A Subexponential Exact Learning Algorithm for DNF Using Equivalence Queries

Nader H. Bshouty[*]

Department of Computer Science

The University of Calgary

Calgary, Alberta, Canada T2N 1N4

`e-mail: bshouty@cpsc.ucalgary.ca`

`http://www.cpsc.ucalgary.ca/~bshouty/home.html`

Abstract

We present a $2^{\tilde{O}(\sqrt{n})}$ time exact learning algorithm for polynomial size DNF using equivalence queries only. In particular, DNF is PAC-learnable in subexponential time under any distribution. This is the first subexponential time PAC-learning algorithm for DNF under any distribution.

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

1 Introduction

One of the outstanding open problems in computational learning theory is whether polynomial size DNF formulas are learnable in polynomial time. In [BCKT94] Bshouty et al. showed that by using an NP-oracle, DNF is exactly learnable from equivalence queries. So, if P=NP, exact learning of DNF from equivalence queries is possible. This raises the question of whether learning DNF in polynomial time implies P=NP, or whether learning DNF in time t implies NP=TIME(t). So researchers started to seek a subexponential time exact learning algorithm for DNF to show that the learnability of DNF is most likely not as hard as solving NP-Complete problems (unless NP$\subseteq$TIME($2^{n^{\epsilon}}$)).

This paper gives the first subexponential time exact learning algorithm for DNF. We show that any circuit that has a subexponential DNF or CNF size also has a subexponential learning algorithm from equivalence queries. For polynomial size DNF our algorithm runs in time $2^{\tilde{O}(\sqrt{n})}$.

Our algorithm is based on learning k-decision list from equivalence queries [HSW90]. We show that every n-decision list of length s is an $O(\sqrt{n \log n \log s})$-decision list. Therefore running the $O(\sqrt{n}\log n)$-decision list learning algorithm will learn any polynomial size DNF.

In Section 2 we give the learning model and some definitions. In Section 3 we show that any n-decision list of length s is an $O(\sqrt{n \log s \log n})$-decision list.

2 Preliminaries

In this section we define the learning model and classes of DNF, decision trees and decision lists. Then we give some results from the literature that will be used for our proof.

2.1 The Learning Model

In the exact learning model, using *equivalence queries* only, the learning algorithm tries to learn a function f, from a class of boolean functions C, that the teacher (or the equivalence oracle) has. The learning algorithm can ask the teacher equivalence queries. In an equivalence query the learning algorithm sends the teacher a hypothesis h and the teacher sends back the answer YES if $h \equiv f$ and NO with a counterexample a if $h \not\equiv f$. The counterexample a satisfies $f(a) \neq h(a)$.

We say that C is *exactly learnable in time t* from equivalence queries if the learning algorithm can identify the target exactly in computational time complexity

t with t equivalence queries and with a hypotheses h that can be represented in t bits that are computable in computational time t. Notice that if we allow the hypothesis to be computable in exponential time then DNF is learnable in polynomial time. All we would need is the halving algorithm [A88,L88] where each step defines the hypothesis to be the majority of all functions in C that are consistent with the answers of the equivalence queries seen so far.

2.2 Classes of Boolean Functions

Our boolean functions will be defined over the *variables* $X = \{x_1, x_2, \ldots\}$. A *literal* is x_i or $\neg x_i$, a *term* is a conjunction of literals and a *Disjunction Normal Form* formula (DNF) is a disjunction of terms. Swapping disjunctions with conjunctions in a DNF we will result in a *Conjunction Normal Form* (CNF) that is a conjunction of *clauses*. It is known that any boolean function can be represented as DNF and CNF. For a term T the *size of* T, $|T|$, is the number of literals in T. The size of a DNF is the number of terms it contains and the *DNF size of a boolean function* f is the minimum size of all equivalent DNF's that represent f. A k-DNF is a DNF with terms of size at most k.

A k-decision list is a list $f = ((T_1, b_1), (T_2, b_2), \cdots, (T_l, b_l))$ where each T_i is a term of size at most k, $b_i \in \{0, 1\}$ and the disjunction of all T_i is 1. The list f is a representation of a boolean function. For any assignment a at least one of the $T_i(a)$ is 1 and if $T_1(a) = \cdots = T_{j-1}(a) = 0$ and $T_j(a) = 1$ then $f(a) = b_j$. Any k-DNF (or k-CNF) is a k-decision list. Notice that since the number of terms of size k is at most $(2n)^k$ the length of any k-decision list is at most $(2n)^k$.

A *decision tree* is a rooted binary tree with internal nodes labeled with variables and with leaves labeled from $\{0, 1\}$. Every decision tree t represents a boolean function. For an internal node v in a decision tree t let $r(v)$ (resp. $l(v)$) be the right (resp. left) subtree of v. To compute $t(a)$ we start from the root and if the root v is labeled with x_i then $t(a) = l(v)(a)$ if $a_i = 1$ and $f(a) = r(f)(a)$ if $a_i = 0$. If the root is a leaf then $t(a)$ is equal to the label in this leaf. The size of a decision tree is the number of leaves in the tree. A *k-decision list* is a decision list with internal nodes labeled with terms of size k.

