Learning Boolean Formulae*

Michael Kearns[†] M.I.T.

Ming Li[‡] University of Waterloo

Leslie Valiant[§] Harvard University

Abstract

Efficient distribution-free learning of Boolean formulae from positive and negative examples is considered. It is shown that classes of formulae that are efficiently learnable from only positive examples or only negative examples have certain closure properties. A new substitution technique is used to show that in the distribution-free case learning DNF (disjunctive normal form formulae) is no harder than learning monotone DNF. We prove that monomials cannot be efficiently learned from negative examples alone, even if the negative examples are uniformly distributed. It is also shown that if the examples are drawn from uniform distributions then the class of DNF in which each variable occurs at most once is efficiently learnable, while the class of all monotone Boolean functions is efficiently *weakly learnable* (i.e., individual examples are correctly classified with a probability larger than $\frac{1}{2} + \frac{1}{p}$, where p is a polynomial in the relevant parameters of the learning problem). We then show an equivalence between the notion of weak learning and the notion of *group learning*, where a group of examples of polynomial size, either all positive or all negative, must be correctly classified with high probability.

Key words: Machine Learning, Inductive Inference.

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,

*Earlier versions of most of the results presented here were described in "On the learnability of Boolean formulae", by M. Kearns, M. Li, L. Pitt and L. Valiant, Proceedings of the 19th A.C.M. Symposium on the Theory of Computing, 1987, pp. 285-295. Earlier versions of Theorems 15 and 22 were announced in "Cryptographic limitations on learning Boolean formulae and finite automata", by M. Kearns and L. Valiant, Proceedings of the 21st A.C.M. Symposium on the Theory of Computing, 1989, pp. 433-444.

[†]This research was done while the author was at Harvard University and supported by grants NSF-DCR-8600379, ONR-N00014-85-K-0445, and an A.T. & T. Bell Laboratories Scholarship. Author's current address: M.I.T. Laboratory for Computer Science, NE43-348, Cambridge, MA 02139.

[‡]This research was done while the author was at Harvard University and supported by grants NSF-DCR-8600379, DAAL03-86-K-0171, and ONR-N00014-85-K-0445. Author's current address: Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1 Canada.

[§]Supported by grants NSF-DCR-8600379 and ONR-N00014-85-K-0445. Author's address: Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138.

for Boolean expressions and 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 [V84]. A more complete discussion and justification of this model can be found in [V85, BEHW86, KLPV87b]. The paper [BEHW86] includes some discussion that is relevant more particularly to infinite representations, such as geometric ones, rather than the finite case of Boolean functions.

The results of this paper fall into four categories: tools for determining efficient learnability, a negative result, distribution-specific positive results, and an equivalence between two models of learnability.

The tools for determining efficient learnability are of two kinds. In Section 3.1 we discuss closure under Boolean operations on the members of the learnable classes. The assumption that the classes are learnable from positive examples only or from negative examples only is sometimes sufficient to ensure closure. In Section 3.2, we give a general substitution technique. It can be used to show, for example, that if disjunctive normal form (DNF) formulae are efficiently learnable in the monotone case, then they are also efficiently learnable in the unrestricted case.

In Section 4 we prove a negative result. We show that for purely information-theoretic reasons, monomials cannot be learned from negative examples alone, regardless of the hypothesis representation, and even if the distribution on the negative examples is uniform. This contrasts with the fact that monomials can be learned from positive examples alone [V84], and can be learned from very few examples if both kinds are available [H88, L88].

The class of DNF expressions in which each variable occurs at most once is called μDNF. In Section 5, we consider learning μDNF and the class of arbitrary monotone Boolean functions, under the restriction that both the positive and negative examples are drawn from uniform distributions. We show that μDNF is learnable (with μDNF as the hypothesis representation) in this distribution-specific case. In the distribution-free setting, learning μDNF with μDNF as the hypothesis representation is known to be NP-hard [PV88]. We also show that arbitrary monotone Boolean functions are *weakly learnable* under uniform distributions, with a polynomial-time program as the hypothesis representation. In weak learning, it is sufficient to deduce a rule by which individual examples are correctly classified with a probability $\frac{1}{2} + \frac{1}{p}$, where p is a polynomial in the parameters of the learning problem. We conclude in Section 6 by showing that weak learning is equivalent to the *group learning* model, in which it is sufficient to deduce a rule that determines with accuracy $1-\epsilon$ whether a large enough set of examples contains only positive or only negative examples, given that one of the two possibilities holds. This equivalence holds in both the distribution-free case and under any restricted class of distributions, thus yielding a group learning algorithm for monotone Boolean functions under uniform distributions. In the distribution-free case, monotone functions are not weakly learnable by any hypothesis representation, independent of any complexity-theoretic assumptions.

2 Definitions and Notation

In this section we give definitions for the model of machine learning we study. This model was first defined in [V84].

2.1 Representing subsets of a domain

Concept classes and their representation. Let X be a set called a *domain*. We may think of X as containing encodings of all objects of interest to us in our learning problem. The goal of a learning algorithm is then to infer some unknown subset of X, called a *concept*, chosen from a known *concept class*. We might imagine a child attempting to learn to distinguish chairs from non-chairs among the set X of all the physical objects in its environment. This particular concept is but one of many concepts in the class, each of which the child might be expected to learn and each of which is a set of related and somehow "interesting" objects. For example, another concept might consist of all metal objects in the environment. We would not expect a randomly chosen subset of objects to be an "interesting" concept, since as humans we do not expect these objects to bear any natural relation to one another.

For computational purposes we always need a way of *naming* or *representing* concepts. Thus, we formally define a *representation class over* X to be a pair (σ, C), where $C \subseteq \{0,1\}^*$ and σ is a mapping $\sigma : C \to 2^X$. For $c \in C$, $\sigma(c)$ is called a *concept* over X; the image space $\sigma(C)$ is the *concept class* that is *represented* by (σ, C). For $c \in C$, we define $pos(c) = \sigma(c)$ (the *positive examples* of c) and $neg(c) = X - \sigma(c)$ (the *negative examples* of c). The domain X and the mapping σ will usually be clear from the context, and we will simply refer to the *representation class* C. We will sometimes use the notation $c(x)$ to denote the value of the characteristic function of $\sigma(c)$ on the domain point x; thus $x \in pos(c)$ (respectively, $x \in neg(c)$) and $c(x) = 1$ ($c(x) = 0$) are used interchangeably. We assume that domain points $x \in X$ and representations $c \in C$ are efficiently encoded using any of the standard schemes (see e.g. [GJ79]), and denote by $|x|$ and $|c|$ the length of these encodings measured in bits.

Parameterized representation classes. We will often study *parameterized* classes of representations. Here we have a stratified domain $X = \cup_{n \geq 1} X_n$ and representation class $C = \cup_{n \geq 1} C_n$. The parameter n can be regarded as an appropriate measure of the complexity of concepts in $\sigma(C)$, and we assume that for a representation $c \in C_n$ we have $pos(c) \subseteq X_n$ and $neg(c) = X_n - pos(c)$. For example, X_n may be the set $\{0,1\}^n$, and C_n the class of all Boolean formulae over n variables. Then for $c \in C_n$, $\sigma(c)$ would contain all satisfying assignments of the formula c.

Efficient evaluation of representations. If C is a representation class over X, we say that C is *polynomially evaluatable* if there is a (probabilistic) polynomial-time *evaluation algorithm* A that on input a representation $c \in C$ and a domain point $x \in X$ outputs $c(x)$.

Samples. A *labeled example* from a domain X is a pair $< x, b >$, where $x \in X$ and $b \in \{0,1\}$. A *labeled sample* $S = < x_1, b_1 >, \ldots, < x_m, b_m >$ from X is a finite sequence of labeled examples from X. If C is a representation class, a *labeled example of* $c \in C$ is a labeled example of the form $< x, c(x) >$, where $x \in X$. A *labeled sample of* c is a labeled sample S where each example of S is a labeled example of c. In the case where all labels b_i or $c(x_i)$ are 1 (respectively, 0), we may omit the labels and simply write S as a list of points $x_1, \ldots, x_m$, and we call the sample a *positive* (*negative*) sample.

We say that a representation h and an example $< x, b >$ *agree* if $h(x) = b$; otherwise they *disagree*. We say that a representation h and a sample S are *consistent* if h agrees with each example in S.

2.2 Distribution-free learning

Distributions on examples. On any given execution, a learning algorithm for a representation class C will be receiving examples of a single distinguished representation $c \in C$. We call this distinguished c the *target representation*. Examples of the target representation are generated probabilistically as follows: let D_c^+ be a fixed but arbitrary probability distribution over $pos(c)$, and let D_c^- be a fixed but arbitrary probability distribution over $neg(c)$. We call these distributions the *target distributions*. When learning c, learning algorithms will be given access to two oracles, POS and NEG, that behave as follows: oracle POS (respectively, NEG) returns in unit time a positive (negative) example of the target representation, drawn randomly according to the target distribution D_c^+ (D_c^-).

We think of the target distributions as representing the "real world" distribution of the environment in which the learning algorithm must perform. For instance, suppose that the target concept were that of "dangerous situations". Certainly the situations "oncoming tractor" and "oncoming taxi" are both contained in this concept. However, a child growing up in a rural environment is much more likely to witness the former event than the latter, and the situation is reversed for a child growing up in an urban environment. These differences in probability are reflected in different target distributions for the same underlying target concept. Furthermore, since we rarely expect to have precise knowledge of the target distributions at the time we design a learning algorithm, ideally we seek algorithms that perform well under *any* target distributions.

Given a fixed target representation $c \in C$, and given fixed target distributions D_c^+ and D_c^-, there is a natural measure of the *error* (with respect to c, D_c^+ and D_c^-) of a representation h from a representation class H. We define $e_c^+(h) = D_c^+(neg(h))$ and $e_c^-(h) = D_c^-(pos(h))$. Note that $e_c^+(h)$ (respectively, $e_c^-(h)$) is simply the probability that a random positive (negative) example of c is identified as negative (positive) by h. If both $e_c^+(h) < \epsilon$ and $e_c^-(h) < \epsilon$, then we say that h is an *ϵ-good* hypothesis (with respect to D_c^+ and D_c^-); otherwise, h is *ϵ-bad*. We define the *accuracy* of h to be the value $min(1 - e_c^+(h), 1 - e_c^-(h))$.

When the target representation c is clear from the context, we will drop the subscript c and simply write D^+, D^-, e^+ and e^-.

In the definitions that follow, we will demand that a learning algorithm produce with high proability an ϵ-good hypothesis regardless of the target representation and target distributions. While at first this may seem like a strong criterion, note that the error of the hypothesis output is always measured with respect to the same target distributions on which the algorithm was trained. Thus, while it is true that certain examples of the target representation may be extremely unlikely to be generated in the training process, these same examples intuitively may be "ignored" by the hypothesis of the learning algorithm, since they contribute a negligible amount of error. Continuing our informal example, the rural child may never be shown an oncoming taxi as an example of a dangerous situation, but provided he remains in the environment in which he was trained, it is unlikely that his inability to recognize this danger will ever become apparent.

