On the Learnability of Boolean Formulae

Michael Kearns
Harvard University

Ming Li
Harvard University

Leonard Pitt
University of Illinois

Leslie Valiant
Harvard University

1 Introduction

We study the computational feasibility of learning boolean expressions from examples. Our goals are to prove results and develop general techniques that shed light on the boundary between the classes of expressions that are learnable in polynomial time and those that are apparently not. The elucidation of this boundary, for boolean expressions and possibly other knowledge representations, is an example of the potential contribution of complexity theory to artificial intelligence.

We employ the distribution-free model of learning introduced in [10]. A more complete discussion and justification of this model can be found in [4,10,11,12]. [4] includes some discussion that is relevant more particularly to infinite representations, such as geometric ones, rather than the finite case of boolean functions. For other recent related work see [1,2,7,8,9].

The results of this paper fall into three categories: closure properties of learnable classes, negative results, and distribution-specific positive results.

The closure properties are of two kinds. In section 3 we discuss closure under boolean operations on the members of the learnable classes. The assumption that the classes are learnable from positive or negative examples alone is sometimes sufficient to ensure closure. In the subsequent section, we give a general substitution technique. It can be used to show, for example, that if disjunctive normal form (DNF) formulae are learnable in the monotone case, then they are also learnable in the unrestricted case.

In section 5 we prove some negative results. One of them shows that for purely information-theoretic reasons, simple monomials cannot be learned from negative examples alone. This contrasts with the fact that monomials can be learned from positive examples alone [10], and can be learned from very few examples if both kinds are available [7]. The remaining results in the section are all predicated on the computational hypothesis NP $\neq$ RP. The classes shown not to be learnable include disjunctions of two monomials, formulae where each variable occurs just once (μ-formulae), and boolean threshold functions. Negative results are also given for heuristic learning, where the rule to be learned needs to account correctly for only a fraction of the examples.

In the final section, we consider learning μDNF and monotone DNF, under the restriction that both the positive and negative examples are drawn from uniform distributions. In the distribution-free setting, the two questions are equivalent (shown in section 4) and unresolved. We show that μDNF is learnable in this distribution-specific case. In fact, the learning algorithm can be used to learn 2-term μDNF, which is NP-hard in the distribution-free case (section 5). We also show that monotone DNF is at least group learnable. In group learning, it is sufficient to deduce a rule for recognizing whether a large enough set of examples contains only positive or only negative examples, given that one of the two possibilities holds.

This research was supported by grants NSF-DCR-8600379 and ONR-N00014-85-K-0445. The third author was also supported by the Computer Science Dept., University of Illinois at Urbana-Champaign, and the fourth author by a Guggenheim Fellowship. Authors' addresses: Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138 and Computer Science Dept., University of Illinois, Urbana, IL 61801.

2 Definitions

Let n be any natural number. A *concept* F is a boolean function with domain $\{0,1\}^n$. Those vectors $\vec{v}$ such that $F(\vec{v}) = 1$ are called positive examples, the rest are negative examples of F. For any F, there are many possible boolean formulae f such that f is consistent with the concept F. A *class of representations* of concepts is a set $A = \cup_{n=1}^{\infty} A_n$ where for each n, A_n is a subset of all possible formulae of n variables. For example, for each constant k, kDNF $= \cup_{n=1}^{\infty}\{k\text{DNF}$ over n variables$\}$ is a class of representations, where kDNF denotes disjunctive normal form in which each disjunct consists of at most k literals. Since learnability may depend on the representation chosen, we define learnability of a class of *representations* of boolean concepts, as opposed to defining learnability of the concepts themselves.

If A is a class of representations, then for each $f \in A$, let *size*(f) denote the fewest number of symbols needed to write the representation f in A.

We assume that the learning algorithm has available a black box called EXAMPLES(f), with two buttons labeled POS and NEG. If POS (NEG) is pushed, a positive (negative) example is generated according to some fixed but unknown probability distribution D^+ (D^-). We assume nothing about the distributions D^+ and D^-, except that $\sum_{f(\vec{x})=1} D^+(\vec{x}) = 1$ and $\sum_{f(\vec{x})=0} D^-(\vec{x}) = 1$ (i.e., $\sum_{f(\vec{x})=1} D^-(\vec{x}) = 0$ and $\sum_{f(\vec{x})=0} D^+(\vec{x}) = 0$).

Definition 1 *Let A be a class of representations. Then A is* learnable from examples *iff there exists a polynomial p, and a (possibly randomized) learning algorithm L, such that $(\forall n)(\forall f \in A_n)(\forall D^+, D^-)$ $(\forall \epsilon > 0)$, L, given only EXAMPLES(f), halts in time $p(n, size(f), \frac{1}{\epsilon})$ and outputs a formula $g \in A_n$ that with probability at least $1 - \epsilon$, has the following properties:*

$$\sum_{g(\vec{x})=0} D^+(\vec{x}) < \epsilon$$

and

$$\sum_{g(\vec{x})=1} D^-(\vec{x}) < \epsilon.$$

Throughout the paper, we shall abbreviate the phrase "is learnable from examples" by the phrase "is learnable". We shall call ϵ the *error parameter* of the learning algorithm.

If there is an algorithm L as above which never asks for any negative (resp. positive) examples, then we'll say that A is learnable from positive (negative) examples only.

This definition can be understood as follows. We think of Nature as providing examples of the formula to be learned according to some unknown probability distribution for which we can make no assumptions. Since there may be very bizarre examples of the formula which occur with low probability, it is unreasonable to expect the learning algorithm to produce a formula which correctly classifies all examples. Hence a successful formula g is one which agrees with the unknown f on *most* of the distribution. That is, the probability that the formula g disagrees with either a positive or a negative example is at most ϵ in either case. A second source of error is introduced by the possibility that the particular sequence of examples provided by Nature is highly unrepresentative. In this case, it is reasonable that the formula g be highly inaccurate. We require that this occur with probability at most ϵ. A further requirement of the definition is that the run time of the algorithm be polynomial in the size of the formula to be learned and in the inverse of the error parameter.

A generalization of the above definition allows the formula output to be from a different class of representations: the class A is *learnable by the class B* iff Definition 1 holds, except that $g \in B_n$. In general, the larger the class A, the harder the learning task, but for fixed A, the larger the class B, the easier the learning task. Thus $A \subseteq B$ does not imply anything about the learnability of A by A as compared to the learnability of B by B.

3 Boolean Closure

These results provide tools for determining learnability of new classes of formulae. Let A and B denote classes of representations of boolean concepts. (For example, k-term-DNF, monomials, etc.)

