Learning nearly monotone k-term DNF*

Jorge Castro, David Guijarro and Víctor Lavín†

{castro,david,vlavin}@goliat.upc.es

Departament de Llenguatges i Sistemes Informàtics

Universitat Politècnica de Catalunya

Pau Gargallo, 5

08028 Barcelona, Spain

September 4, 1996

Abstract

This note studies the learnability of the class k-term DNF with a bounded number of negations per term. We study the case of learning with membership queries alone, and give tight upper and lower bounds on the number of negations that makes the learning task feasible. We also prove a negative result for equivalence queries. Finally, we show that a slight modification in our algorithm proves that the considered class is also learnable in the Simple PAC model, extending Li and Vitányi result for monotone k-term DNF.

*Research supported by the Esprit EC program under project 7141 (ALCOM-II), the Working Group 8556 (NeuroColt), and the Spanish DGICYT (project PB92-0709).

†Supported by FP93 13717942 grant from the Spanish Government.

1 Introduction

Among the different models of learning proposed in Computational Learning Theory , one of the most widely studied is the *exact learning via queries* model, introduced by Angluin in [Ang87b] and [Ang88].

In this model the learner's goal is to identify an unknown target function f, taken from some representation class $\mathcal{C}$. In order to get information about the target, the learner has available two types of queries: membership and equivalence queries. In a membership query, the learner supplies an instance x from the domain and gets $f(x)$ as answer. The input to an equivalence query is a hypotesis h, and the answer is either "yes" if $h \equiv f$ or a counterexample in the symmetric difference of f and h.

Many interesting results have been obtained within this framework using different combinations of queries. However, there is a great deal of problems that still remain open. In particular, one of the most challenging open problems is whether the class of DNF formulas is learnable. Although it is known that this class is not learnable using either membership or equivalence queries alone, not much can be said by now about learning using both types of queries.

Several subclasses of the class DNF have been studied from the point of view of learnability. We will focus our attention on the class k-term DNF, i.e., the class of DNF formulas whose number of terms is at most k.

We review now some of the results concerning this class. In [Ang87a], Angluin gives a polynomial time algorithm that learns k-term DNF with membership and equivalence queries. The algorithm is proper in the sense that it uses as hypotheses k-term DNF formulas. Improvements in the running time can be found in [BGHM93] and [Ber93].

On the other hand, it is known that the class k-term DNF cannot be learned with a polynomial number of membership queries. However, if we restrict that class to be monotone, then learning can be achieved in polynomial time. Thus, a natural question to ask is: What amount of non-monotonicity can we afford so that the task of learning is still feasible?

We answer this question by giving tight upper and lower bounds in the number of negations per term that make learning feasible. We prove that if the number of negations per term is bounded by a constant then we can learn in polynomial time, using membership queries. However, any bound that grows faster than a constant makes the learning task impossible in polynomial time. We also show that the number of terms is tight, i.e., if the number of terms grows faster that a constant learning becomes unfeasible. Both non-learnability proofs are information theoretic, which implies that the negative results hold even if we allow the learner to output a representation from any arbitrary class of hypotheses.

We also prove that any learning algorithm for the considered class requires a superpolynomial number of equivalence queries, if this is the only type of query allowed.

Our results, negative and positive, on membership queries apply also to k-term DNF formulas with a bounded number of positive literals per term, and to k-clause CNF formulas where the number of negative or positive literals per clause is bounded.

Finally, we consider the Simple PAC model introduced by Li and Vitányi in [LV91], and extend their result for monotone k-term DNF to the class considered in our paper.

The paper is organized as follows. In Section 2 we introduce notation and definitions needed along the paper. In Section 3 we describe our algorithm to learn (k, j)-DNF, and prove its correctness and time bounds. Section 4 is devoted to negative results. Section 5 extends our results to other classes and Section 6 deals with Simple PAC.

2 Definitions

As we have already mentioned, we use the model of exact learning via queries as it was proposed in [Ang88]. We refer the reader to that reference for the definitions of learnability, queries, etc.

We deal with boolean functions represented as disjunctions of conjunctions of literals (DNF). A literal is a variable, or the negation of a variable, and a term is a conjunction of literals. A k-term DNF formula is a disjunction of at most k terms. A (k, j)-DNF formula is a k-term DNF formula with at most j negated variables per term. We treat DNF formulas also as sets of terms.

Variables are considered to be indexed by natural numbers from 1 to n (where n is the number of variables over which the target class is defined). If A and B are two disjoint sets of indices of variables let $\mathrm{Term}(A, B)$ denote the term with all its non-negated variables taken from A and the negated variables taken from B.