Learnability. Let C and H be representation classes over X. Then C is *learnable from examples by* H if there is a (probabilistic) algorithm A with access to POS and NEG, taking inputs ϵ, δ, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any input values $0 < \epsilon, \delta < 1$, algorithm A halts and outputs a representation $h_A \in H$ that with probability greater than $1 - \delta$ satisfies

$$(i)\ e^+(h_A) < \epsilon$$

and

$$(ii)\ e^-(h_A) < \epsilon.$$

We call C the *target class* and H the *hypothesis class*; the output $h_A \in H$ is called the *hypothesis* of A. A will be called a *learning algorithm* for C. If C and H are polynomially evaluatable, and A runs in time polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}$ and $|c|$ then we say that C is *polynomially learnable from examples by* H; if C is parameterized we also allow the running time of A to have polynomial dependence on n. We will drop the phrase "from examples" and simply say that C is *learnable by* H, and C is *polynomially learnable by* H. We say C is *polynomially learnable* to mean that C is polynomially learnable by H for some polynomially evaluatable H. We will sometimes call ϵ the *accuracy parameter* and δ the *confidence parameter*.

Thus, we ask that for any target representation and any target distributions, a learning algorithm finds an ϵ-good hypothesis with probability at least $1 - \delta$. A major goal of research in this model is to discover which representation classes C are polynomially learnable.

We will sometimes bound the probability that a learning algorithm fails to output an ϵ-good hypothesis by separately analyzing a constant number k of different ways the algorithm might fail, and showing that each of these occurs with probability at most $\frac{\delta}{k}$. Thus, we will use the expression "with high probability" to mean with probability at least $1 - \frac{\delta}{k}$ for an appropriately large constant k, which for brevity may sometimes be left unspecified.

We refer to this model as the *distribution-free* model, to emphasize that we seek algorithms that work for any target distributions. We also occasionally refer to this model as *strong learnability*, in contrast with the notion of weak learnability defined below.

Weak learnability. We will also consider a distribution-free model in which the hypothesis of the learning algorithm is required to perform only slightly better than random guessing.

Let C and H be representation classes over X. Then C is *weakly learnable from examples by* H if there is a polynomial p and a (probabilistic) algorithm A with access to POS and NEG, taking input δ, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any input value $0 < \delta < 1$, algorithm A halts and outputs a representation $h_A \in H$ that with probability greater than $1 - \delta$ satisfies

$$(i)\ e^+(h_A) < \tfrac{1}{2} - \tfrac{1}{p(|c|)}$$

and

$$(ii)\ e^-(h_A) < \frac{1}{2} - \frac{1}{p(|c|)}.$$

Thus, the accuracy of h_A must be at least $\frac{1}{2} + \frac{1}{p(|c|)}$. A will be called a *weak learning* algorithm for C. In the case that the target class C is parameterized, we allow the polynomial p in conditions (i) and (ii) to depend on n. If C and H are polynomially evaluatable, and A runs in time polynomial in $\frac{1}{\delta}$ and $|c|$ (as well as polynomial in n if C is parameterized), we say that C is *polynomially weakly learnable by* H and C is *polynomially weakly learnable* if it is weakly learnable by H for some polynomially evaluatable H.

While the motivation for weak learning may not be as apparent as that for strong learning, we may intuitively think of weak learning as the ability to detect some slight bias separating positive and negative examples. In fact, Schapire [S89] has shown that in the distribution-free setting, polynomial-time weak learning is equivalent to polynomial-time learning.

Positive-only and negative-only learning algorithms. We will sometimes study learning algorithms that need only positive examples or only negative examples. If A is a learning algorithm for a representation class C, and A makes no calls to the oracle NEG (respectively, POS), then we say that A is a *positive-only* (*negative-only*) learning algorithm, and C is *learnable from positive examples* (*learnable from negative examples*). Analogous definitions are made for weak learnability. Note that although the learning algorithm may receive only one type of example, the hypothesis output must still be accurate with respect to *both* the positive and negative distributions.

Many learning algorithms in the distribution-free model are positive-only or negative-only. The study of positive-only and negative-only learning is important for at least two reasons. First, it helps to quantify more precisely what kind of information is required for learning various representation classes. Second, it is crucial for applications where, for instance, positive examples are rare but must be classified accurately when they do occur.

Distribution-specific learnability. The models for learnability described above demand that a learning algorithm work regardless of the distributions on the examples. We will sometimes relax this condition, and consider these models under restricted classes of target distributions, for instance the class consisting only of the uniform distribution. Here the definitions are the same as before, except that we ask that the performance criteria for learnability be met only under these restricted target distributions.

Sample complexity. Let A be a learning algorithm for a representation class C. Then we denote by $s_A(\epsilon, \delta)$ the number of calls to the oracles POS and NEG made by A on inputs ϵ, δ; this is a worst-case measure over all possible target representations in C and all target distributions D^+ and D^-. In the case that C is a parameterized representation class, we also allow s_A to depend on n. We call the function s_A the *sample complexity* or *sample size* of A. We denote by s_A^+ and s_A^- the number of calls of A to POS and NEG, respectively. Note that here we have defined the sample complexity deterministically, since all of the algorithms we give use a number of examples depending only on ϵ, δ and n. In general, however, we may wish to allow the number of examples drawn to depend on the coin flips of the learning algorithm and the actual sequence of examples seen; in this case the sample

complexity would be an expected value rather than an absolute value. Following the sample complexity lower bound of Theorem 12 in Section 4, we discuss how to adapt the proof of this theorem and other known deterministic sample complexity lower bounds to yield lower bounds on the expected number of examples.

Chernoff bounds. We shall make extensive use of the following bounds on the area under the tails of the binomial distribution. 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 independent 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 $0 \leq \alpha \leq 1$,

$$\textbf{Fact CB1.}\ LE(p, m, (1-\alpha)mp) \leq e^{-\alpha^2 mp/2}$$

and

$$\textbf{Fact CB2.}\ GE(p, m, (1+\alpha)mp) \leq e^{-\alpha^2 mp/3}$$

These bounds in the form they are stated are from [AV79]; see also [C52]. Although we will make frequent use of Fact CB1 and Fact CB2, we will do so in varying levels of detail, depending on the complexity of the calculation involved. However, we are primarily interested in the Chernoff bounds for the following consequence of Fact CB1 and Fact CB2: given an event E of probability p, we can obtain an estimate p' of p by drawing m points from the distribution and computing the frequency p' with which E occurs in this sample. Then for m polynomial in $\frac{1}{p}$ and $\frac{1}{\alpha}$, say $m = O(\frac{1}{p} log \frac{1}{\delta})$, p' satisfies $\frac{p}{2} < p' < 2p$ with probability at least $1 - \delta$. If we also allow m to depend polynomially on $\frac{1}{\beta}$, say $m = O(\frac{1}{\beta^2} log \frac{1}{\delta})$, we can obtain an estimate p' such that $p - \beta < p' < p + \beta$ with probability at least $1 - \delta$.

Notational conventions. Let $E(x)$ be an event and $\psi(x)$ a random variable that depend on a parameter x that takes on values in a set X. Then for $X' \subseteq X$, we denote by $\mathbf{Pr}_{x \in X'}(E(x))$ the probability that E occurs when x is drawn uniformly at random from X'. Similarly, $\mathbf{E}_{x \in X'}(\psi(x))$ is the expected value of ψ when x is drawn uniformly at random from X'. We also need to work with distributions other than the uniform distribution; thus if P is a distribution over X we use $\mathbf{Pr}_{x \in P}(E(x))$ and $\mathbf{E}_{x \in P}(\psi(x))$ to denote the probability of E and the expected value of ψ, respectively, when x is drawn according to the distribution P. When E or ψ depend on several parameters that are drawn independently from different distributions we use multiple subscripts. For example, $\mathbf{Pr}_{x_1 \in P_1, x_2 \in P_2, x_3 \in P_3}(E(x_1, x_2, x_3))$ denotes the probability of event E when x_1 is drawn from distribution P_1, x_2 from P_2, and x_3 from P_3, independently.

2.3 Some representation classes

We now define some of the restricted Boolean formula representation classes whose learnability we will study. Here the domain X_n is always $\{0,1\}^n$ and the mapping σ simply maps each formula to its set of satisfying assignments. The classes defined below are all parameterized; for each class we define the subclasses C_n, and then C is defined by $C = \cup_{n \geq 1} C_n$.

Monomials: The representation class M_n consists of all conjunctions of literals (monomials) over the Boolean variables $x_1, \ldots, x_n$.

kCNF: For constant k, the representation class $k\,CNF_n$ consists of Boolean formulae of the form $C_1 \wedge \cdots \wedge C_l$, where each C_i is a disjunction of at most k literals over the Boolean variables $x_1, \ldots, x_n$. No a priori bound is placed on l. Note that $M_n = 1CNF_n$.

kDNF: For constant k, the representation class $kDNF_n$ consists of Boolean formulae of the form $T_1 \vee \cdots \vee T_l$, where each T_i is a conjunction of at most k literals over the Boolean variables $x_1, \ldots, x_n$. No a priori bound is placed on l.

k-clause CNF: For constant k, the representation class $k - clause\ CNF_n$ consists of all conjunctions of the form $C_1 \wedge \cdots \wedge C_l$ for $l \leq k$, where each C_i is a disjunction of literals over the Boolean variables $x_1, \ldots, x_n$.

k-term DNF: For constant k, the representation class $k - term\ DNF_n$ consists of all disjunctions of the form $T_1 \vee \cdots \vee T_l$ for $l \leq k$, where each T_i is a monomial over the Boolean variables $x_1, \ldots, x_n$.

CNF: The representation class CNF_n consists of all formulae of the form $C_1 \wedge \cdots \wedge C_l$, where each C_i is a disjunction of literals over the Boolean variables $x_1, \ldots, x_n$. No a priori bound is placed on l.

DNF: The representation class DNF_n consists of all formulae of the form $T_1 \vee \cdots \vee T_l$, where each T_i is a disjunction of literals over the Boolean variables $x_1, \ldots, x_n$. No a priori bound is placed on l.

3 Useful Tools for Distribution-free Learning

In this section we describe some general tools for determining polynomial-time learnability in the distribution-free model. These tools fall into two classes: closure theorems for polynomially learnable representation classes, and reductions between learning problems via variable substitution. We also discuss some applications of these techniques. Although we are primarily interested here in polynomial-time learnability, the results presented in this section are easily generalized to algorithms with higher time complexities.

3.1 Closure theorems for polynomially learnable classes: composing learning algorithms

Suppose that C_1 is polynomially learnable by H_1, and C_2 is polynomially learnable by H_2. Then it is easy to see that the class $C_1 \cup C_2$ is polynomially learnable by $H_1 \cup H_2$: we first assume that the target representation c is in the class C_1 and run algorithm A_1 for learning C_1. We then test the hypothesis h_1 output by A_1 on a polynomial-size random sample of c to determine with high probability if it is ϵ-good (this can be done using Fact CB1 and Fact CB2). If h_1 is ϵ-good, we halt; otherwise, we run algorithm A_2 for learning C_2 and use the hypothesis h_2 output by A_2. This algorithm demonstrates one way in which existing learning algorithms can be composed to learn more powerful representation classes, and it generalizes to any polynomial number of unions of polynomially learnable classes. Are there more interesting ways to compose learning algorithms?

In this section we describe techniques for composing existing learning algorithms to obtain new learning algorithms for representation classes that are formed by performing logical operations on the *members* of the learnable classes. We apply the results to obtain polynomial-time learning algorithms for two classes of Boolean formulae not previously known to be polynomially learnable. Recently in [HSW88] a general composition technique has been proposed and carefully analyzed in several models of learnability.