Theorem 1 *If A is learnable, and if B is learnable from negative examples only, then $A \vee B = \{f_1 \vee f_2 : f_1 \in A, f_2 \in B\}$ is learnable.*

Proof: Let $f = f_1 \vee f_2$ be a formula in $A \vee B$, where $f_1 \in A$ and $f_2 \in B$, and let D^+ and D^- be the unknown distributions on positive and negative examples (respectively) for f. Since $\{\vec{v} : f(\vec{v}) = 0\} \subseteq \{\vec{v} : f_2(\vec{v}) = 0\}$, we can learn a formula $g \in B$ for f_2 from negative examples only, using the examples generated by D^-. We call the learning algorithm for B with error parameter $\frac{\epsilon}{2}$ to obtain a g that has probability of error at most $\frac{\epsilon}{2}$ on the distribution D^-. At this point, we must determine how well g performs as an approximation to f on the positive examples of f. We use the following bounds on the tails of binomial distributions [2,3]:

For $0 \leq p \leq 1$ and m a positive integer, let $LE(p, m, r)$ denote the probability of at most r successes in m inde-

pendent trials of a Bernoulli variable with probability of success p, and let $GE(p, m, r)$ denote the probability of at least r successes. Then for $0 \le \beta \le 1$,
Fact 1. $LE(p, m, (1-\beta)mp) \le e^{-\beta^2 mp/2}$ and
Fact 2. $GE(p, m, (1+\beta)mp) \le e^{-\beta^2 mp/3}$
We will make use of these facts throughout the paper.

Let q be the probability (according to D^+) that a random positive example of f does not satisfy g. We now take $\frac{1}{\epsilon^2}$ positive examples from D^+ in order to estimate the value of q. From Fact 1, if $q \ge \epsilon$, at least a fraction $\frac{\epsilon}{2}$ of our positive examples will not satisfy g with probability at least $1 - e^{-1/8\epsilon}$. From Fact 2, and the fact that GE is a nondecreasing function for increasing p, we know that if $q \le \frac{\epsilon}{4}$, then at most a fraction $\frac{\epsilon}{2}$ of our positive examples will not satisfy g with probability at least $1 - e^{-1/12\epsilon}$. Thus, if at least a fraction $\frac{\epsilon}{2}$ of the $\frac{1}{\epsilon^2}$ positive examples do not satisfy g, then $q \ge \frac{\epsilon}{4}$ with overwhelming probability. Otherwise, g is already a good approximation to f, and we are done.

If it is estimated that $q \ge \frac{\epsilon}{4}$, then use the algorithm for learning A from examples as follows: When a negative example is requested, supply one using D^-. When a positive example is requested, search for a positive example generated from D^+ which does not satisfy g. From the above argument, the probability of not drawing such an example in $\frac{4}{\epsilon^2}$ tries is at most

$$(1 - \frac{\epsilon}{4})^{4/\epsilon^2} \quad < \quad e^{-1/\epsilon}$$

Thus, the algorithm for learning A must output a good approximation h for f_1 when called with error parameter $\frac{\epsilon}{2}$, and then $h \vee g$ is an approximation for f, with probability of error at most ϵ. Note that if $p_B(n, \frac{1}{\epsilon})$ is the time complexity of the learning algorithm for class B, and $p_A(n, \frac{1}{\epsilon})$ the time complexity of the algorithm for class A, then the complexity for learning $A \vee B$ by the above procedure is $p_B(n, \frac{2}{\epsilon}) + O(\frac{1}{\epsilon^2}) + \frac{4}{\epsilon^2} p_A(n, \frac{2}{\epsilon})$. □

Theorem 2 *If A and B are each learnable from positive examples only, then $A \wedge B = \{f_1 \wedge f_2 : f_1 \in A, f_2 \in B\}$ is learnable from positive examples only.*

Proof: Let $f = f_1 \wedge f_2$ be a formula in $A \wedge B$, where $f_1 \in A$ and $f_2 \in B$, and let D^+ and D^- be the unknown distributions on the positive and negative examples (respectively) for f. Since $\{\vec{v} : f(\vec{v}) = 1\} \subseteq \{\vec{v} : f_1(\vec{v}) = 1\}$, we can learn a formula $g \in A$ for f_1 using positive examples generated by D^+. By calling the learning algorithm for A with error parameter $\frac{\epsilon}{2}$, we force g to cover at least a fraction $1 - \frac{\epsilon}{2}$ (according to D^+) of the positive examples of f with high probability. Similarly, we can learn a formula $h \in B$

$A \vee B$	A learnable from POS	A learnable from NEG	A learnable from POS and NEG
B learnable from POS	NO	YES	NO
B learnable from NEG	YES	YES	YES
B learnable from POS and NEG	NO	YES	NO

Figure 1: Learnability of $A \vee B$.

$A \wedge B$	A learnable from POS	A learnable from NEG	A learnable from POS and NEG
B learnable from POS	YES	YES	YES
B learnable from NEG	YES	NO	NO
B learnable from POS and NEG	YES	NO	NO

Figure 2: Learnability of $A \wedge B$.

for f_2 from positive examples only that has this same accuracy on D^+. Then $g \wedge h$ covers a fraction $1 - \epsilon$ of the positive examples of f with high probability. Furthermore, since f_1 (respectively, f_2) is learnable from positive examples only, g (respectively, h) can be satisfied by no negative examples of f_1 (respectively, f_2). Thus, $g \wedge h$ makes no errors on the negative examples of f. □

There are natural duals for both Theorems 1 and 2 (switch "$\wedge$" and "$\vee$" wherever they occur, as well as "positive" and "negative"), and together with Theorem 9, we have two tables (Figures 1 and 2) for the learnability of $A \wedge B$, and $A \vee B$, given the learnability of A and the learnability of B. In the tables, we label the rows and columns according to whether each of A and B is learnable from positive examples only, negative examples only, or from both positive and negative examples. We then label the corresponding learning problem ($A \vee B$ for Figure 1, $A \wedge B$ for Figure 2) by YES (if we can always learn in polynomial time) or NO (if the learning problem is NP-hard for some pairs (A, B)). Note that Theorem 1 is optimal in that we cannot relax the constraints on B to allow B to be learnable from both positive and negative examples.

As corollaries we can deduce the learnability of classes not previously known to be learnable. For example:

Corollary 3 *$\{f \vee g : f \in kCNF, g \in kDNF\}$ is learnable.*

Corollary 4 *$\{f \wedge g : f \in kDNF, g \in kCNF\}$ is learnable.*

Here kCNF is the class of conjunctive normal form formulae in which each clause contains at most k literals. Proofs for these corollaries follow from Theorem 1 and the algorithm in [10] for learning kCNF from positive examples only.

4 A Substitution Argument

We describe a simple substitution argument that shows that learnable classes are closed under certain kinds of substitution.