For an assignment a, $\mathrm{ones}(a)$ denotes the set of indices of variables where a has a 1 and $\mathrm{zeros}(a)$ denotes the set of indices of variables where a has a 0. The function $\mathrm{flip}(a, i)$ produces a new assignment equal to a except in the i-th variable, where the value is changed.

We direct the reader to [LV91] for definitions related to Simple PAC.

3 A Positive Result

We present an algorithm that learns the class (k, j)-DNF in polynomial time, using only membership queries. This algorithm is based on [BGHM93]. In that paper Bshouty et al. give an algorithm that learns k-term DNF formulas with membership and equivalence queries. We show that equivalence queries are not necessary if we bound the number of negations per term. Our algorithm differs from theirs basically in the way we get counterexamples and how we produce terms from them.

Before giving a detailed description of the algorithm, we prove a simple lemma that will help us to show correctness and time bounds.

Lemma 1 *Let f and g be a pair of (k,j)-DNF formulas such that g does not imply f. Then there exists an assignment with at most $k+j$ zeros that satisfies g but not f.*

Proof.

Note that j zeros are enough to satisfy a term in g, and since the clauses on the CNF representation of f have at most k literals, then k zeros suffice to falsify f, so the lemma follows. $\square$

Let f^* be the target formula. We assume that k and j are known. The algorithm **Learn** (Figure 1) simulates the construction of a tree of depth at most k, where each node contains a (k,j)-DNF formula that implies f^*. Each node branches at most $2^k jn^j$; therefore, the size of the tree is at most $O(2^{k^2} j^k n^{jk})$.

For all levels i of the tree the following property holds: for each (k,j)-DNF representation f of f^* there is some node at level i that contains a function g of i terms that coincides with a subset of i terms of f. This property guarantees that at the end of the process some leaf of the tree will contain a (k,j)-DNF representation of f^*.

The first call to Learn is made with input parameter $g = False$. There is a global variable S containing the set $\{\langle a, f^*(a)\rangle : a \in \{0,1\}^n \text{ and } \|\text{zeros}(a)\| \leq k+j\}$.

Whenever we need to find a counterexample to extend a current hypothesis g, we search for an assignment $\langle a, 1\rangle \in S$ that satisfies $g(a) = 0$. Notice that Lemma 1, together with the fact that all the hypoteses we produce imply f^*, guarantee that such a counterexample will be found, if some exists (otherwise, we will have $g \equiv f^*$, and consequently stop the process and output g).

The next step is to discover the term in the target that is satisfied by a. To do that we run a "minimization" process that consists on flipping ones until no more can be flipped without violating the property "$f^*(a) = 1$ and $g(a) = 0$".

Let b be the minimized assignment. Two problems arise when we want to find a new term of the target formula from b. First, we do not know whether the target term contains negated literals (at most j). On the other hand, it might be that not all the ones of b would correspond to positive literals in the target term, since some of them might have been kept in order to falsify the terms discovered so far, that is, in order to maintain $g(b) = 0$ along the minimization process. The number of these ones is at most k, for we need to falsify at most k terms.

Therefore, we can consider b divided in three disjoint blocks: The block A of ones that keep $f^*(b) = 1$, all of which must appear as positive literals in the target term, the block B of at most k ones that guarantee $g(b) = 0$, some of which might be in the target term, and the block C of zeros, from which at most j might appear as negated literals in the target term. Note that blocks A and B are disjoint because g implies f^*.

Now we create a term $\text{Term}(A \cup B', C')$ for each $B' \subseteq B$ and for each $C' \subseteq C$ of size at most j. Notice that the number of terms we produce after a minimized positive example is at most $2^k \sum_{i=0}^{j} \binom{n}{i}$, which is less than $2^k jn^j$.

Finally, we use Lemma 1 to eliminate from the set of terms produced those that do not imply the target function.

```
procedure Learn (g)
    search ⟨a, 1⟩ ∈ S such that g(a) = 0
    if none is found then halt and output g
    else
        if |g| < k then
            let b the minimization of a with g
            T := Produce-terms(g, b)
            for all t ∈ T do
                Learn(g ∨ t)
            enfor
        endif
    endif
```

Figure 1: The Learning Algorithm

The function **Produce-terms** (Figure 2) gives us the terms associated to a positive assignment in the way described above.

All the previous considerations are formalized in the following lemma:

Lemma 2 *If g is a (k,j)-DNF formula that implies f^* but not viceversa, b a minimized assigment that witnesses that fact, and T the result of* **Produce-terms**(g,b) *then*

- *all the terms in T imply f^*, and*
- *for any (k,j)-DNF representation of f^* there is a term of it that is in T but not in g.*

Once we have got the set T of terms that are possible extensions of g, we recursively call Learn with parameter $g \vee t$ for each $t \in T$.