If $c_1 \in C_1$ and $c_2 \in C_2$ are representations, the representation $c_1 \vee c_2$ is defined by $pos(c_1 \vee c_2) = pos(c_1) \cup pos(c_2)$. Similarly, $pos(c_1 \wedge c_2) = pos(c_1) \cap pos(c_2)$. We then define $C_1 \vee C_2 = \{c_1 \vee c_2 : c_1 \in C_1, c_2 \in C_2\}$ and $C_1 \wedge C_2 = \{c_1 \wedge c_2 : c_1 \in C_1, c_2 \in C_2\}$.

Theorem 1 *Let C_1 be polynomially learnable by H_1, and let C_2 be polynomially learnable by H_2 from negative examples. Then $C_1 \vee C_2$ is polynomially learnable by $(H_1 \vee H_2) \cup H_2$.*

Proof: Let A_1 be a polynomial-time algorithm for learning C_1 by H_1, and A_2 a polynomial-time negative-only algorithm for learning C_2 by H_2. We describe a polynomial-time algorithm A for learning $C_1 \vee C_2$ by $(H_1 \vee H_2) \cup H_2$ that uses A_1 and A_2 as subroutines. We first give a high level description of our algorithm, and then provide the quantitative sample bounds and formal arguements.

Learning Algorithm A.

Step 1 Simulate algorithm A_2. Draw negative examples from D^-. Formulate a concept $h_2 \in H_2$ for c_2. Now h_2 does not make too much errors on D^-, but likely h_2 makes errors on D^+.

Step 2 Next A determines if $e^+(h_2) < \epsilon$ (by sampling and using Chernoff bounds). If $e^+(h_2) < \epsilon$, then $h_A := h_2$ and quit.

Step 3 If $e^+(h_2) > \epsilon$, A simulates A_1, drawing samples from both D^+ and D^- to produce h_1. Output $h_A := h_1 \vee h_2$.

Let $c = c_1 \vee c_2$ be the target representation in $C_1 \vee C_2$, where $c_1 \in C_1$ and $c_2 \in C_2$, and let D^+ and D^- be the target distributions on $pos(c)$ and $neg(c)$, respectively. Let s_{A_1}, a function depending ϵ, δ, and $|h_1|$, be the number of examples needed by algorithm A_1.

Since $neg(c) \subseteq neg(c_2)$, the distribution D^- may be regarded as a distribution on $neg(c_2)$, with $D^-(x) = 0$ for $x \in neg(c_2) - neg(c)$. Thus A first runs the negative-only algorithm A_2 to obtain a representation $h_2 \in H_2$ for c_2, using the examples generated from D^- by NEG. This simulation is done with accuracy parameter $\frac{\epsilon}{k s_{A_1}}$ and confidence parameter, say $\frac{\delta}{5}$, where k is a constant that can be determined by applying Fact CB1 and Fact CB2 in the analysis below. A_2 then outputs an $h_2 \in H_2$ satisfying with high probability $e^-(h_2) < \frac{\epsilon}{k s_{A_1}}$. Note that although we are unable to bound $e^+(h_2)$ directly (because D^+ is not a distribution over $pos(c_2)$), the fact that the simulation of the negative-only algorithm A_2 must work for any target distribution on $pos(c_2)$ implies that h_2 must satisfy with high probability

$$\mathbf{Pr}_{x \in D^+}(x \in neg(h_2) \text{ and } x \in pos(c_2)) \leq \mathbf{Pr}_{x \in D^+}(x \in neg(h_2) | x \in pos(c_2)) < \frac{\epsilon}{k s_{A_1}}. \quad (1)$$

A next attempts to determine if $e^+(h_2) < \epsilon$. A takes $m = O(\frac{1}{\epsilon} ln \frac{1}{\delta})$ examples from POS and uses these examples to compute an estimate p for the value of $e^+(h_2)$. Using Fact CB1 it can be shown that if

$e^+(h_2) \geq \epsilon$, then with high probability $p > \frac{\epsilon}{2}$. Using Fact CB2 it can be shown that if $e^+(h_2) \leq \frac{\epsilon}{4}$, then with high probability $p \leq \frac{\epsilon}{2}$. Thus, if $p \leq \frac{\epsilon}{2}$ then A guesses that $e^+(h_2) \leq \epsilon$. In this case A halts with $h_A = h_2$ as the hypothesis.

On the other hand, if $p > \frac{\epsilon}{2}$ then A guesses that $e^+(h_2) \geq \frac{\epsilon}{4}$. In this case A runs A_1 in order to obtain an h_1 that is ϵ-good with respect to D^- and also with respect to that portion of D^+ on which h_2 is wrong. More specifically, A runs A_1 with accuracy parameter $\frac{\epsilon}{k}$ and confidence parameter, say, $\frac{\delta}{5}$ according to the following distributions: each time A_1 calls NEG, A supplies A_1 with a negative example of c drawn according to the target distribution D^-; each such example is also a negative example of c_1 since $neg(c) \subseteq neg(c_1)$. Each time A_1 calls POS, A draws from the target distribution D^+ until a point $x \in neg(h_2)$ is obtained. Since the probability of drawing such an x is exactly $e^+(h_2)$, if $e^+(h_2) \geq \frac{\epsilon}{4}$ then the time needed to obtain with high probability a sample of size s_{A_1} of such x is polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ and s_{A_1} by Fact CB1. Now

$$\mathbf{Pr}_{x \in D^+}(x \in neg(h_2) \text{ and } x \in neg(c_1)) \leq \mathbf{Pr}_{x \in D^+}(x \in neg(h_2) \text{ and } x \in pos(c_2)) < \frac{\epsilon}{k s_{A_1}} \tag{2}$$

by Equation 1 and the fact that for $x \in pos(c)$, $x \in neg(c_1)$ implies $x \in pos(c_2)$. Since A_1 needs at most s_{A_1} positive examples and h_2 satisfies Equation 2, with high probability all of the positive examples x given to A_1 in this simulation satisfy $x \in pos(c_1)$ for k a large enough constant. Following this simulation, A_1 with high probability outputs h_1 satisfying $e^-(h_1) < \frac{\epsilon}{k}$ and also

$$\mathbf{Pr}_{x \in D^+}(x \in neg(h_1) \text{ and } x \in neg(h_2)) < \mathbf{Pr}_{x \in D^+}(x \in neg(h_1) | x \in neg(h_2)) < \frac{\epsilon}{k}. \tag{3}$$

Setting $h_A = h_1 \vee h_2$, we have $e^+(h_A) < \epsilon$ by Equation 3 and $e^-(h_A) < \epsilon$, as desired. Note that the time required by this simulation is polynomial in the time required by A_1 and the time required by A_2. □

The time complexity for algorithm A in Theorem 1 is roughly $O(t_2 + m + \frac{1}{\epsilon} t_1)$ where t_1 and t_2 are the time complexities of A_1 and A_2, respectively, and $m = O(\frac{1}{\epsilon} log \frac{1}{\delta})$ is the number of samples needed in Step 2 of the algorithm A. The following dual to Theorem 1 has a similar proof:

Theorem 2 *Let C_1 be polynomially learnable by H_1, and let C_2 be polynomially learnable by H_2 from positive examples. Then $C_1 \wedge C_2$ is polynomially learnable by $H_1 \wedge H_2$.*

As corollaries we have that the following classes of Boolean formulae are polynomially learnable:

Corollary 3 *For any fixed k, let $kCNF \vee kDNF = \cup_{n \geq 1}(kCNF_n \vee kDNF_n)$. Then $kCNF \vee kDNF$ is polynomially learnable by $kCNF \vee kDNF$.*

Corollary 4 *For any fixed k, let $kCNF \wedge kDNF = \cup_{n \geq 1}(kCNF_n \wedge kDNF_n)$. Then $kCNF \wedge kDNF$ is polynomially learnable by $kCNF \wedge kDNF$.*

Proofs of Corollaries 3 and 4 follow from Theorems 1 and 2 and the algorithms in [V84] for learning $kCNF$ from positive examples and $kDNF$ from negative examples. Note that algorithms obtained in Corollaries 3 and 4 use both positive and negative examples. Following Theorem 12 of Section 4 we show that the representation classes $kCNF \vee kDNF$ and $kCNF \wedge kDNF$ *require* both positive and negative examples for polynomial learnability, regardless of the hypothesis class. We note that the more recent results of

Rivest [R87] imply that the above classes are learnable by the class of k-decision lists, a class that properly includes them.

Under the stronger assumption that both C_1 and C_2 are learnable from positive examples, we can prove the following result, which shows that the classes that are polynomially learnable from positive examples are closed under conjunction of representations.

Theorem 5 *Let C_1 be polynomially learnable by H_1 from positive examples, and let C_2 be polynomially learnable by H_2 from positive examples. Then $C_1 \wedge C_2$ is polynomially learnable by $H_1 \wedge H_2$ from positive examples.*

Proof: Let A_1 be a polynomial-time positive-only algorithm for learning C_1 by H_1, and let A_2 be a polynomial-time positive-only algorithm for learning C_2 by H_2. We describe a polynomial-time positive-only algorithm A for learning $C_1 \wedge C_2$ by $H_1 \wedge H_2$ that uses A_1 and A_2 as subroutines.

Let $c = c_1 \wedge c_2$ be the target representation in $C_1 \wedge C_2$, where $c_1 \in C_1$ and $c_2 \in C_2$, and let D^+ and D^- be the target distributions on $pos(c)$ and $neg(c)$. Since $pos(c) \subseteq pos(c_1)$, A can use A_1 to learn a representation $h_1 \in H_1$ for c_1 using the positive examples from D^+ generated by POS. A simulates algorithm A_1 with accuracy parameter $\frac{\epsilon}{2}$ and confidence parameter $\frac{\delta}{2}$, and obtains $h_1 \in H_1$ that with high probability satisfies $e^+(h_1) \leq \frac{\epsilon}{2}$. Note that although we are unable to bound $e^-(h_1)$ by $\frac{\epsilon}{2}$, we must have

$$\begin{aligned}\mathbf{Pr}_{x \in D^-}(x \in pos(h_1) - pos(c_1)) &= \mathbf{Pr}_{x \in D^-}(x \in pos(h_1) \text{ and } x \in neg(c_1)) \\ &\leq \mathbf{Pr}_{x \in D^-}(x \in pos(h_1)) | x \in neg(c_1)) \leq \frac{\epsilon}{2}\end{aligned}$$

since A_1 must work for any fixed distribution on $neg(c_1)$. Similarly, A simulates algorithm A_2 with accuracy parameter $\frac{\epsilon}{2}$ and confidence parameter $\frac{\delta}{2}$ to obtain $h_2 \in H_2$ that with high probability satisfies $e^+(h_2) \leq \frac{\epsilon}{2}$ and $\mathbf{Pr}_{x \in D^-}(pos(h_2) - pos(c_2)) \leq \frac{\epsilon}{2}$. Then we have

$$e^+(h_1 \wedge h_2) \leq e^+(h_1) + e^+(h_2) \leq \epsilon.$$

We now bound $e^-(h_1 \wedge h_2)$ as follows:

$$\begin{aligned}e^-(h_1 \wedge h_2) &= \mathbf{Pr}_{x \in D^-}(x \in pos(h_1 \wedge h_2) - pos(c_1 \wedge c_2)) \\ &= \mathbf{Pr}_{x \in D^-}(x \in pos(h_1) \cap pos(h_2) \cap neg(c_1 \wedge c_2)) \\ &= \mathbf{Pr}_{x \in D^-}(x \in pos(h_1) \cap pos(h_2) \cap (neg(c_1) \cup neg(c_2))) \\ &= \mathbf{Pr}_{x \in D^-}(x \in (pos(h_1) \cap pos(h_2) \cap neg(c_1)) \cup (pos(h_1) \cap pos(h_2) \cap neg(c_2))) \\ &\leq \mathbf{Pr}_{x \in D^-}(x \in pos(h_1) \cap pos(h_2) \cap neg(c_1)) + \mathbf{Pr}_{x \in D^-}(x \in pos(h_1) \cap pos(h_2) \cap neg(c_2)) \\ &\leq \mathbf{Pr}_{x \in D^-}(x \in pos(h_1) \cap neg(c_1)) + \mathbf{Pr}_{x \in D^-}(x \in pos(h_2) \cap neg(c_2)) \\ &= \mathbf{Pr}_{x \in D^-}(x \in pos(h_1) - pos(c_1)) + \mathbf{Pr}_{x \in D^-}(x \in pos(h_2) - pos(c_2)) \\ &\leq \tfrac{\epsilon}{2} + \tfrac{\epsilon}{2} = \epsilon.\end{aligned}$$