Theorem 5 *Let C be learnable, and let $\mathcal{G}$ be a finite collection of boolean formulae over k (constant) variables. Let C' be the class of all formulae that can be obtained by choosing an $f(x_1,\ldots,x_n) \in C$, and replacing one or more of the variables x_i with any formula $g_i(x_{i_1},\ldots,x_{i_k})$, where $g_i \in \mathcal{G}$, and each x_{i_j} is one of the original n variables (thus, the resulting formula is still over n variables). Then C' is also learnable.*

Proof: We sketch the proof for the case that $\mathcal{G}$ contains only the single formula g. The general case is similar. Let M_C be the learning algorithm for the class C. Create new variables $z_1,\ldots,z_{n^k}$. Suppose we are given a positive (negative case identical) example $(b_1,\ldots,b_n) \in \{0,1\}^n$ of a formula $f'(x_1,\ldots,x_n) \in C'$. The intention is that z_i will simulate the value of the formula g with the ith choice of k inputs from $\{x_1,\ldots,x_n\}$ (note that there are exactly n^k such choices). Let $c_i \in \{0,1\}$ be the value assigned to z_i by this simulation. Then we give the example $(b_1,\ldots,b_n,c_1,\ldots,c_{n^k})$ to algorithm M_C. Since there is a formula in C that is consistent with all the samples we generate (it is just f' with each occurrence of the formula g replaced by the z_i that simulates the inputs to the occurrence of g), M_C must output a good hypothesis $\tilde{f} \in C$. We then obtain a good approximation for f' over n variables by replacing each occurrence of a z_i with an occurrence of g with the ith choice of inputs. □

Theorem 6 *Let C be learnable. Let $p(n)$ be a fixed polynomial, let f be a formula in C over $n + p(n)$ variables, and let the description of the $p(n)$-tuple $(B_1^n,\ldots,B_{p(n)}^n)$ be polynomial time computable from n, where each B_i^n is a boolean formula of n inputs. Then the class C' of formulae of the form*

$$f(x_1,\ldots,x_n,B_1^n(x_1,\ldots,x_n),\ldots,B_{p(n)}^n(x_1,\ldots,x_n))$$

is also learnable.

Proof: Similar to the proof of Theorem 5. □

An important corollary of both Theorems 5 and 6 is that the monotone learning problem is always as hard as the general learning problem (for classes that are closed under the required substitutions). For example:

Corollary 7 *If monotone DNF is learnable, then DNF is learnable.*

Proof: In the statement of Theorem 6, let C be the class of monotone DNF formulae, and let $B_i^n(x_1,\ldots,x_n) = \overline{x_i}$ for $1 \le i \le n$. The resulting class C' is simply DNF. □

Another example is that learning DNF is as hard as learning depth 3 formulae with constant fanout "ORs" at the bottom layer.

Corollary 8 *Let C be a class of formulae. Let μC be the class of those formulae in C in which each variable occurs at most once. Then if μC is learnable, then C is learnable.*

Proof: In the statement of Theorem 6, let $f(x_1,\ldots,x_n,x_{n+1},\ldots,x_{n+p(n)})$ be a formula in C in which each variable occurs at most once. Then setting $B_i^n(x_1,\ldots,x_n) = x_j$ for some $1 \le i \le p(n)$ and some $1 \le j \le n$ yields a formula in which x_j occurs at most twice. Perform this substitution as many times as necessary to allow extra occurrences of the original n variables. □

Corollary 8 is surprising, in that allowing only one occurrence of each variable is a strong restriction. This theorem states that the general learning problem is no harder than this restricted version. One of our results given later is that if A is the class of arbitrary formulae, then μA is not learnable (Theorem 11).

5 Negative Results

All negative results except Theorem 15 assume that RP $\neq$ NP.

For each constant number k, let k-term DNF be the class of formulae representable in DNF with at most k terms. While 1-term DNF is known to be learnable, we have:

Theorem 9 *For all integers $k \ge 2$, (monotone) k-term-DNF is not learnable by k-term-DNF.*

Proof: We define a generalization of the graph k-colorability problem [6]. The k-NM-colorability problem is described by:

Instance: A finite set S and a collection $C = \{c_1, c_2, \ldots, c_m\}$ of constraints $c_i \subseteq S$.

Question: Is there a k-coloring of the elements of S (*i.e.* a function $\chi : S \rightarrow \{1, 2, \ldots, k\}$), such that for each constraint $c_i \in C$ the elements of c_i are Not Monochromatically colored (*i.e.* $(\forall c_i \in C)(\exists x, y \in c_i)$ such that $\chi(x) \neq \chi(y)$)?

Lemma For all integers $k \geq 2$, k-NM-colorability is NP-complete.

The lemma is proved by reduction from Satisfiability. To prove Theorem 9, we reduce k-NM-coloring to the k-term-DNF learning problem. Let (S, C) be an instance of k-NM-coloring. We construct a k-term-DNF learning problem as follows:

Each instance will correspond to a particular k-term-DNF formula to be learned. We must describe what the positive and negative examples are, as well as the distributions D^+ and D^- .

If $S = \{s_1, s_2, \ldots, s_n\}$ then we will have n variables $\{x_1, x_2, \ldots x_n\}$ for the learning problem. The set of positive examples will be the vectors $\{\vec{p_i}\}_{i=1}^{n}$, where $\vec{p_i}$ is the vector with $x_i = 0$ and all other bits set to 1. The distribution D^+ will be uniform over these n positive examples, with each $\vec{p_i}$ occurring with probability $\frac{1}{n}$. We'll form $|C|$ negative examples, $\{\vec{n_i}\}_{i=1}^{|C|}$, each occurring with probability $\frac{1}{|C|}$ in the distribution D^-. For each constraint $c \in C$, if $c = \{s_{i_1}, s_{i_2}, \ldots, s_{i_m}\}$, then the negative example $\vec{n_i}$ is the vector which is all "1's" except at positions $i_1, i_2, \ldots, i_m$. Thus the constraint $\{s_1, s_3, s_8\}$ gives rise to the negative example $\langle 01011110111\ldots \rangle$. The theorem follows (by appropriate choice of ϵ) from the claim that there is a k-term-DNF formula consistent with all of the positive and negative examples above iff (S, C) is k-NM-colorable. To prove this claim, assume (S, C) is k-NM-colorable by a coloring $\chi : S \rightarrow \{1, 2, \ldots, k\}$ that uses every color at least once. Let f be the k-term-DNF expression $f = T_1 + T_2 + \ldots + T_k$, where the i^{th} term T_i is defined by