Lemma 2 guarantees that in the k-th level of the tree there will be a representation equivalent to the target, so the correctness of Learn follows.

Query complexity. We ask at most $(k+j)n^{k+j}$ membership queries to create S, and at most n^2, for each node of the tree, to minimize the counterexample. This is at most $(k+j)n^{k+j} + 2^{k^2}j^k n^{jk+2}$ queries, which is polynomial in n. In section 6 we will show how to reduce that number to $(k+j)n^{k+j}$.

Time complexity. In each node we explore S at most $1 + 2^k j n^j$ times to look for the counterexample and to test if the produced terms imply the target. We also spend n^2 steps minimizing the counterexample. This is at most $O(n^{j(k+2)+k})$.

```
function Produce-terms(g, a)
    T := ∅
    for each X ⊆ {i : i ∈ ones(a) and g(flip(a, i)) = 1} do
        for each Y ⊆ zeros(a) such that ‖Y‖ ≤ j do
            add Term(ones(a) − X, Y) to T
        end-for
    end-for
    delete from T the terms that do not imply the target function
    return T
```

Figure 2: Obtaining the terms

Theorem 3 *(k, j)-DNF are properly learnable with membership queries in polynomial time.*

4 Lower Bounds

In this section we prove superpolynomial lower bounds on the number of queries needed to learn the class (k, j)-DNF when k or j are not constants.

In the case of membership queries we prove that our positive result is tight in two directions: we cannot afford either more than a constant number of terms or more than a constant number of negations per term.

For the first case we use a target class in [BC+95] and for the second we simply count the number of singleton sets that can be represented with a number of negations per term that is bigger than any constant.

Negative results for membership queries can be of two kinds: information theoretic and computational. Our results are of the first type, which implies that learning is unfeasible no matters what representation class is used as hypotheses class.

For equivalence queries we describe an adversary that forces a superpolynomial number of queries. The result is also information theoretic. In the case of equivalence queries this does not imply non-learnability using any hypotheses class. However, our proof of Theorem 6 implies that (k, j)-DNF are not learnable even if we use general k-term DNF as hypothesis.

4.1 Membership Queries

Theorem 4 *There is no polynomial time algorithm that learns* $(f(n), 0)$*-DNF formulas using membership queries, if* f *grows faster than any constant.*

Proof(Sketch).

Supposse that the class is learnable with n^c membership queries. We use the class T_n of part (1) of Theorem 22 in [BC+95]. Let m be $c+1$, l be n/m and T_n be $\{f : f = t \vee \bigvee_{i=1}^{m} t_i\}$, where each $t_i = \bigwedge_{j=(i-1)l+1}^{il} x_j$ and t is a monotone term that contains all variables except one from each t_i. Observe that T_n is a subset of $(f(n), 0)$-DNF for n such that $f(n) > c+1$. We choose n large enough such that $n^c < l^m$. Following the same argument as in [BC+95] one can show that l^m queries are necessary to learn T_n, which proves the theorem. $\Box$

We will prove that the number of negations we allow per term is also optimal.

Theorem 5 *There is no polynomial time algorithm that learns* $(1, f(n))$*-DNF formulas using membership queries, if* f *grows faster than any constant.*

Proof.

The number of different singleton sets we can represent with $(1, f(n))$-DNF formulas is

$$\sum_{i=0}^{f(n)} \binom{n}{i} = S_f$$

We use a standard adversary argument: a teacher that answers "no" to the first $S_f - 1$ membership queries, forces any learner to ask at least $S_f - 1$ membership queries. Since S_f grows faster that any polynomial the theorem follows. $\Box$

4.2 Equivalence Queries

Theorem 6 *For any* $k > 1$ *and* $j \geq 0$*, the class* (k, j)*-DNF is not learnable using a polynomial number of equivalence queries that are* k*-term DNF (without the* j *restriction).*

Proof.

Let $T_{n,k}$ be the target class of functions where each function is the disjunction of k pairwise disjoint monotone terms of length n/k. Here, disjoint means that no pair of terms have a variable in common.

It can be shown that $|T_{n,k}| = \frac{n!}{((n/k)!)^k k!}$. We will prove that for any k-term DNF formula h there exists an assignment a such that the number of functions in $T_{n,k}$ that agree with h in the classification of a is a "small" fraction of $|T_{n,k}|$. Therefore, providing that assignment as a counterexample to an equivalence query on input h will force any learner to make a "large" number of equivalence queries, before finding out the target. To see that some a satisfying these requirements exists we consider the following cases:

1. $|h| < k$. Then h is falsified by some assignment containing less than k 0's. Clearly, such assignment does not falsify any of the functions in $T_{n,k}$.