The time required by this simulation is polynomial in the time taken by A_1 and A_2. □

The proof of Theorem 5 generalizes to allow any polynomial number p of conjuncts of representations in the target class. Thus, if $C_1, \ldots, C_p$ are polynomially learnable from positive examples by $H_1, \ldots, H_p$ respectively, then the class $C_1 \wedge \cdots \wedge C_p$ is polynomially learnable by $H_1 \wedge \cdots \wedge H_p$ from positive examples.

We can also prove the following dual to Theorem 5:

$C_1 \vee C_2$ polynomially learnable by $C_1 \vee C_2$?	C_1 polynomially learnable by C_1 from POS	C_1 polynomially learnable by C_1 from NEG	C_1 polynomially learnable by C_1 from POS and NEG
C_2 polynomially learnable by C_2 from POS	NP-hard in some cases	YES from POS and NEG	NP-hard in some cases
C_2 polynomially learnable by C_2 from NEG	YES from POS and NEG	YES from NEG	YES from POS and NEG
C_2 polynomially learnable by C_2 from POS and NEG	NP-hard in some cases	YES from POS and NEG	NP-hard in some cases

Figure 1: Polynomial learnability of $C_1 \vee C_2$ by $C_1 \vee C_2$.

Theorem 6 *Let C_1 be polynomially learnable by H_1 from negative examples, and let C_2 be polynomially learnable by H_2 from negative examples. Then $C_1 \vee C_2$ is polynomially learnable by $H_1 \vee H_2$ from negative examples.*

Again, if $C_1, \ldots, C_p$ are polynomially learnable from negative examples by $H_1, \ldots, H_p$ respectively, then the class $C_1 \vee \cdots \vee C_p$ is polynomially learnable by $H_1 \vee \cdots \vee H_p$ from negative examples, for any polynomial value p.

We can also use Theorems 1, 2, 5 and 6 to characterize the conditions under which the class $C_1 \vee C_2$ (respectively, $C_1 \wedge C_2$) is polynomially learnable by $C_1 \vee C_2$ (respectively, $C_1 \wedge C_2$). Figures 1 and 2 summarize this information, where a "YES" entry indicates that for C_1 and C_2 polynomially learnable as indicated, $C_1 \vee C_2$ (respectively, $C_1 \wedge C_2$) is always polynomially learnable by $C_1 \vee C_2$ (respectively, $C_1 \wedge C_2$), and an entry "NP-hard" indicates that the learning problem is NP-hard for some choice of C_1 and C_2. All NP-hardness results follow from the results of [PV88]. For learning problems, since we also allow randomized algorithms, these NP-hardness results usually are taken to mean that if any of such problem is solved in random polynomial time, *i.e.* in RP, then all NP problems are in RP, that is, $NP = RP$.

3.2 Reductions between Boolean formulae learning problems via variable substitutions

In traditional complexity theory, the notion of polynomial-time reducibility has proven extremely useful for comparing the computational difficulty of problems whose exact complexity or tractability is unresolved. Similarly, in computational learning theory, we might expect that given two representation classes C_1 and C_2 whose polynomial learnability is unresolved, we may still be able to prove conditional statements to the effect that if C_1 is polynomially learnable, then C_2 is polynomially learnable. This suggests a notion of reducibility between learning problems.

$C_1 \wedge C_2$ polynomially learnable by $C_1 \wedge C_2$?	C_1 polynomially learnable by C_1 from POS	C_1 polynomially learnable by C_1 from NEG	C_1 polynomially learnable by C_1 from POS and NEG
C_2 polynomially learnable by C_2 from POS	YES from POS	YES from POS and NEG	YES from POS and NEG
C_2 polynomially learnable by C_2 from NEG	YES from POS and NEG	NP-hard in some cases	NP-hard in some cases
C_2 polynomially learnable by C_2 from POS and NEG	YES from POS and NEG	NP-hard in some cases	NP-hard in some cases

Figure 2: Polynomial learnability of $C_1 \wedge C_2$ by $C_1 \wedge C_2$.

In this section we describe polynomial-time reductions between learning problems for classes of Boolean formulae. These reductions are very general and involve simple variable substitutions. Similar transformations have been given for the mistake-bounded model of learning in [L88]. Recently the notion of reducibility among learning problems has been elegantly generalized and developed into a complexity theory for polynomial-time learnability [PW88].

If $F = \cup_{n \geq 1} F_n$ is a parameterized class of Boolean formulae, we say that F is *naming invariant* if for any formula $f(x_1, \ldots, x_n) \in F_n$, we have $f(x_{\pi(1)}, \ldots, x_{\pi(n)}) \in F_n$, where π is any permutation of $\{1, \ldots, n\}$. We say that F is *upward closed* if for $n \geq 1$, $F_n \subseteq F_{n+1}$.

We note that all of the classes of Boolean formulae studied here are both naming invariant and upward closed. For a simple example, let us consider the class of monotone DNF M_n over n variables. Formula $x_1 + x_2 \in M_k$ for all $k \geq 2$, and so is $x_2 + x_1$.

Theorem 7 *Let $F = \cup_{n \geq 1} F_n$ be a parameterized class of Boolean formulae that is naming invariant and upward closed. Let G be a finite set of Boolean formulae over a constant number of variables k. Let F'_n be the class of formulae obtained by choosing any $f(x_1, \ldots, x_n) \in F_n$ and substituting for one or more of the variables x_i in f any formula $g_i(x_{i_1}, \ldots, x_{i_k})$, where $g_i \in G$, and each $x_{i_j} \in \{x_1, \ldots, x_n\}$ (thus, the formula obtained is still over $x_1, \ldots, x_n$). Let $F' = \cup_{n \geq 1} F'_n$. Then if F is polynomially learnable, F' is polynomially learnable.*

Proof: Let A be a polynomial-time learning algorithm for F. We describe a polynomial-time learning algorithm A' for F' that uses algorithm A as a subroutine. For each formula $g_i \in G$, A' creates n^k new variables $z^i_1, \ldots, z^i_{n^k}$. The intention is that z^i_j will simulate the value of the formula g_i when g_i is given the jth choice in some canonical ordering of k (not necessarily distinct) inputs from $x_1, \ldots, x_n$. Note that there are exactly n^k such choices.

Whenever algorithm A requests a positive or negative example, A' takes a positive or negative example $(v_1, \ldots, v_n) \in \{0,1\}^n$ of the target formula $f'(x_1, \ldots, x_n) \in F'_n$. Let $c^i_j \in \{0,1\}$ be the value assigned to z^i_j

by the simulation described above. Then A' gives the example

$$(v_1, \ldots, v_n, c_1^1, \ldots, c_{n^k}^1, \ldots, c_1^{|G|}, \ldots, c_{n^k}^{|G|})$$

to algorithm A. Since $f'(x_1, \ldots, x_n)$ was obtained by substitutions on some $f \in F_n$, and since F is naming invariant and upward closed, there is a formula in $F_{n+|G|n^k}$ that is consistent with all the examples we generate by this procedure (it is just f' with each occurrence of the formula g_i replaced by the variable z_j^i that simulates the inputs to the occurrence of g_i). Thus A must output an ϵ-good hypothesis

$$h_A(x_1, \ldots, x_n, z_1^1, \ldots, z_{n^k}^1, \ldots, z_1^{|G|}, \ldots, z_{n^k}^{|G|}).$$

We then obtain an ϵ-good hypothesis over n variables by defining

$$h_{A'}(v_1, \ldots, v_n) = h_A(v_1, \ldots, v_n, c_1^1, \ldots, c_{n^k}^1, \ldots, c_1^{|G|}, \ldots, c_{n^k}^{|G|})$$

for any $(v_1, \ldots, v_n) \in \{0,1\}^n$, where each c_j^i is computed as described above. □

Note that if the learning algorithm A for F uses only positive examples or only negative examples, this property is preserved by the reduction of Theorem 7. As a corollary of Theorem 7 we have that for most natural Boolean formula classes, the monotone learning problem is no harder than the general learning problem:

Corollary 8 *Let $F = \cup_{n \geq 1} F_n$ be a parameterized class of Boolean formulae that is naming invariant and upward closed. Let monotone F be the class containing all monotone formulae, i.e., formulae containing no negations, in F. Then if monotone F is polynomially learnable, F is polynomially learnable.*

Proof: In the statement of Theorem 7, let $G = \{\overline{y}\}$. Then all of the literals $\overline{x}_1, \ldots, \overline{x}_n$ can be obtained as instances of the single formula in G. □

Theorem 7 says that the learning problem for a class of Boolean formulae does not become harder if an unknown subset of the variables is replaced by a constant-sized set of formulae whose inputs are unknown. The following result says this is also true if the number of substitution formulae is larger, but the order and inputs are known.

Theorem 9 *Let $F = \cup_{n \geq 1} F_n$ be a parameterized class of Boolean formulae that is naming invariant and upward closed. Let $p(n)$ be a fixed polynomial, and let the description of the $p(n)$-tuple $(g_1^n, \ldots, g_{p(n)}^n)$ be computable in polynomial time on input n in unary, where each g_i^n is a Boolean formula over n variables. Let F'_n consist of formulae of the form*

$$f(g_1^n(x_1, \ldots, x_n), \ldots, g_{p(n)}^n(x_1, \ldots, x_n))$$

where $f \in F_{p(n)}$. Let $F' = \cup_{n \geq 1} F'_n$. Then if F is polynomially learnable, F' is polynomially learnable.

Proof: Let A be a polynomial-time learning algorithm for F. We describe a polynomial-time learning algorithm A' for F' that uses algorithm A as a subroutine. Similar to the proof of Theorem 7, A' creates new variables $z_1, \ldots, z_{p(n)}$. The intention is that z_i will simulate $g_i^n(x_1, \ldots, x_n)$.