$$T_i = \prod_{\chi(s_j) \neq i} x_j.$$

In other words, the i^{th} term is the conjunction of all variables x_j for which the corresponding element s_j is not colored i. Then it is easy to show that f is consistent with all positive and negative examples. On the other hand, suppose that $T_1 + T_2 + \cdots + T_k$ is a k-term-DNF formula which is satisfied by all of the positive examples and no negative example. Then note without loss of generality, each T_i is a product of positive literals only: If T_i contains two or more negated variables, then none of the positive examples can satisfy it (since they all have only a single "0"), so it may be eliminated. If T_i contains exactly one negative literal $\overline{x_j}$ then it can be satisfied by at most the single positive example $\vec{p_j}$ so T_i can be replaced with $T'_i = \prod_{j \neq i} x_j$ which is satisfied by only the vectors $\vec{p_j}$ and the vector $\vec{1}$, neither of which are negative examples. Now color the elements of S by the function $\chi : S \rightarrow \{1, 2, \ldots, k\}$ defined by $\chi(s_i) = \min\{j :$ literal x_i does not occur in term $T_j\}$. Note that χ is well defined: since each positive example satisfies the formula $T_1 + T_2 + \cdots + T_k$, each positive example $\vec{p_i}$ must satisfy some term T_j. But each term is a conjunct of only positive literals, therefore for some j, x_i must not occur in term T_j. Thus each element of S receives a color. Furthermore, it is easy to show that if χ violates some color constraint, then some negative example satisfies one of the terms T_j. □

Corollary 10 *For all integers $k \geq 6$, (monotone) k-term-DNF is not learnable by $(2k-5)$-term- DNF.*

Proof: The difficulty of learning k-term-DNF stems from the NP-hardness of a generalization of the graph k-colorability problem. In fact, we could have used graph k-colorability directly to obtain the same result for $k \geq 3$. (However, the NP-hardness of 2-NM-colorability proves useful in a number of other constructions.) It is shown in [5] that for every $\epsilon > 0$, unless P = NP, no polynomial time algorithm can approximate the fewest number of colors needed to color a graph within a constant factor of $2 - \epsilon$. The proof in that paper, together with the proof of Theorem 9 above, provides a proof of Corollary 10. These results show that the problem of learning from examples a small DNF expression is at least as hard as the coloring approximation problem, and appears to be harder. □

Thus, even when the unknown formula is the sum of two monotone terms, it is NP-hard to find a two term (possibly non-monotone) expression for the examples seen. Corollary 10 is even stronger, and states that finding a good formula from examples is hard even if we allow the formula found to have roughly twice as many disjuncts as the formula to be learned.

A boolean tree is a circuit which is a tree, with the input variables as leaves. Boolean trees are equivalent to μ-formulae, the class of formulae in which each variable occurs at most once. It is surprising that for this simple type of function, we have the following result.

Theorem 11 *μ-formulae (boolean trees) are not learnable.*

Proof: We reduce the 2-NM-colorability problem to the μ-formula learning problem. Let (S, C) be an instance of 2-NM-colorability, and let $\{\vec{p_i}\}, \{\vec{n_i}\}$ be the positive and negative examples corresponding to (S, C) as in the proof of Theorem 9. By reasoning as in the proof of Theorem 9, we need only prove the claim that there is a μ-formula consistent with the positive and negative examples if and only if (S, C) is 2-NM-colorable.

To prove the claim, observe that if (S, C) is 2-NM-colorable, then (from the proof of Theorem 9) the formula $T_1 + T_2$ is in fact a μ-formula. Conversely, suppose that there is a μ-formula f consistent with the examples $\{\vec{p_i}\}$ and $\{\vec{n_i}\}$. Then consider the boolean tree T_f which computes f. T_f is a rooted directed tree with variables as leaves, internal nodes labeled with "AND", "OR", and "NOT", and edges directed away from the root. By a lemma we may assume without loss of generality that no node has the same label as its immediate ancestor, that none of the nodes are labeled "NOT", and that each variable occurs as a leaf. There are two cases to consider, depending on the label of the root of the tree. In each case we show how a 2-NM-Coloring may be found from T_f.
CASE 1: The root node is labeled "OR".

Then T_f is equivalent to the formula $f_1 + f_2 + \cdots + f_k$, where f_i is the subformula computed by the subtree of T_f with T_f's i^{th} child as root. Let $L = f_1$ and $R = f_2 + \cdots + f_k$. Color each element s_i with color C_L iff x_i occurs in formula L, otherwise color s_i with color C_R. We show that this is a legitimate 2-NM-Coloring: Suppose a constraint $c_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_m}\}$ is violated. Then all of $s_{i_1}, s_{i_2}, \ldots, s_{i_m}$ are colored the same color, and all of $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$ occur in the same subformula, say L without loss of generality. Then since formula R contains only positive literals, and does not contain the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$, it follows that $\vec{n_i}$ satisfies formula R, and therefore satisfies f, a contradiction.
CASE 2: The root node is labeled "AND".

Since there are no "NOT"s in the tree, there must be an OR on the path from the root to each x_i, otherwise the positive example $\vec{p_i}$ couldn't satisfy T_f Thus there are $k \geq 2$ OR nodes which are children of the root AND node. We divide the subtree beneath the i^{th} OR into two groups, L_i and R_i, where L_i is the function computed by the leftmost subtree of the i^{th} OR, and R_i is the function computed by the OR of the remaining branches of the i^{th} OR.

Thus $f = (L_1 + R_1)(L_2 + R_2) \cdots (L_k + R_k)$. Then let $f' = L + R$, where $L = L_1 L_2 \cdots L_k$ and $R = R_1 R_2 \cdots R_k$. Clearly $f' \Rightarrow f$, therefore no negative example satisfies f'. Now color s_i with color C_L iff x_i occurs in formula L, and color it C_R otherwise. By the same argument as in Case 1, if some coloring constraint is violated, then all of the elements of the constraint occur in the same subformula, and then the other subformula is satisfied by the negative example associated with the given constraint. □

A boolean threshold function may be thought of intuitively as follows. Among the set of n variables $\{x_i\}$ there is some *important subset* Y for the concept to be learned. There is also a critical threshold k such that whenever an example $\vec{x}$ has at least k of the bits of Y set to 1, it is a positive example, otherwise it is a negative example. We write this rule as $TH_k(\vec{y})$ where $\vec{y}$ is the characteristic vector for the set Y. Thus $\vec{x}$ is a positive example iff it satisfies $\vec{x} \cdot \vec{y} \geq k$.

Theorem 12 *Boolean threshold functions are not learnable.*

Proof: We reduce the NP-complete Zero-One Integer Programming problem (ZIP) to the learning problem [6]. An instance of ZIP is a set of s pairs $\vec{c_i}, b_i$ and the pair $\vec{a}, B$, where $\vec{c_i} \in \{0,1\}^n$, $\vec{a} \in \{0,1\}^n$, $b_i \in \{0,1\}$, and $0 \leq B \leq n$. The problem is to determine if there exists a vector $\vec{d} \in \{0,1\}^n$ such that $\vec{c_i} \cdot \vec{d} \leq b_i$ for $1 \leq i \leq s$ and $\vec{a} \cdot \vec{d} \geq B$.