A $(k, s, j, l)-$decision tree is a k-decision tree of size s with leaves that are j-decision lists of length l. To compute $t(a)$, where t is a (k, s, j, l)-decision tree, we start from the root and do the following at each node. If the node is a leaf then we compute the decision list in that leaf and if it is an internal node v we proceed to the left tree if the value of the term in v is 1 and to the right tree if it is 0.

2.3 Previous Results

Helmbold et al. in [HSW90] showed that k-decision lists are learnable from equivalence queries in time

$$n^{O(k)}.$$

In [Bl92] Blum proved that any decision tree of size s is a $\log s$-decision list. Therefore, there is a learning algorithm for decision trees of size s that runs in time

$$n^{O(\log s)}.$$

3 Decision List for DNF

In this section we prove the following lemma

Lemma 1: An n-decision list of length s is an

$$O\left(\sqrt{n \log s \log n}\right) - \text{decision list.}$$

Since a DNF of size s has an n-decision list of length s it follows that any DNF of size s is an $O(\sqrt{n \log s \log n})$ – decision list.

This implies the following theorem.

Theorem 1: There is a learning algorithm for DNF from equivalence queries that runs in time

$$2^{O\left(\sqrt{n \log s} \log^{1\frac{1}{2}} n\right)}.$$

Notice that this complexity is subexponential for any subexponential size DNF (or CNF).

Proof of Lemma 1: We will first show that an n-decision list of length s is a $(1, 2^h, h, 2^h)$-decision tree for $h = O(\sqrt{n \log s \log n})$ and then show that a $(1, 2^h, h, 2^h)$-decision tree is a $2h$-decision list.

Let $f = ((T_1, b_1), \ldots, (T_s, b_s))$ be an n-decision list of length s. Let $h = O(\sqrt{n \log s \log n})$. Let $T_{i_1}, \ldots, T_{i_r}$ be all the terms that have size greater than or equal to h. Since the size of each T_{i_j} is more than h, there is a variable x_i that occurs in more than $(rh)/n$ terms. We put this variable in the root of the tree. The children of the root will be trees for the projections $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$. We do this recursively for the projections $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$ until no more terms of size greater than h occur in the list. This way we are building a $(1, \eta, h, 2^h)$-decision tree where η is the size of the tree. We now show that $\eta \leq 2^h$.

Since x_i appears in at least rh/n terms of f, for some of the values $\xi \in \{0,1\}$ the projection $f|_{x_i \leftarrow \xi}$ will be a decision list that has at most $r - (rh)/(2n)$ terms of size more than h. Therefore the size of the tree η is at most $\phi(n,r)$ with ϕ is the solution of the recurrence

$$\phi(n,r) = \phi(n-1,r) + \phi\left(n-1, r\left(1 - \frac{h}{2n}\right)\right)$$

where

$$\phi(1,\star) = 1 \text{ and } \phi(\star,0) = 1.$$

It is easy to see that such a recurrence gives

$$\phi(n,r) \le n^{\frac{2n}{h}\ln(r+1)} \le n^{\frac{2n}{h}\ln(s+1)} = 2^{O(h)}.$$

We now show that a $(1,2^h,h,2^h)$-decision tree is a $2h$-decision list. The proof is similar to Blum's proof in [Bl92]. Since the decision tree has size 2^h there is a leaf v in the tree of depth h. Let $x_{i_1} = \xi_1, \ldots, x_{i_q} = \xi_q$, $q \le h$, be the variables and the values that lead to this leaf and let $((T_1', b_1'), \ldots, (T_w', b_w'))$ be the decision list in that leaf. The $2h$-decision list will be constructed as follows. The prefix of the $2h$-decision list will be $L_1 = ((TT_1', b_1'), \ldots, (TT_w', b_w'))$ where $T = x_{i_1}^{\xi_1} \wedge \cdots \wedge x_{i_q}^{\xi_q}$ where $x^0 = \neg x$ and $x^1 = x$. The prefix of the $2h$-decision tree is constructed recursively for the decision tree that resultes from the removal of leaf v and its parent vertex in the tree (the node that contained x_{i_q}). The proof of correctness of this construction is identical to the proof by Blum in [Bl92].□

Acknowledgment. I would like to thank David Wilson and Tino Tamon for proof reading the paper. I also would like to thank Tino Tamon and Lisa Hellerstein for many helpful discussions and comments on the early version of this paper.

References

[A88] D. Angluin. Queries and Concept Learning. *Machine Learning*, 2(4):319–342, 1988.

[Bl92] A. Blum. Rank-r Decision Trees are a Subclass of r-decision Lists. *Information Processing Letters*, **43**, pages 183–185, 1992.

[BCKT94] N. H. Bshouty and R. Cleve and S. Kannan and C. Tamon. Oracles and Queries that are Sufficient for Exact Learning. *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory.* 1994, 130–139.

[HSW90] D. Helmbold, R. Sloan and M. K. Warmuth. Learning Nested Differences of Intersection-Closed Concept Classes. *Machine Learning*, 165–196, 5, 2, 1990.

[L88] N. Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear Threshold Algorithm. *Machine Learning*, pp. 285–318, v. 2, n. 4, 1988.

[R87] R. L. Rivest, Learning decision lists. *Machine Learning*, **2**, pages 229–246, 1987.

[Val84] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.