When algorithm A requests a positive or a negative example, A' takes a positive or negative example $(v_1, \ldots, v_n) \in \{0,1\}^n$ of the target formula $f'(x_1, \ldots, x_n) \in F'_n$ and sets $c_i = g_i^n(v_1, \ldots, v_n)$. A' then gives the vector $(c_1, \ldots, c_{p(n)})$ to A. As in the proof of Theorem 7, A must output an ϵ-good hypothesis h_A over $p(n)$ variables. We then define $h_{A'}(v_1, \ldots, v_n) = h_A(c_1, \ldots, c_{p(n)})$, for any $v_1, \ldots, v_n \in \{0,1\}^n$, where each c_i is computed as described above. □

If the learning algorithm A for F uses only positive examples or only negative examples, this property is preserved by the reduction of Theorem 9.

Corollary 10 *Let $F = \cup_{n \geq 1} F_n$ be a parameterized class of Boolean formulae that is naming invariant and upward closed. Let μF consist of all formulae in F in which each variable occurs at most once. Then if μF is polynomially learnable, F is polynomially learnable.*

Proof: Let $f \in F$, and let l be the maximum number of times any variable occurs in f. Then in the statement of Theorem 9, let $p(n) = ln$ and $g^n_{in+j} = x_j$ for $0 \leq i \leq l-1$ and $1 \leq j \leq n$. Note that if we do not know the value of l, we can try successively larger values, testing the hypothesis each time until an ϵ-good hypothesis is obtained. □

Corollaries 8 and 10 are particularly useful for simplifying the learning problem for classes whose polynomial-time learnability is in question. For example:

Corollary 11 *If monotone μDNF (respectively, monotone μCNF) is polynomially learnable, then DNF (CNF) is polynomially learnable.*

It is important to note that the substitutions suggested by Theorems 7 and 9 and their corollaries do not preserve the underlying target distributions. For example, it does *not* follow from Corollary 11 that if monotone μDNF is polynomially learnable under uniform target distributions (as is shown in Section 5) then DNF is polynomially learnable under uniform distributions. The results presented in this section should also work for other concept classes whenever similar conditions hold, as pointed out by one referee.

4 A Negative Result

A number of of polynomial-time learning algorithms in the literature require only positive examples or only negative examples. Among other issues, this raises the question of whether every polynomially learnable class is polynomially learnable either from positive examples only or from negative examples only.

In this section we prove a superpolynomial lower bound on the number of examples required for learning monomials from negative examples. The proof can actually be tightened to give a strictly exponential lower bound, and the proof technique has been generalized in [G89]. A necessary condition for (general) learning from positive examples is given in [S90]. Our bound is information-theoretic in the sense that it holds regardless of the computational complexity and hypothesis class of the negative-only learning algorithm and is independent of any complexity-theoretic assumptions. By duality, we obtain lower bounds on the number of examples needed for learning disjunctions from positive examples, and it follows from our proof that the same bound holds for learning from negative examples any class properly containing monomials (e.g., $k CNF$)

or for learning from positive examples any class properly containing disjunctions (e.g., $kDNF$). In fact, these results hold even for the monotone versions of these classes. We apply our lower bound to show that the polynomially learnable class $kCNF \vee kDNF$ requires both positive and negative examples, thus answering negatively the question raised above.

Theorem 12 *Let A be a negative-only learning algorithm for the class of monotone monomials, and let s_A^- denote the number of negative examples required by A. Then for any n, for $\epsilon, \delta \leq \frac{1}{n}$ and any constant $k > 0$, $s_A^-(n) = \Omega(n^k)$.*

Proof: For any n, fix $\epsilon = \delta \leq \frac{1}{n}$. Assume for contradiction that A is a negative-only learning algorithm for monotone monomials such that $s_A^-(n) = n^k$ for some constant k and $\epsilon, \delta \leq \frac{1}{n}$. Call a monomial *monotone dense* if it is monotone and contains at least $\frac{n}{2}$ of the variables. Let T be an ordered sequence of (not necessarily distinct) vectors from $\{0,1\}^n$ such that $|T| = n^k$, and let Ψ be the set of all such sequences. If c is a monotone dense monomial, then define $\vec{u}_c \in \{0,1\}^n$ to be the unique vector such that $\vec{u}_c \in pos(c)$ and $\vec{u}_c$ has the fewest bits set to 1. We say that $T \in \Psi$ is *legal negative* for c if T contains no vector $\vec{v}$ such that $\vec{v} \in pos(c)$.

We first define target distributions for a monotone dense monomial c. Let $D^+(\vec{u}_c) = 1$ and let D^- be uniform over $neg(c)$. Note that $\vec{u}_c \notin pos(h)$ implies $e^+(h) = 1$, so any ϵ-good h must satisfy $\vec{u}_c \in pos(h)$. For $T \in \Psi$ and c a monotone dense monomial, define the predicate $P(T, c)$ to be 1 if and only if T is legal negative for c and when T is received by A as a sequence of negative examples for c from NEG, A outputs an hypothesis h_A such that $\vec{u}_c \in pos(h_A)$. Note that this definition assumes that A is deterministic. To allow probabilistic algorithms, we simply change the definition to $P(T, c) = 1$ if and only if T is legal negative for c, and when T is given to A, A outputs an hypothesis h_A such that $\vec{u}_c \in pos(h_A)$ with probability at least $\frac{1}{2}$, where the probability is taken over the coin tosses of A.

Suppose we draw $\vec{v}$ uniformly at random from $\{0,1\}^n$. Fix any monotone dense monomial c. We have

$$\mathbf{Pr}_{\vec{v} \in \{0,1\}^n}(\vec{v} \in pos(c)) \leq 2^{\frac{n}{2}} \frac{1}{2^n} = \frac{1}{2^{\frac{n}{2}}}$$

since at most $2^{\frac{n}{2}}$ vectors can satisfy a monotone dense monomial. Thus, if we draw n^k points uniformly at random from $\{0,1\}^n$, the probability that we draw *some* point satisfying c is at most $\frac{n^k}{2^{\frac{n}{2}}} \leq \frac{1}{2}$ for n large enough. By this analysis, we conclude that the number of $T \in \Psi$ that are legal negative for c must be at least $\frac{|\Psi|}{2}$. Since D^- is uniform and A is a learning algorithm, at least $\frac{|\Psi|}{2}(1-\delta) = \frac{|\Psi|}{2}(1-\epsilon)$ of these must satisfy $P(T, c) = 1$. Let $M(n)$ be the number of monotone dense monomials over n variables. Then summing over all monotone dense monomials, we obtain

$$\frac{|\Psi|}{2}(1-\epsilon)M(n) \quad \leq \quad \sum_{T \in \Psi} N(T)$$

where $N(T)$ is defined to be the number of monotone dense monomials satisfying $P(T, c) = 1$. From this inequality, and the fact that $N(T)$ is always at most $M(n)$, we conclude that at least $\frac{1}{8}$ of the $T \in \Psi$ must satisfy $N(T) \geq \frac{1}{8}(1-\epsilon)M(n)$. Since D^- is uniform, and since at least a fraction $1 - \frac{n^k}{2^{\frac{n}{2}}}$, for large n, of the $T \in \Psi$ are legal negative for the target monomial c, A has probability at least $\frac{1}{16}$ of receiving a T with such

a large $N(T)$ for n large enough. But then the hypothesis h_A output by A has at least $\frac{1}{8}(1-\epsilon)M(n)$ positive examples by definition of the predicate P. Since the target monomial c has at most $2^{\frac{n}{2}}$ positive examples,

$$e^-(h_A) \geq \frac{\frac{1}{8}(1-\epsilon)M(n) - 2^{\frac{n}{2}}}{2^n}$$

and this error must be less than ϵ. But this cannot be true for ϵ a small enough constant and n large enough. Thus, A cannot achieve arbitrarily small error on monotone dense monomials, and the theorem follows. □

An immediate consequence of Theorem 12 is that monomials are not polynomially learnable from negative examples (regardless of the hypothesis class). This is in contrast to the fact that monomials are polynomially learnable (by monomials) from positive examples [V84]. It also follows that any class that contains the class of monotone monomials (e.g., $k\mathit{CNF}$) is not polynomially learnable from negative examples. Further, Theorem 12 implies that the polynomially learnable classes $k\mathit{CNF} \vee k\mathit{DNF}$ and $k\mathit{CNF} \wedge k\mathit{DNF}$ of Corollaries 3 and 4 require both positive and negative examples for polynomial learnability. The same is true for the class of decision lists studied by Rivest [R87].

We also note that it is possible to obtain similar but weaker results with simpler proofs. For instance, since 2-term DNF is not learnable by 2-term DNF unless $NP = RP$, it follows from Theorem 1 that monomials are not polynomially learnable by monomials from negative examples unless $NP = RP$. However, Theorem 12 gives a lower bound on the sample size for negative-only algorithms that holds regardless of the hypothesis class, and is independent of any complexity-theoretic assumption.

By duality we have the following lower bound on the number of positive examples needed for learning monotone disjunctions.

Corollary 13 *Let A be a positive-only learning algorithm for the class of monotone disjunctions, and let s_A^+ denote the number of positive examples required by A. Then for $\epsilon, \delta \leq \frac{1}{n}$ and any constant $k > 0$, $s_A^+(n) = \Omega(n^k)$.*

We conclude this section with a brief discussion of how the lower bound of Theorem 12 and similar lower bounds can be used to obtain lower bounds on the expected number of examples for algorithms whose sample size may depend on coin flips and the actual sequence of examples received. We first define what we mean by the expected number of examples. Let A be a (randomized) learning algorithm for a class C, let $\tilde{r}$ be an infinite sequence of bits (interpreted as the random coin tosses for A), and let $\tilde{w}$ be an infinite sequence of alternating positive and negative examples of some $c \in C$. Then we define $s_A(\epsilon, \delta, \tilde{r}, \tilde{w})$ to be the number of examples read by A (where each request for an example results in either the next positive or next negative example being read from $\tilde{w}$) on inputs ϵ, δ, $\tilde{r}$ and $\tilde{w}$. The *expected sample complexity* of A is then the supremum over all $c \in C$ and all target distributions D^+ and D^- for c of the expectation $\mathbf{E}(s_A(\epsilon, \delta, \tilde{r}, \tilde{w}))$, where the infinite bit sequence $\tilde{r}$ is drawn uniformly at random and the infinite example sequence $\tilde{w}$ is drawn randomly according to the target distributions D^+ and D^-.

The basic format of the proof of the lower bound of Theorem 12 (as well as those of [BEHW86] and [EHKV88]) is to give specific distributions such that a random sample of size at most B has probability at least p of causing any learning algorithm to fail to output an ϵ-good hypothesis. To obtain a lower bound on the expected sample complexity, let A be any learning algorithm, and let q be the probability

that A draws fewer than B examples when run on the same distributions that were given to prove the deterministic lower bound. Then the probability that algorithm A fails to output an ϵ-good hypothesis is bounded below by pq. Since A is a learning algorithm we must have $pq \leq \delta$, so $q \leq \frac{\delta}{p}$. This gives a lower bound of $(1 - \frac{\delta}{p})B$ on the expected sample complexity. Since the value of p proved in Theorem 12 is $\frac{1}{16}$, we immediately obtain an asymptotic lower bound of $\Omega(n^k)$ for any constant k on the expected sample complexity of any negative-only learning algorithm for monomials.