Let $\vec{0}$ and $\vec{1}$ denote the all 0 and all 1 vectors, respectively. Similarly, let $\vec{1}_{i,j,\ldots}$ denote the vector which is set to 1 *only* at the positions $i, j, \ldots$. and $\vec{0}_{i,j,\ldots}$ denote the vector which is 0 only at the positions $i, j, \ldots$.

Given an instance of ZIP, we construct a boolean threshold learning problem. We'll have $2n$ features $x_1, x_2, \ldots, x_{2n}$. We will sometimes write a vector of length $2n$ as the concatenation of two vectors $\vec{x}, \vec{y}$ of length n, and denote this by $(\vec{x}, \vec{y})$. There are two positive examples, $\vec{p_1} = (\vec{0}, \vec{1})$, and $\vec{p_2} = (\vec{a}, \vec{1}_{1,2,\ldots,n-B})$. There are two types of negative examples. First, for each of the vectors $\vec{c_i}$, $1 \leq i \leq s$, from the ZIP instance, we define the negative example $(\vec{c_i}, \vec{1}_{1,2\ldots,n-b_i-1})$. Second, for $1 \leq i \leq n$ we define the negative example $(\vec{0}, \vec{0}_i)$. We claim that there is a solution to the ZIP instance iff there is a boolean threshold function consistent with the given examples. If our claim is true, then any learning algorithm can be used to decide the ZIP problem in random polynomial time by letting D^+ and D^- be uniform over the positive and negative examples respectively, and choosing $\epsilon = \frac{1}{s+n}$.

Suppose that Y is a set with characteristic vector $\vec{y} = (\vec{z}, \vec{w})$, and k is a positive integer such that the rule $Th_k(\vec{y})$ is consistent with the positive and negative examples defined above. (We must have $1 \leq k \leq 2n$.)

Then $k \leq \vec{p_1} \cdot \vec{y} = (\vec{0}, \vec{1}) \cdot \vec{y} \leq n$, thus $k \leq n$. Furthermore, by the positive and negative examples, for each i, $(\vec{0}, \vec{0}_i) \cdot (\vec{z}, \vec{w}) < k$ and $(\vec{0}, \vec{1}) \cdot (\vec{z}, \vec{w}) \geq k$, and it follows that (since $(\vec{0}, \vec{1})$ differs from $(\vec{0}, \vec{0}_i)$ only in the position $n + i$,) $\vec{w} = \vec{1}$. Thus $\vec{y} = (\vec{z}, \vec{1})$. Since $(\vec{0}, \vec{0}_i)$ is a negative example, $(\vec{0}, \vec{0}_i) \cdot (\vec{z}, \vec{1}) = n - 1 < k$, therefore, $k = n$.

We then have that $n \leq \vec{p_2} \cdot \vec{y} = (\vec{a}, \vec{1}_{1,2,\ldots,n-B}) \cdot (\vec{z}, \vec{1})$, so $\vec{a} \cdot \vec{z} \geq B$. Also,for each i, since $(\vec{c_i}, \vec{1}_{1,2\ldots,n-b_i-1})$ is a negative example, $(\vec{c_i}, \vec{1}_{1,2\ldots,n-b_i-1}) \cdot (\vec{z}, \vec{1}) \leq n-1$, and therefore $\vec{c_i} \cdot \vec{z} \leq b_i$.

Thus $\vec{z}$ is a solution to the ZIP instance. Similarly, any ZIP solution $\vec{z}$ gives rise in an obvious manner to a boolean threshold function consistent with the given examples. □

In cases where learning a class of representations is thought to be intractable, or when no learning algorithm exists, we may wonder whether we can learn *heuristic* rules for the concept; a rule which accounts for some significant fraction of the positive examples, while avoiding incorrectly classifying most of the negative examples. For example, since there is a learning algorithm for monomials, but not for k-term-DNF, and none known for general DNF, perhaps we can find a single monomial which covers half of the positive examples while avoiding error on all but $1-\epsilon$ of the negative examples whenever such a monomial exists. In the full paper, we formalize the notion of $h(n)$-heuristic learnability, so that a class is $h(n)$-heuristically learnable iff a learning algorithm can produce a hypothesis which correctly classifies at least the fraction $h(n)$ of the positive examples, while correctly classifying $1-\epsilon$ of the negative examples.

Theorem 13 *For any c, $0 < c < 1$, DNF is not c-heuristically learnable by monomials.*

Proof: See [8].

Theorem 14 *μ-formulae (boolean trees) of n variables are not $e^{-n^{1/3}}$-heuristically learnable, if the tree or formula produced must avoid misclassifying all negative examples.*

Proof: See [8].

Theorem 13 shows that the heuristic of covering as large a fraction of the positive examples as possible with a single monomial while learning DNF is not feasible.

Theorem 14 is a very strong result. It shows that even if there is a μ-formula correctly classifying all of the positive and negative examples, it is NP-hard to find one which correctly classifies an exponentially vanishing fraction of the positive examples, if we require that it avoid misclassifying any negative example.

While several classes of formulae are learnable from examples of only one kind [10], the question of whether both positive and negative examples are ever required was unsolved. The following result, which is independent of any complexity-theoretic hypothesis, answers this in the affirmative for the classes mentioned in Corollaries 3 and 4.

Theorem 15 *If A is the class of monotone monomials and B is any class of representations, then A is not learnable by B from negative examples only.*

Proof: Suppose for contradiction that M were a learning algorithm requiring only negative examples, and let n^c be the number of samples required by M for learning monomials over n variables, with $\epsilon = \frac{1}{n}$. Let a monomial be *monotone dense* if it is monotone and contains at least $\frac{n}{2}$ of the variables. Let S be a set of exactly n^c vectors from $\{0,1\}^n$, and let Ψ be the set of all such S's. If m is a monotone dense monomial, then $p_m \in \{0,1\}^n$ is the unique vector satisfying m with the fewest 1's. Finally, we say that $S \in \Psi$ is *legal negative* for m if S contains no vector satisfying m.