2. $|h| = k$ and some term t of h has at most n/k positive literals. The assignment that satisfies those positive literals and has 0's in the remaining positions is a positive example for h. However, the fraction of formulas in T_n satisfied by such assignment is at most
$$\frac{|T_{n-n/k,k-1}|}{|T_{n,k}|} = \frac{(n-n/k)!(n/k)!k}{n!}$$
which is smaller than $1/n^c$ for any constant c, and n large enough.

3. Otherwise, since all the terms in h have more than n/k positive literals, at least two of them must share one positive literal. Therefore h can be falsified by some assignment with less than k 0's as in case 1.

□

5 Applications to Other Classes

The algorithm in Figure 1 can be used to learn k-term DNF formulas with at most j positive literals per term; it is enough to exchange the roles of ones and zeros. Furthermore, lower bounds also hold for this class.

In the case of k-clause CNF formulas with a bounded number of negative (positive) literals per clause, the algorithm is also appliable; it suffices to consider the complementary concept which is a k-term DNF formula with a bounded number of positive (negative) literals per term.

This later observation implies that negative results also hold for the considered subclasses of k-clause CNF.

Furthermore, we can combine the positive results to learn a subclass of k-term DNF (CNF) where each term (clause) has either at most j negative variables or at most j positive variables.

6 Simple PAC

This section is devoted to the proof of learnability of (k, j)-DNF in the Simple PAC setting. In [LV91], Li and Vitányi prove that the class of monotone k-term DNF is Simple Pac learnable. In fact, they prove a stronger result, since their algorithm outputs, with high probability, an exact representation of the target function, instead of producing a "good" approximation. The key observation in their proof is that sampling according to the universal distribution, all the assignments with at most k zeros are in a polynomial-sized sample, with high probability.

We will proceed in the same manner for the class (k, j)-DNF. We ask for a polynomial-sized sample and get, with high probability, all the assignments with at most $k+j$ zeros. If the sample contains all those assignments we run Learn, otherwise we do nothing. Whenever Learn asks a membership query for an assignment not in the sample (as it might occur in the minimization process), we answer it correctly by ourselves using Lemmas 1 and 7.

Lemma 7 *Let f be a DNF with at most j negations per term. For any assignment a, $f(a) = 1$ if and only if there exists a term t with at most j negations such that $t(a) = 1$, t implies f, and the set of positive variables in t is ones(a).*

Proof.

The if part is trivial. For the only if part, let t' be a term in f that accepts a. Clearly the positive variables of t' are a subset of ones(a). We define t as a term with the negated variables taken from the negated variables of t' and with positive variables all ones(a). It is trivial to see that $t(a) = 1$. It is also easy to show that t implies f, since t implies t' and t' implies f. Moreover, ones(a) are the positive variables of t by construction. $\square$

Let b be the input assignment to a membership query in the minimization process. Now, we construct all the terms that have as positive literals the variables in ones(b) and at most j negations. Then, we use lemma 1 to test if any of those terms implies the target function. If none of them implies the target we answer NO, otherwise we answer YES.

So we have proven the following theorem

Theorem 8 *(k, j)-DNF are Simple PAC learnable.*

Note that this procedure of simulating membership queries could be used to improve the query complexity stated in section 3, since the initial $(k+j)n^{k+j}$ membership queries would suffice, although at the price of increasing the time complexity.

7 Acknowledgments

We thank José L. Balcázar and Ricard Gavaldà for helpful comments.

References

[Ang87a] D. Angluin. "Learning k-term DNF formulas using queries and counterexamples". Technical Report YALEU/DCS/RR-559, Yale University, August 1987.

[Ang87b] D. Angluin. "Learning regular sets from queries and counterexamples". Information and Computation, 75:87-106, November 1987.

[Ang88] D. Angluin. "Queries and concept learning". Machine Learning, 2(4):319-342, 1988.

[Ber93] Ulf Berggren. "Linear Time Deterministic Learning of k-term DNF" In Proc. 6th Annu. Workshop on Comput. Learning Theory, pages 37-40. ACM Press, New York, NY, 1993.

[BC+95] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. "Oracles and queries that are sufficient for exact learning". Electronic Colloquium on Complexity. Technical Report TR95-015, 1995.

[BGHM93] N. Bshouty, S. Goldman, T. Hancock, and S. Matar. "Asking queries to minimize errors". In Proc. 6th Annu. Workshop on Comput. Learning Theory, pages 41-50. ACM Press, New York, NY, 1993.

[LV91] M. Li and P. Vitányi. "Learning simple concepts under simple distributions". SIAM Journal of Computing 20, pages 911-935, 1991.