5 Distribution-specific Learning in Polynomial Time

It has been shown elsewhere that for several natural representation classes the learning problem is computationally intractable (modulo various complexity-theoretic or cryptographic assumptions), in some cases even if we allow arbitrary polynomially evaluatable hypothesis representations (see e.g. [KV89, PV88, PW88] for hardness results in the distribution-free model). In other cases, most notably the class of unrestricted DNF formulae, researchers have been unable to provide firm evidence for either the polynomial-time learnability or the intractability of learning. Given this state of affairs, we seek to obtain partial positive results by weakening our demands on a learning algorithm, thus making the computational problem easier (in cases such as Boolean formulae, where we already have strong evidence for intractability) or the mathematical problem easier (in cases such as DNF, where essentially nothing is currently known). This approach has been pursued in at least two directions: by providing learning algorithms with additional information about the target concept in the form of queries, and by relaxing the demand for performance against arbitrary target distributions to that of performance against specific natural distributions. In this section we describe results in the latter direction. Distribution-specific learning is also considered in [BI88, N87].

We describe polynomial-time algorithms for learning under uniform distributions representation classes for which the learning problem under arbitrary distributions is either intractable or unresolved. We begin with an algorithm for weakly learning the class of all monotone Boolean functions under uniform target distributions.

5.1 A polynomial-time weak learning algorithm for all monotone Boolean functions under uniform distributions

For $T \subseteq \{0,1\}^n$ and $\vec{u}, \vec{v} \in \{0,1\}^n$ define

$$\vec{u} \oplus \vec{v} = (u_1 \oplus v_1, \ldots, u_n \oplus v_n)$$

and $T \oplus \vec{v} = \{\vec{u} \oplus \vec{v} : \vec{u} \in T\}$. For $1 \leq i \leq n$ let $e(\vec{i})$ be the vector with the ith bit set to 1 and all other bits set to 0.

The following lemma is due to Aldous [A86].

Lemma 14 *[A86] Let $T \subset \{0,1\}^n$ be such that $|T| \leq \frac{2^n}{2}$. Then for some $1 \leq i \leq n$,*

$$|T \oplus e(\vec{i}) - T| \geq \frac{|T|}{2n}.$$

Theorem 15 *The class of all monotone Boolean functions is polynomially weakly learnable under uniform D^+ and uniform D^-.*

Proof: Let f be any monotone Boolean function on $\{0,1\}^n$. First assume that $|pos(f)| \leq \frac{2^n}{2}$. For $\vec{v} \in \{0,1\}^n$ and $1 \leq i \leq n$, let $\vec{v}_{i=b}$ denote $\vec{v}$ with the ith bit set to $b \in \{0,1\}$, *i.e.*, $v_i = b$.

Now suppose that $\vec{v} \in \{0,1\}^n$ is such that $\vec{v} \in neg(f)$ and $v_j = 1$ for some $1 \leq j \leq n$. Then $\vec{v}_{j=0} \in neg(f)$ by monotonicity of f. Thus for any $1 \leq j \leq n$ we must have

$$\mathbf{Pr}_{\vec{v} \in D^-}(v_j = 1) \leq \frac{1}{2} \tag{4}$$

since D^- is uniform over $neg(f)$.

Let $e(\vec{i})$ be the vector satisfying $|pos(f) \oplus e(\vec{i}) - pos(f)| \geq \frac{|pos(f)|}{2n}$ in Lemma 14 above. Let $\vec{v} \in \{0,1\}^n$ be such that $\vec{v} \in pos(f)$ and $v_i = 0$. Then $\vec{v}_{i=1} \in pos(f)$ by monotonicity of f. However, by Lemma 14, the number of $\vec{v} \in pos(f)$ such that $v_i = 1$ and $\vec{v}_{i=0} \in neg(f)$ is at least $\frac{|pos(f)|}{2n}$. Thus, we have

$$\mathbf{Pr}_{\vec{v} \in D^+}(v_i = 1) \geq \frac{1}{2} + \frac{1}{4n}. \tag{5}$$

Similarly, if $|neg(f)| \leq \frac{2^n}{2}$, then for any $1 \leq j \leq n$ we must have

$$\mathbf{Pr}_{\vec{v} \in D^+}(v_j = 0) \leq \frac{1}{2} \tag{6}$$

and for some $1 \leq i \leq n$,

$$\mathbf{Pr}_{\vec{v} \in D^-}(v_i = 0) \geq \frac{1}{2} + \frac{1}{4n}. \tag{7}$$

Note that either $|pos(f)| \leq \frac{2^n}{2}$ or $|neg(f)| \leq \frac{2^n}{2}$.

We use these differences in probabilities to construct a polynomial-time weak learning algorithm A. A first assumes $|pos(f)| \leq \frac{2^n}{2}$; if this is the case, then Equations 4 and 5 must hold. A then finds an index $1 \leq k \leq n$ satisfying

$$\mathbf{Pr}_{\vec{v} \in D^+}(v_k = 1) \geq \frac{1}{2} + \frac{1}{8n} \tag{8}$$

The existence of such a k is guaranteed by Equation 5. A finds such a k with high probability by sampling POS enough times according to Fact CB1 and Fact CB2 to obtain an estimate p of $\mathbf{Pr}_{\vec{v} \in D^+}(v_k = 1)$ satisfying

$$\mathbf{Pr}_{\vec{v} \in D^+}(v_k = 1) - \frac{1}{8n} < p < \mathbf{Pr}_{\vec{v} \in D^+}(v_k = 1) + \frac{1}{8n}.$$

If A successfully identifies an index k satisfying Equation 8, then the hypothesis h_A is defined as follows: given an unlabeled input vector $\vec{v}$, h_A flips a biased coin and with probability $\frac{1}{16n}$ classifies $\vec{v}$ as negative. With probability $1 - \frac{1}{16n}$, h_A classifies $\vec{v}$ as positive if $v_i = 1$ and as negative if $v_i = 0$. It is easy to verify by Equations 4 and 8 that this is a randomized hypothesis meeting the conditions of weak learnability.

If A is unable to identify an index k satisfying Equation 8, then A assumes that $|neg(f)| \leq \frac{2^n}{2}$, and in a similar fashion proceeds to form a hypothesis h_A based on the differences in probability of Equations 6 and 7. □

It is shown in [BEHW86] (see also [EHKV88]) that the number of examples needed for learning (and therefore the computation time required) is bounded below by the Vapnik-Chervonenkis dimension of the

target class; furthermore, this lower bound is proved using the uniform distribution over a shattered set and holds even for the weak learning model in the case of superpolynomial Vapnik-Chervonenkis dimension. From this it follows that the class of monotone Boolean functions is not polynomially weakly learnable under arbitrary target distributions (since the Vapnik-Chervonenkis dimension of this class is exponential in n) and that the class of all Boolean functions is not polynomially weakly learnable under uniform target distributions (since the entire set $\{0,1\}^n$ is shattered). It can also be shown that the class of monotone Boolean functions is not polynomially (strongly) learnable under uniform target distributions — to see this, consider only those monotone functions defined by an arbitrary set of vectors with exactly half the bits on. The positive examples are the vectors that can be obtained by choosing one of the vectors in the defining set and turning 0 or more of its off bits on. This is clearly a monotone function, and it is not possible to achieve $\epsilon < \frac{1}{\sqrt{n}}$ on the uniform distribution. Thus, Theorem 15 is optimal in the sense that generalization in any direction — uniform distributions to arbitrary distributions, weak learning to strong learning, or monotone functions to arbitrary functions — results in intractability.

5.2 A polynomial-time learning algorithm for μDNF under uniform distributions

We now give a polynomial-time algorithm for learning DNF in which each variable occurs at most once (μDNF) under uniform target distributions. Recall that in the distribution-free setting, this learning problem is as hard as the general DNF learning problem by Corollary 11.

Theorem 16 *μDNF is polynomially learnable by μDNF under uniform D^+ and uniform D^-.*

Proof: Let $f = m_1 + \cdots m_s$ be the target μDNF formula over n variables, where each m_i is a monomial. Let d be a constant such that $n^d = \frac{1}{\epsilon}$, for $n \geq 2$. Note, when $n = 1$, this case can be handled trivially. We say that a monomial m appearing in f is *significant* if $\mathbf{Pr}_{\vec{v} \in D^+}(\vec{v} \in pos(m)) \geq \frac{\epsilon}{4n} = \frac{1}{4n^{d+1}}$. Note that $s \leq n$ since no variable appears twice in f. Thus the error on D^+ incurred by ignoring all monomials that are not significant is at most $\frac{\epsilon}{4}$. We now give an outline of the learning algorithm and then show how each step can be implemented and prove its correctness. For simplicity, our algorithm assumes that the target formula is monotone; this restriction is easily removed, because since each variable appears at most once we may simply regard any occurrence of $\overline{x_i}$ as an unnegated occurrence of a new variable y_i.

Algorithm A:

Step 1. Assume that every significant monomial in f has at least $r \log n$ literals for $r = 2d$. This step will learn an approximation for 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 (using only negative examples). The substeps of Step 1 are as follows:

Substep 1.1. For each i, use positive examples to determine whether the variable x_i appears in one of the significant monomials of f. (With high probability, this step will find all variables in significant monomials, some variables in insignificant monomials, and no variables not in f.).

Substep 1.2. For each i, j such that variables x_i and x_j were determined in Substep 1.1 to appear in some monomial of f, use positive examples to decide whether they appear in the same significant monomial.

Substep 1.3. Form a μDNF hypothesis h_A in the obvious way.

Step 2. Decide whether h_A is an ϵ-good hypothesis by testing it on a polynomial number of positive and negative examples. If it is decided that h_A is ϵ-good, stop and output h_A. Otherwise, guess that the assumption of Step 1 is not correct and go to Step 3.

Step 3. Assuming that some significant monomial in f is shorter than $rlogn$, we can also assume that all the monomials are shorter than $2rlogn$, since the longer ones are not significant. We use only negative examples in this step. The substeps are:

Substep 3.1. For each i, use negative examples to determine whether variable x_i appears in some significant monomial of f. (Again, this might accidentally include some variables in insignificant monomials, as in Step 1.1.)

Substep 3.2. For each i, j such that variables x_i and x_j were determined in Substep 3.1 to appear in some monomial, use negative examples to decide if they appear in the same monomial.

Substep 3.3. Form a μDNF hypothesis h_A in the obvious way and stop.

Throughout the following analysis, we will make use of the following fact: let E_1 and E_2 be events over a probability space, and let $\mathbf{Pr}(E_1 \cup E_2) = 1$ with respect to this probability space. Then for any event E, we have

$$\begin{aligned}\mathbf{Pr}(E) &= \mathbf{Pr}(E|E_1)\mathbf{Pr}(E_1) + \mathbf{Pr}(E|E_2)\mathbf{Pr}(E_2) - \mathbf{Pr}(E|E_1 \cap E_2)\mathbf{Pr}(E_1 \cap E_2) \qquad (9)\\ &= \mathbf{Pr}(E|E_1)\mathbf{Pr}(E_1) + \mathbf{Pr}(E|E_2)(1 - \mathbf{Pr}(E_1) + \mathbf{Pr}(E_1 \cap E_2)) \\ &\qquad -\mathbf{Pr}(E|E_1 \cap E_2)\mathbf{Pr}(E_1 \cap E_2) \\ &= \mathbf{Pr}(E|E_1)\mathbf{Pr}(E_1) + \mathbf{Pr}(E|E_2)(1 - \mathbf{Pr}(E_1)) + K,\end{aligned}$$

where $K \leq O(\mathbf{Pr}(E_1 \cap E_2))$.