We first define distributions over the examples of monotone dense monomials in such a way that if m is being learned, any accurate hypothesis output by M must satisfy p_m. Thus, let $D^+(p_m) = 1$, and D^+ is 0 elsewhere. Let D^- be uniform over all $\vec{v}$ such that $m(\vec{v}) = 0$. We now define the predicate $P(S, m)$ to be 1 if and only if S is legal negative for m and when S is given to M, M outputs a hypothesis h such that $h(p_m) = 1$ (Note that this definition assumes that M is deterministic. To allow for probabilistic algorithms, we simply change the definition to $P(S, m) = 1$ if and only if S is legal negative for m, and when S is given to M, M outputs a hypothesis h such that $h(p_m) = 1$ with probability at least $\frac{1}{2}$, where the probability is taken over the coin tosses of M).

Suppose we draw $\vec{v}$ uniformly at random from $\{0,1\}^n$. Then

$$\Pr(m(\vec{v}) = 1) \leq 2^{\frac{n}{2}} \times \frac{1}{2^n} = \frac{1}{2^{\frac{n}{2}}}$$

since at most $2^{\frac{n}{2}}$ vectors can satisfy a monotone dense monomial. Thus, if we draw n^c points uniformly at random from $\{0,1\}^n$, the probability that we draw *some* point satisfying m is at most $\frac{n^c}{2^{\frac{n}{2}}}$. By this probability analysis, we conclude that the number of samples of size n^c that are legal negative for m must be at least $\frac{|\Psi|}{2}$. Since M is a learning algorithm, at least $\frac{|\Psi|}{2}(1-\epsilon)$

of these must satisfy $P(S,m) = 1$. Summing over all monotone dense monomials, we obtain

$$\frac{|\Psi|}{2}(1-\epsilon)\frac{2^n}{2} \leq \sum_{S\in\Psi}\left(\sum_{m \text{ is monotone dense}} P(S,m)\right)$$

For $S \in \Psi$, define $N(S)$ to be the number of monotone dense monomials satisfying $P(S,m) = 1$. Then the above can be rewritten as

$$\frac{|\Psi|}{2}(1-\epsilon)\frac{2^n}{2} \leq \sum_{S\in\Psi} N(S)$$

From this inequality, and the fact that $N(S)$ is always at most $\frac{2^n}{2}$, we conclude that at least $\frac{|\Psi|}{8}$ of the S must satisfy $N(S) \geq \frac{1}{4}(1-\epsilon)\frac{2^n}{2}$. Since D^- is uniform, for any particular monotone dense m being learned by M, we have probability at least $\frac{1}{8}$ of drawing such an S. But then the hypothesis output by M is satisfied by at least $\frac{1}{4}(1-\epsilon)\frac{2^n}{2}$ vectors, and m is satisfied by at most $2^{\frac{n}{2}}$ vectors. Our error on D^- is then at least

$$\frac{\frac{1}{4}(1-\epsilon)\frac{2^n}{2} - 2^{\frac{n}{2}}}{2^n}$$

and this error must be less than ϵ. But this cannot be true for $\epsilon \leq \frac{1}{10}$. Thus, M cannot achieve arbitrarily small error on monotone dense monomials, and the theorem follows. □

We note that there are results similar to Theorem 15 with easier proofs. For instance, combining the fact that monomials are learnable from positive examples only with Theorems 1 and 9, it is easy to show that monomials are not learnable from negative examples only unless NP = RP. There is also a simpler proof for Theorem 15 when the class A is relaxed to include all monomials; one could then apply the substitution techniques of Theorems 5 and 6 to show that monotone monomials are not learnable from negative examples only by any class of representations that is closed under the appropriate substitutions. However, the result of Theorem 15 is stronger, as we have shown that monotone monomials are not learnable from negative examples only by *any* representation, independent of any complexity-theoretic hypothesis (such as NP $\neq$ RP).

6 Distribution-Specific Positive Results

In typical computational domains, such as graph algorithms, it is found that NP-hard problems are intractable in practice, except in situations where the inputs can be identified as belonging to a restricted case of the domain. A problem may become tractable, for example, if the inputs are drawn from certain distributions. We suspect that learnability conforms to this pattern. In this section we shall show that learning is in fact tractable in certain cases when D^+ and D^- are both uniform distributions. Since the learnability of DNF is perhaps the most tantalizing open problem, and since general DNF is no harder than μDNF in the distribution-free case by Corollary 8, the following result is of some interest.

Theorem 16 *If D^+ and D^- are uniform over the positive and negative examples, respectively, of the formula being learned, then μDNF (DNF in which each variable occurs at most once) is learnable.*

Proof: Let n be the number of variables in the μDNF formula, F, to be learned. For some fixed integer d, let $\frac{1}{n^d} = \epsilon$ be the error parameter. We say that a monomial in F is *significant* if the probability that a random positive example satisfies this monomial is at least $\frac{1}{n^{d+1}}$.

We first give an outline of our algorithm and then show how each step can be implemented.

Learning Algorithm :

Step 1. Assume that every monomial in F is of size larger than $Clogn$ for $C = (d+2)^2$. This step will learn F using only positive examples if this assumption is true. If this assumption is not true, then we will discover this in step 2, and learn correctly in step 3 (from the negative examples). The substeps of step 1 are as follows :

1.1. For each i, decide whether x_i is in one of the monomials of F.

1.2. For each i, j such that x_i and x_j are in some monomials of F, decide whether they are in the same monomial.

1.3. Form the DNF μ-formula in the obvious way.

Step 2. Test whether the DNF μ-formula learned in step 1 is correct by trying it on a polynomial number of new examples. If it is correct, stop and output the formula, otherwise (the assumption of step 1 is not correct) go to step 3.

Step 3. Since some monomial in F is shorter than $Clogn$, we can assume that all the monomials are shorter than C^3logn and discard the long ones (which are not significant). We use only negative examples in this step. The substeps are :

3.1. For each i, decide whether x_i is in some (short) monomial.

3.2. For each i, j such that x_i and x_j are known to be in some monomials by step 3.1, decide if they belong to the same monomial.

3.3. Form the μDNF formula accordingly. (End of Algorithm)

Now we show how each step is implemented and prove its correctness and analyze its time complexity along the way.

In step 1, we draw only positive examples. Since there are at most n (disjoint) monomials in F, and we assumed that the size of each monomial is at least $Clogn$, the probability that a positive example satisfies 2 monomials of F is at most $\frac{1}{n^{C-2}}$. Therefore, in the following analysis, we consider only the positive examples which satisfy precisely one monomial each, without loss of generality.

Analysis of Substep 1.1 : For each i, if x_i is not in any monomials of F then in any randomly drawn positive example,

$\Pr(x_i = 0) = \Pr(x_i = 1) = \frac{1}{2}$.

If x_i is in a significant monomial in F then

(1) $\Pr(x_i = 1) \geq \frac{1}{2} + \frac{1}{2n^{d+1}} - O(n^{-C+1})$.

(Notice that if x_i is in an insignificant monomial then we simply think that x_i is in no monomial.) Using Facts 1 and 2, we can determine if x_i appears in a significant monomial by drawing a polynomial number of examples.

