly at random in $\{0, 1\}^n$.

Proof: If $f \not\equiv 0$ then it has at least one term. Let M be a term in f of maximal size. Consider an arbitrary assignment of values for the variables not in M . By Fact ??, any such partial assignment gives a nonzero function on the (at most k) variables of M . This means, that for at least one assignment (out of at most 2^k possibilities) for the variables in M we get an assignment (for all n variables) on which f is 1. Since this holds for every partial assignment it follows that at least $1/2^k$ of the 2^n assignments are non-zero. Therefore, with probability at least 2^{-k} , a random assignment a satisfies $f(a) \neq 0$. \square

Algorithm test-zero(δ):
 $t \leftarrow 2^k \log(1/\delta)$
 Pick uniformly and independently t random assignments a_1, \dots, a_t
 For $i = 1, \dots, t$ If $f(a_i) = 1$ return(a_i);
 Return(TRUE)

Figure 1: Algorithm test-zero(δ)

The procedure `test-zero` is described in Figure ??. It gets as input a parameter δ and it either answers TRUE (to indicate that $f \equiv 0$) or it gives back an assignment a_i for which $f(a_i) = 1$. By Lemma ??, for every function f , the test is correct with probability at least $1 - \delta$. We remark that since our algorithm uses `test-zero` as a procedure it becomes a randomized algorithm; that is, it is subject to an error with some small probability. However, if we use it with δ sufficiently small we guarantee that with high probability in *all* executions of `test-zero` no mistake is made.

Now, the algorithm tries to learn a target function $f \in \text{XOR}(k\text{-TERMS})$ by recognizing its terms one by one. It does so by asking only membership queries. Note that for such a function f if f contains at least one term then it is not identically zero. We start by executing `test-zero`. If the function f is identically zero then the test returns TRUE and we stop. Otherwise, with probability at least $1 - \delta$, we have found an assignment $a^{(1)}$ such that $f(a^{(1)}) = 1$. We define the projected function f_1 by fixing to 0 all the variables x_i for which $a_i^{(1)} = 1$ and leaving all other variables “alive” (note that since the assignments are chosen at random we expect “many” of the variables to be set to 0). Notice that f_1 is not identically zero because when we substitute 1 in all the variables of f_1 we get exactly the value of $f(a^{(1)})$ which is 1. Also notice that the set of terms in f_1 is a subset of the set of terms of f .

We proceed by applying `test-zero` to f_1 and getting a projected function f_2 on even smaller number of variables and so on. Continuing this way we get a sequence of functions

f_1, f_2, \dots until the number of variables is not reduced anymore; i.e., $f_{s+1} = f_s$. In this case we define a term M_1 which is the conjunction of all variables on which f_s depends.

Fact 1.3 *If $f_s \neq M_1$ then with probability at least $1/2$ we get $f_{s+1} \neq f_s$.*

After we find a term M_1 of f , we try to learn $f^{(1)} = f \oplus M_1$. Note that if indeed M_1 is a term of f then the function $f^{(1)}$ is the function f without the term M_1 . To simulate a membership query $\text{MQ}(a)$ for $f^{(1)}$, we ask $\text{MQ}(a)$ for f and add the value $M_1(a)$ to the answer. We repeat the above procedure to find the other terms.

Exercise 1.2.1 *Write the algorithm in details and analyze it.*

Exercise 1.2.2 *Show that the class of depth k decision trees is a subset of the class $\text{XOR}(k\text{-TERMS})$.*

Exercise 1.2.3 *Show that if $f = M_1 \oplus M_2 \oplus \dots \oplus M_t$ and M_1, \dots, M_t are distinct monotone terms then $f \not\equiv 0$.*

Exercise 1.2.4 *Let $f = M_1 \oplus M_2 \oplus \dots \oplus M_t$ and M_1, \dots, M_t be monotone terms. Show that if $f(a) = 1$ and for every $b < a$ we have $f(b) = 0$ then for some i*

$$M_i = \bigwedge_{j : a_j=1} x_j.$$

Exercise 1.2.5 *Prove Facts ?? and ??.*