In Step 1, we draw only positive examples. Since there are at most n (disjoint) monomials in f, and we assume that the size of each monomial is at least $rlogn$, the probability that a positive example of f drawn at random from D^+ satisfies 2 or more monomials of f is at most $\frac{n}{2^{rlogn}} = \frac{1}{n^{r-1}} << \epsilon$. Therefore, in the following analysis, we restrict our attention to positive examples of f which satisfy precisely one monomial of f. Intuitively, such restriction will not be a problem since positive examples that satisfy more than one monomials are very rare (with probability much less than ϵ). Thus we can do analysis as if they were not there, and eventaully these vectors can only cause very small error much less than ϵ.

Analysis of Substep 1.1. For each i, if the variable x_i is not in any monomial of f,

$$\mathbf{Pr}_{\vec{v} \in D^+}(v_i = 0) = \mathbf{Pr}_{\vec{v} \in D^+}(v_i = 1) = \frac{1}{2}$$

since D^+ is uniform. Now suppose that variable x_i appears in a significant monomial m of f. Note that, since the length of a significant monomial is at least $rlogn$, we have,

$$1/2 \geq \mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1|\vec{v} \in neg(m)) \geq \frac{2^{rlogn-1}-1}{2^{rlogn}-1}.$$

Then we have

$$\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1) = \mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1|\vec{v} \in pos(m))\mathbf{Pr}_{\vec{v}\in D^+}(\vec{v} \in pos(m)) \tag{10}$$

$$+\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1|\vec{v} \in neg(m))\mathbf{Pr}_{\vec{v}\in D^+}(\vec{v} \in neg(m))$$

$$\geq \mathbf{Pr}_{\vec{v}\in D^+}(\vec{v} \in pos(m)) + \tfrac{2^{rlogn-1}-1}{2^{rlogn}-1}(1 - \mathbf{Pr}_{\vec{v}\in D^+}(\vec{v} \in pos(m))) \geq \tfrac{1}{2} + \tfrac{1}{8n^{d+1}} - O(\tfrac{1}{n^{2d-1}}).$$

Thus there is a difference of $\Omega(\frac{1}{n^{d+1}})$ between the probability that a variable appearing in a significant monomial is set to 1 and the probability that a variable not appearing in f is set to 1. Using Facts CB1 and CB2, we can determine with high probability if x_i appears in a significant monomial of f by drawing a polynomial number of examples from POS. Notice that if x_i appears in a monomial that is not significant then it really does not matter. We might in this process also find some of those variables in some insignificant monomials with higher probability, and lose some others with lower probability. These variables do not matter to us. This will not affect Substeps 1.2 and 1.3.

Analysis of Substep 1.2. For each pair of variables x_i and x_j that appear in some monomial of f (as decided in Substep 1.1), we now decide whether they appear in the same monomial of f.

Lemma 17 *If variables x_i and x_j appear in the same monomial of f, then*

$$\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \text{ or } v_j = 1) = \frac{3}{4} + \frac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1) - \frac{1}{2}) \pm O(\frac{1}{n^{r-1}}).$$

Proof: Since x_i and x_j appear in the same monomial of f and appear only once in f, we have $\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1) = \mathbf{Pr}_{\vec{v}\in D^+}(v_j = 1)$ since D^+ is uniform. Let m be the monomial of f in which x_i and x_j appear, and let E_1 be the event that m is satisfied. Let E_2 be the event that at least one monomial of f besides (but possibly in addition to) m is satisfied. Note that $\mathbf{Pr}_{\vec{v}\in D^+}(E_1 \cup E_2) = 1$. Using the facts that since D^+ is uniform, $\mathbf{Pr}_{\vec{v}\in D^+}(E_1 \cap E_2) \leq \frac{1}{n^{r-1}}$ (because given that a positive example already satisfies a monomial of f, the remaining variables are independent and uniformly distributed) and $\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1) = \mathbf{Pr}_{\vec{v}\in D^+}(E_1) + \frac{1}{2}(1 - \mathbf{Pr}_{\vec{v}\in D^+}(E_1))$ and by Equation 9, we have

$$\begin{aligned}
&\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \text{ or } v_j = 1)\\
&\quad= \mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \text{ or } v_j = 1|E_1)\mathbf{Pr}_{\vec{v}\in D^+}(E_1)\\
&\qquad\quad +\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \text{ or } v_j = 1|E_2)\mathbf{Pr}_{\vec{v}\in D^+}(E_2) - O(\tfrac{1}{n^{r-1}})\\
&\quad= \mathbf{Pr}_{\vec{v}\in D^+}(E_1) + \tfrac{3}{4}(1 - \mathbf{Pr}_{\vec{v}\in D^+}(E_1)) \pm O(\tfrac{1}{n^{r-1}})\\
&\quad= \tfrac{3}{4} + \tfrac{1}{4}\mathbf{Pr}_{\vec{v}\in D^+}(E_1) \pm O(\tfrac{1}{n^{r-1}})\\
&\quad= \tfrac{3}{4} + \tfrac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1) - \tfrac{1}{2}) \pm O(\tfrac{1}{n^{r-1}}).
\end{aligned}$$

□(Lemma 17)

Lemma 18 *If variables x_i and x_j appear in different monomials of f, then*

$$\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \text{ or } v_j = 1) = \frac{3}{4} + \frac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1) - \frac{1}{2}) + \frac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}(v_j = 1) - \frac{1}{2}) \pm O(\frac{1}{n^{r-1}}).$$

Proof: Let E_1 be the event that the monomial m_1 of f containing x_i is satisfied, and E_2 the event that the monomial m_2 containing x_j is satisfied. Let E_3 be the event that some monomial other than (but possibly in addition to) m_1 and m_2 is satisfied. Note that $\mathbf{Pr}_{\vec{v}\in D^+}(E_1 \cup E_2 \cup E_3) = 1$. Then similar to the proof of Lemma 17, we have

$$\begin{aligned}
&\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \ or \ v_j = 1) \\
&\quad = \mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \ or \ v_j = 1|E_1)\mathbf{Pr}_{\vec{v}\in D^+}(E_1) + \mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \ or \ v_j = 1|E_2)\mathbf{Pr}_{\vec{v}\in D^+}(E_2) \\
&\qquad\qquad + \mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \ or \ v_j = 1|E_3)\mathbf{Pr}_{\vec{v}\in D^+}(E_3) - O(\tfrac{1}{n^{r-1}}) \\
&\quad = \mathbf{Pr}_{\vec{v}\in D^+}(E_1) + \mathbf{Pr}_{\vec{v}\in D^+}(E_2) + \tfrac{3}{4}\mathbf{Pr}_{\vec{v}\in D^+}(E_3) - O(\tfrac{1}{n^{r-1}}) \\
&\quad = \mathbf{Pr}_{\vec{v}\in D^+}(E_1) + \mathbf{Pr}_{\vec{v}\in D^+}(E_2) + \tfrac{3}{4}(1 - \mathbf{Pr}_{\vec{v}\in D^+}(E_1) - \mathbf{Pr}_{\vec{v}\in D^+}(E_2)) \pm O(\tfrac{1}{n^{r-1}}) \\
&\quad = \tfrac{3}{4} + \tfrac{1}{4}(\mathbf{Pr}_{\vec{v}\in D^+}(E_1) + \mathbf{Pr}_{\vec{v}\in D^+}(E_2)) \pm O(\tfrac{1}{n^{r-1}}) \\
&\quad = \tfrac{3}{4} + \tfrac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1) - \tfrac{1}{2}) + \tfrac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}(v_j = 1) - \tfrac{1}{2}) \pm O(\tfrac{1}{n^{r-1}})
\end{aligned}$$

□(Lemma 18)

From Equation 10, Lemma 17, Lemma 18 and the fact that if x_i and x_j appear in the same monomial of f, then $\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1) = \mathbf{Pr}_{\vec{v}\in D^+}(v_j = 1)$, we have that there is a difference $\Omega(\frac{1}{n^{d+1}-n^{r-1}})$ between the value of $\mathbf{Pr}_{\vec{v}\in D^+}(v_i = 1 \ or \ v_j = 1)$ in the two cases addressed by Lemmas 17 and 18. Thus we can determine whether x_i and x_j appear in the same monomial by drawing a polynomial number of examples from POS using Facts CB1 and CB2.

In Step 2, we draw a polynomial number of examples from both POS and NEG to test if the hypothesis h_A produced in Step 1 is ϵ-good, again using Facts CB1 and CB2. If it is determined that h_A is not ϵ-good, then A guesses the assumption made in Step 1 is not correct, and therefore that there is a monomial in f which is of length at most $rlogn$. This implies that all the monomials of length larger than $2rlogn$ are not significant. Therefore in Step 3 we assume that all the monomials in f are shorter than $2rlogn$. We use only the negative examples.

Analysis of Substep 3.1. If variable x_i does not appear in any monomial of f, then

$$\mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0) = \frac{1}{2} \tag{11}$$

since D^+ is uniform.

Lemma 19 *If variable x_i appears in a significant monomial of f, then*

$$\mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0) \geq \frac{1}{2} + \frac{1}{2(n^{2r}-1)}.$$

Proof: Let l be the number of literals in the monomial m of f that variable x_i appears in. Then in a vector $\vec{v}$ drawn at random from D^-, if some bit of $\vec{v}$ is set such that m is already not satisfied, the remaining bits are independent and uniformly distributed. Thus

$$\mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0) = \tfrac{2^{l-1}}{2^l-1} = \tfrac{1}{2} + \tfrac{1}{2(2^l-1)}.$$

Since $l \leq 2rlogn$, the claim follows. □(Lemma 19)

By Equation 11 and Lemma 19, there is a difference of $\Omega(\frac{1}{n^{2r}})$ between the probability that a variable in a significant monomial of f is set to 0 and the probability that a variable not appearing in f is set to 0. Thus we can draw a polynomial number of examples from NEG, and decide if variable x_i appears in some significant monomial of f, using Facts CB1 and CB2.

Analysis of Substep 3.2. We have to decide whether variables x_i and x_j appear in the same monomial of f, given that each appear in some monomial of f.

Lemma 20 *If variables x_i and x_j are not in the same monomial of f, then*

$$\mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0 \text{ and } v_j = 0) = \mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0)\mathbf{Pr}_{\vec{v}\in D^-}(v_j = 0).$$

Proof: If x_i and x_j do not appear in the same monomial, then they are independent of each other with respect to D^- since each variable appears only once in f. □(Lemma 20)

Lemma 21 *If variables x_i and x_j appear in the same monomial of f, then*

$$\mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0 \text{ and } v_j = 0) = \frac{1}{2}\mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0).$$

Proof:

$$\mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0 \text{ and } v_j = 0) = \mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0)\mathbf{Pr}_{\vec{v}\in D^-}(v_j = 0|v_i = 0)$$

But $\mathbf{Pr}_{\vec{v}\in D^-}(v_j = 0|v_i = 0) = \frac{1}{2}$. □(Lemma 21)

By Lemmas 19, 20 and 21 we have that there is a difference of $\Omega(\frac{1}{n^{2r}})$ in the value of $\mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0 \text{ and } v_j = 0)$ in the two cases addressed by Lemmas 20 and 21, since

$$\mathbf{Pr}_{\vec{v}\in D^-}(v_i = 0) \geq \frac{1}{2} + \frac{1}{2(n^{2r}-1)}.$$

Thus we can test if x_i and x_j appear in the same monomial of f by drawing a polynomial number of examples from *NEG* using Facts CB1 and CB2. This completes the proof of Theorem 16. □

The results of [PV88] show that k-term μDNF is not learnable by k-term μDNF unless $NP = RP$. However, the algorithm of Theorem 16 outputs an hypothesis with *at most* the same number of terms as the target formula. This is an example of a class for which learning under arbitrary target distributions is *NP*-hard, but learning under uniform target distributions is tractable.