Analysis of Substep 1.2 : For each pair x_i and x_j that are in some monomials (as decided in substep 1.1), we now decide whether they are in the same monomial. Consider $x_i \vee x_j$.

Claim 1. If x_i and x_j are in the same monomial, then for a random positive example

(2) $\Pr(x_i \vee x_j = 1)$
$= \frac{3}{4} + \frac{1}{2}(\Pr(x_i = 1) - \frac{1}{2}) \pm o(n^{-C+2})$.

Proof. Since x_i and x_j appear in the same monomial, $\Pr(x_i = 1) = \Pr(x_j = 1)$. Let A be the event where the monomial containing x_i and x_j is satisfied in a randomly drawn positive example, and B be the event of some other monomial(s) being satisfied. Using the facts that $\Pr(A \cap B) = o(n^{-C+2})$, $\Pr(x_i = 1) = \Pr(A) + \frac{1}{2}(1 - \Pr(A)) \pm o(n^{-C+2})$, and $\Pr(A) + \Pr(B) = 1 + o(n^{-C+2})$, we have

$$
\begin{aligned}
&\Pr(x_i \vee x_j = 1)\\
&= \Pr(x_i \vee x_j = 1|A)\Pr(A)+\\
&\qquad \Pr(x_i \vee x_j = 1|B)\Pr(B) - o(n^{-C+2})\\
&= \Pr(A) + \tfrac{3}{4}(1 - \Pr(A)) \pm o(n^{-C+2})\\
&= 2(\Pr(x_i = 1) - \tfrac{1}{2})+\\
&\qquad \tfrac{3}{4}(1 - 2(\Pr(x_i = 1) - \tfrac{1}{2})) \pm o(n^{-C+2})\\
&= \tfrac{3}{4} + \tfrac{1}{2}(\Pr(x_i = 1) - \tfrac{1}{2}) \pm o(n^{-C+2}).
\end{aligned}
$$

□

Claim 2. If x_i and x_j appear in different monomials, then in a random positive example

(3) $\Pr(x_i \vee x_j = 1) = \frac{3}{4} + \frac{1}{2}(\Pr(x_i = 1) - \frac{1}{2})+$
$\frac{1}{2}(\Pr(x_j = 1) - \frac{1}{2}) \pm o(n^{-C+2})$

Proof. Let A be the event where the monomial containing x_i is satisfied, B be the similar event for x_j, and D be the event where some monomial other than the above two is satisfied. Similar to Claim 1, we have

$$
\begin{aligned}
&\Pr(x_i \vee x_j = 1)\\
&= \Pr(x_i \vee x_j = 1|A)\Pr(A)+\\
&\qquad \Pr(x_i \vee x_j = 1|B)\Pr(B)+\\
&\qquad \Pr(x_i \vee x_j = 1|D)\Pr(D) - o(n^{-C+2})\\
&= \Pr(A) + \Pr(B) + \tfrac{3}{4}\Pr(D)\\
&\qquad -o(n^{-C+2})\\
&= \Pr(A) + \Pr(B)+\\
&\qquad \tfrac{3}{4}(1 - \Pr(A) - \Pr(B)) \pm o(n^{-C+2})\\
&= \tfrac{3}{4} + \tfrac{1}{4}(\Pr(A) + \Pr(B)) \pm o(n^{-C+2})\\
&= \tfrac{3}{4} + \tfrac{1}{2}(\Pr(x_i = 1) - \tfrac{1}{2})\\
&\qquad +\tfrac{1}{2}(\Pr(x_j = 1) - \tfrac{1}{2}) \pm o(n^{-C+2})
\end{aligned}
$$

□

Combining formulas (1),(2),(3) and the fact that if x_i and x_j are in the same monomial, then $\Pr(x_i = 1) = \Pr(x_j = 1)$, we can determine whether x_i and x_j are in the same monomial by drawing a polynomial number of examples, and using Facts 1 and 2.

In step 2, we draw a polynomial number of examples to test if our learned formula is correct, again using Facts 1 and 2. Now if in step 2, it is decided that the formula learned in step 1 is not correct, then the assumption made in the beginning of step 1 is not correct. So there is a monomial in F which is of size at most $Clogn$. This implies that all the formulas of size larger than $C^3 logn$ are not significant. Therefore in step 3 we assume that all the monomials in F are shorter than $C^3 logn$. We use only the negative examples.

Analysis of Substep 3.1 : If x_i is not in any monomials of F, then in a randomly drawn negative example, we have

(4) $\Pr(x_i = 0) = \frac{1}{2}$.

Claim 3. If x_i is in a significant monomial of F, then

$\Pr(x_i = 0) \geq \frac{1}{2} + \frac{1}{2(n^{C^3} - 1)}$.

Proof. Assume s is the size of the monomial x_i is in. Then

$\Pr(x_i = 0) = \frac{2^s/2}{2^s - 1} = \frac{1}{2} + \frac{1}{2(2^s - 1)}$.

Since $s \leq C^3 logn$, the claim is true. □

By (4) and Claim 3, we can draw a polynomial number of examples, and decide if x_i is in some monomial of F, using Facts 1 and 2.

Analysis of Substep 3.2 : We have to decide whether x_i and x_j are in the same monomial, given that x_i, x_j are in some monomial(s).

Claim 4. If x_i, x_j are not in the same monomial, then

$\Pr(x_i = x_j = 0) = \Pr(x_i = 0)\Pr(x_j = 0)$.

Proof. If x_i, x_j are not in the same monomial, then they are independent of each other. □

Claim 5. If x_i, x_j are in the same monomial, then $\Pr(x_i = x_j = 0) = \frac{1}{2}\Pr(x_i = 0)$.

Proof.

$\Pr(x_i = x_j = 0) = \Pr(x_i = 0) \times \Pr(x_j = 0 | x_i = 0)$

But $\Pr(x_j = 0 | x_i = 0) = \frac{1}{2}$. □

Putting Claims 3,4,5 together, we can test if x_i, x_j are in the same monomial using Facts 1 and 2. This completes the proof of the theorem. □

It follows from Theorem 9 and Corollary 8 that k-term μDNF (μDNF in which there are at most k disjuncts) is not learnable. However, the algorithm given in the proof of Theorem 16 always outputs a hypothesis which has at most the same number of terms as the formula being learned - thus, it is also an algorithm for learning k-term μDNF under uniform distributions. This is the first example of a class for which distribution-free learning is NP-hard, but learning under a uniform distribution is tractable.

For monotone DNF under the uniform distribution we prove a similar result for the weaker model of "group learning", which appears to be of independent interest. Informally, we say that class A is *group learnable* if there is a learning algorithm L that runs in polynomial time, and outputs a deterministic hypothesis program H with the following property : for some polynomial $p(n)$, when H is given $p(n)$ examples that are either all positive or all negative (a group), H determines whether the group is positive or negative with high probability ($> 1 - \epsilon$).

Theorem 17 *If D^+ and D^- are uniform over the positive and negative examples, respectively, of the formula being learned, then monotone DNF is group learnable.*

Proof: (Outline) We state four technical claims without proofs. We then sketch the learning algorithm, which follows easily from the claims.

Let $F = m_1 + m_2 + \ldots + m_{q(n)}$ be a monotone DNF formula of n variables, where $q(n)$ is some polynomial. For $\vec{v}_P$ a positive example of F, $\vec{v}_N$ a negative example of F, and $\vec{x} \in \{0,1\}^n$, we define d_P (d_N, respectively) to be the Hamming distance between $\vec{x}$ and $\vec{v}_P$ ($\vec{v}_N$, respectively). Without loss of generality, we assume that m_1 is the shortest monomial in F. Let C be a constant such that $n^C > q^3(n)$ for large n.

The first claim says that if the shortest monomial in F is long, then two positive vectors (i.e., drawn from the uniform D^+) are more likely (with an inverse polynomial advantage) to be closer (in Hamming distance) than are one positive and one negative vector.

Claim 1. Let $|m_1| > C\log(n)$. Let $\vec{v}_P$ and $\vec{x}$ be drawn from the uniform distribution D^+, and let $\vec{v}_N$ be drawn from the uniform distribution D^-. Then

$$(1) \quad \Pr(d_P < d_N) \geq \Pr(d_P > d_N) + \frac{1}{p(n)}$$

for some polynomial $p(n)$.

The second claim says that if the shortest monomial is short, two *negative* vectors are more likely to be closer to each other (with an inverse polynomial advantage) than are one positive and one negative vector.

Claim 2. Let $|m_1| \leq C\log(n)$. Let $\vec{v}_P$ be drawn randomly from the uniform distribution D^+, and let $\vec{v}_N$ and $\vec{x}$ be drawn randomly from the uniform distribution D^-. Then

$$(2) \quad \Pr(d_N < d_P) \geq \Pr(d_N > d_P) + \frac{1}{p(n)}$$

for some polynomial $p(n)$.

We also use two easier claims that are true independent of the size of the smallest monomial.

Claim 3. Let $\vec{v}_P$ and $\vec{x}$ be drawn from D^+, and let $\vec{v}_N$ be drawn from D^-. Then

$$(3) \quad \Pr(d_P < d_N) \geq \Pr(d_P > d_N)$$

Claim 4. Let $\vec{v}_P$ be drawn randomly from D^+, and let $\vec{v}_N$ and $\vec{x}$ be drawn from D^-. Then

$$(4) \quad \Pr(d_N < d_P) \geq \Pr(d_N > d_P)$$

Suppose $G = \{\vec{x}_1, \ldots, \vec{x}_{n^k}\}$ is a set of vectors that are either all drawn from D^+ or all drawn from D^-. We use the above claims in the following algorithm to determine if G is a positive or negative group.

Learning Algorithm :

Step 1. Draw a polynomial number of pairs $(\vec{v}_P, \vec{v}_N)$, where $\vec{v}_P$ is drawn from D^+ and $\vec{v}_N$ is drawn from D^- (the exact number of pairs needed is determined by the use of Facts 1 and 2 in the proofs of the claims). Form a table of these pairs.

Step 2. Determine with high probability if (1) holds. This can be done by performing a polynomial number of experiments, drawing $\vec{v}_P, \vec{x}$ from D^+, drawing $\vec{v}_N$ from D^-, and using Facts 1 and 2. If (1) holds, then with high probability, $\vec{x}$ will be closer to $\vec{v}_P$ than it is to $\vec{v}_N$. We can notice this probabilistic advantage in polynomial time using Facts 1 and 2. If it is decided that (1) does *not* hold, then with high probability, the condition of Claim 1 must be false. Therefore, the condition of Claim 2 must hold, so (2) holds.

Step 3. Suppose we determined that (1) holds. To determine if G is positive or negative, for each i, compute the Hamming distance between $\vec{x}_i$ and the positive and negative vectors of the ith table entry. If $\vec{x}_i$

is closer to the positive vector, consider this a positive vote; if it is closer to the negative vector, consider this a negative vote. (1) says that if G is positive, the cumulative vote will be "noticeably" (i.e., in polynomial time, using Facts 1 and 2) positive; (3) says that if G is negative, the cumulative vote will be at least half negative. We can decide if G is positive or negative in a similar way if (2) holds. □

7 Acknowledgements

We would like to thank Umesh Vazirani for many helpful ideas and discussions on learning under uniform distributions.

References

[1] D. Angluin.
Learning Regular Sets From Queries and Counter-examples.
Technical Report, Yale University Computer Science Dept., TR-464, 1986.

[2] D. Angluin and P.D. Laird.
Identifying k-CNF Formulas From Noisy Examples.
Technical Report, Yale University Computer Science Dept., TR-478, 1986.

[3] D. Angluin and L.G. Valiant.
Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings. *JCSS, 18(2):155-193, 1979.*

[4] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth.
Classifying Learnable Geometric Concepts With the Vapnik-Chervonenkis Dimension.
In *Proceedings of the* 18th *Annual STOC*, pp 273-282. Assoc. Comp. Mach., New York, 1986.

[5] M. Garey and D. Johnson.
The Complexity of Near-optimal Graph Coloring.
J. ACM 23(1):43-49, 1976.

[6] M. Garey and D. Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness,
W. H. Freeman, San Francisco, 1979.

[7] D. Haussler.
Quantifying the Inductive Bias in Concept Learning.
Unpublished manuscript, November, 1986.

[8] L. Pitt and L.G. Valiant.
Computational Limitations on Learning From Examples.
Technical Report, Harvard University, TR-05-86, and submitted for publication.

[9] R. Rivest.
Learning Decision-Lists.
Unpublished manuscript, December, 1986.

[10] L. G. Valiant.
A Theory of the Learnable.
Comm. ACM, 27(11):1134-1142, 1984.

[11] L. G. Valiant.
Learning Disjunctions of Conjunctions.
In *Proceedings of the* 9th *IJCAI,* vol. 1, pp 560-566, Los Angeles, CA. August, 1985.

[12] L. G. Valiant.
Deductive Learning.
Phil. Trans. R. Soc. Lond. A 312, 441-446, 1984.

[13] A. Wigderson.
A New Approximate Graph Coloring Algorithm.
In *Proceedings of the* 14th *Annual STOC*, pp 325-329. Assoc. Comp. Mach., New York, 1982.