6 Equivalence of Weak Learning and Group Learning

In this section we prove the equivalence of the model of weak learning with another model which we call *group learning*. Informally, in group learning we ask that the learning algorithm output an hypothesis that is $1-\epsilon$ accurate in classifying a polynomial-size group of examples that are either all positive or all negative. Thus, the basic model of (strong) learnability is a special case of group learning where the group size is 1. The question we wish to address here is whether learning becomes easier in some cases if the group size is allowed to be larger.

Recently it has been shown by Schapire [S89] that in the distribution-free setting, polynomial-time weak learning is in fact equivalent to polynomial-time strong learning. His proof gives a recursive technique for taking an algorithm outputting hypotheses with accuracy slightly above $\frac{1}{2}$ and constructing hypotheses of accuracy $1-\epsilon$. This result combined with ours shows that group learning is in fact equivalent to strong learning. Thus, allowing the hypothesis to accurately classify only larger groups of (all positive or all

negative) examples does not increase what is polynomially learnable. These results also demonstrate the robustness of our underlying model of learnability, since it is invariant under these apparently significant but reasonable modifications. Related equivalences are given in [HKLW88].

Our equivalence proof also holds in both directions under *fixed* target distributions: thus, C is polynomially group learnable under a restricted class of distributions if and only if C is polynomially weakly learnable under these same distributions. As an immediate corollary, we have by the results of Section 5.1 that the class of all monotone Boolean functions is group learnable in polynomial time under uniform distributions. Furthermore, since it was argued in Section 5.1 that the class of all monotone Boolean functions cannot be strongly learned in polynomial time under uniform distributions, there cannot be a distribution-preserving reduction of the strong learning model to the weak learning model. Thus, the problem of learning monotone functions under uniform distributions exhibits a trade-off: we may either have accuracy slightly better than guessing on single examples, or high accuracy on large groups of examples, but not high accuracy on single examples.

Our formal definitions are as follows: for p any fixed polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}$ and $|c|$, the hypothesis h_A of learning algorithm A is now defined over $X^{p(\frac{1}{\epsilon},\frac{1}{\delta},|c|)}$. We ask that if $p(\frac{1}{\epsilon},\frac{1}{\delta},|c|)$ examples all drawn from D^+ (respectively, D^-) are given to h_A, then h_A classifies this *group* as positive (negative) with probability at least $1-\epsilon$. If A runs in polynomial time, we say that C is *polynomially group learnable.*

Theorem 22 *Let C be a polynomially evaluatable parameterized Boolean representation class. Then C is polynomially group learnable if and only if C is polynomially weakly learnable.*

Proof: (If) Let A be a polynomial-time weak learning algorithm for C. We construct a polynomial-time group learning algorithm A' for C as follows: A' first simulates algorithm A and obtains an hypothesis h_A that with probability $1-\delta$ has accuracy $\frac{1}{2}+\frac{1}{p(|c|,n)}$ for some polynomial p. Now given m examples that are either all drawn from D^+ or all drawn from D^-, A' evaluates h_A on each example. Suppose all m points are drawn from D^+. Assuming that h_A in fact has accuracy $\frac{1}{2}+\frac{1}{p(|c|,n)}$, the probability that h_A evaluates as positive on fewer than $(\frac{1}{2}+\frac{1}{2p(|c|,n)})m$ of the examples can be made smaller than $\frac{\epsilon}{2}$ by choosing m to be a large enough polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{p(|c|,n)}$ using Fact CB1. On the other hand, if all m examples are drawn from D^- then the probability that h_A evaluates as positive on more than $(\frac{1}{2}+\frac{1}{2p(|c|,n)})m$ can be made smaller than $\frac{\epsilon}{2}$ for m large enough by Fact CB2. Thus, if h_A evaluates as positive on more than $(\frac{1}{2}+\frac{1}{2p(|c|,n)})m$ of the m examples, A' guesses that the sample is positive; otherwise, A' guesses that the sample is negative. The probability of misclassifying the sample is then at most ϵ, so A' is a group learning algorithm.

(Only if) Let A be a polynomial-time group learning algorithm for C. We use A as a subroutine in a polynomial-time weak learning algorithm A'. Suppose algorithm A is run to obtain with high probability an $\frac{\epsilon}{2}$-good hypothesis h_A for groups of size $l = p(\frac{2}{\epsilon},\frac{1}{\delta},|c|,n)$ all drawn from D^+ or all drawn from D^- for some polynomial p. Note that although h_A is guaranteed to produce an output only when given l positive examples or l negative examples, the *probability* that h_A produces an output when given a mixture of positive and negative examples is well-defined. Thus for $0 \leq i \leq l$ let q_i denote the probability that h_A evaluates as positive when given as input a group whose first i examples are drawn from D^+ and whose last $l-i$

examples are drawn from D^-. Then since h_A is an $\frac{\epsilon}{2}$-good hypothesis we have $q_0 \leq \frac{\epsilon}{2}$ and $q_l \geq 1 - \frac{\epsilon}{2}$. Thus,

$$(q_l - q_0) \geq (1 - \frac{\epsilon}{2}) - \frac{\epsilon}{2} = 1 - \epsilon.$$

Then

$$1 - \epsilon \leq (q_l - q_0)$$

$$= (q_l - q_{l-1}) + (q_{l-1} - q_{l-2}) + \cdots + (q_1 - q_0).$$

This implies that for some $1 \leq j \leq l$, $(q_j - q_{j-1}) \geq \frac{1-\epsilon}{l}$.

Algorithm A' first runs algorithm A with accuracy parameter $\frac{\epsilon}{2}$. A' next obtains an estimate q'_i of q_i for each $0 \leq i \leq l$ that is accurate within an additive factor of $\frac{1-\epsilon}{16l}$, that is

$$q_i - \frac{1-\epsilon}{16l} \leq q'_i \leq q_i + \frac{1-\epsilon}{16l}. \tag{12}$$

This is done by repeatedly evaluating h_A on groups of l examples in which the first i examples are drawn from D^+ and the rest are drawn from D^-, and computing the fraction of runs for which h_A evaluates as positive. These estimates can be obtained in time polynomial in l, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ with high probability using Facts CB1 and CB2. Now for the j such that $(q_j - q_{j-1}) \geq \frac{1-\epsilon}{l}$, the estimates q'_j and q'_{j-1} will have a difference of at least $\frac{1-\epsilon}{2l}$ with high probability. Furthermore, for any i if $(q_i - q_{i-1}) \leq \frac{1-\epsilon}{4l}$ then with high probability the estimates satisfy $(q'_i - q'_{i-1}) \leq \frac{1-\epsilon}{2l}$. Thus, if k is the index such that $(q_k - q_{k-1})$ is largest, assume without loss of generality that $\frac{1}{2} \geq q_k > q_{k-1}$.

The intermediate hypothesis $h_{A'}$ of A' is now defined as follows: given an example whose classification is unknown, $h_{A'}$ constructs l input examples for h_A consisting of $k-1$ examples drawn from D^+, the unknown example, and $l-k$ examples drawn from D^-. The prediction of $h_{A'}$ is then the same as the prediction of h_A on this constructed group. The probability that $h_{A'}$ predicts positive when the unknown example is drawn from D^+ is then q_k and the probability that $h_{A'}$ predicts positive when the unknown example is drawn from D^- is q_{k-1}.

One problem with the $h_{A'}$ defined at this point is that new examples need to be drawn from D^+ and D^- each time an unknown point is classified. This sampling is eliminated as follows: for U a fixed sequence of $k-1$ positive examples of the target representation and V a fixed sequence of $l-k$ negative examples, define $h_{A'}(U,V)$ to be the one-input intermediate hypothesis described above using the *fixed* constructed sample consisting of U and V. Let $p^+(U,V)$ be the probability that $h_{A'}(U,V)$ classifies a random example drawn from D^+ as positive, and let $p^-(U,V)$ be the probability that $h_{A'}(U,V)$ classifies a random example drawn from D^- as positive. Then for U drawn randomly according to D^+ and V drawn randomly according to D^-, define the random variable

$$R(U,V) = p^+(U,V) - p^-(U,V).$$

Then the expectation of R obeys

$$\mathbf{E}(R(U,V)) \geq 2\alpha$$

where $\alpha = \frac{1-\epsilon}{4l}$. However, it is always true that

$$R(U,V) \leq 1.$$

Thus, let r be the probability that

$$R(U, V) \geq \alpha. \tag{13}$$

Then we have

$$r + (1 - r)(\alpha) \geq 2\alpha.$$

Solving, we obtain $r \geq \alpha = \frac{1-\epsilon}{4l}$.

Thus, A' repeatedly draws U from D^+ and V from D^- until Equation 13 is satisfied; by the above argument, this takes only $\frac{8l}{1-\epsilon}$ tries with high probability. Note that A' can test whether Equation 13 is satisfied in polynomial time. The (almost) final hypothesis $h_{A'}(U_0, V_0)$ simply "hardwires" the successful U_0 and V_0 as the constructed sample, leaving one input free for the example whose classification is to be predicted by $h_{A'}$.

Lastly, we need to "center" the bias of the hypothesis $h_{A'}(U_0, V_0)$. Let b be a value such that

$$b + (1 - b)q_k \geq \frac{1}{2} + \frac{q_k - q_{k-1}}{4}$$

and

$$b + (1 - b)q_{k-1} \leq \frac{1}{2} - \frac{q_k - q_{k-1}}{4}.$$

Note that A' can compute an accurate estimate b' of b from accurate estimates of q_k and q_{k-1}. The final hypothesis $h_{A'}$ of A' is now defined as follows: given an example whose classification is unknown, $h_{A'}$ flips a coin of bias b'. If the outcome is heads, $h_{A'}$ predicts that the input example is positive. If the outcome is tails, $h_{A'}$ predicts with the classification given by $h_{A'}(U_0, V_0)$. Then we have

$$1 - e^+(h_{A'}) \geq b' + (1 - b')q_k \geq \frac{1}{2} + \frac{1 - \epsilon}{c_0 l}$$

for an appropriate constant $c_0 > 1$, and

$$1 - e^-(h_{A'}) \geq 1 - b' + (1 - b')q_{k-1} \geq \frac{1}{2} + \frac{1 - \epsilon}{c_0 l}.$$

□

7 Concluding Remarks

Our original motivation was to study the learnability of DNF formulae. We have proved some closure properties between the learnable classes. Our method of reducing one learning problem to another turns out to be a useful idea and it has been extensively investigated by L. Pitt and M. Warmuth [PW88], recently. We have proved an information theoretic bound for learning from negative examples only. We also proved that certain classes of Boolean formulae are learnable under various special assumptions on distributions. Many progress has also been made in this direction since a preliminary version of this paper [KLPV87a] was published in 1987. However our major question as whether DNF formulae is learnable or such problem is NP-complete is still wide open.

8 Acknowledgements

We would like to thank Lenny Pitt for helpful discussions on this paper and Umesh Vazirani for many ideas on learning under uniform distributions. We also thank the two very careful referees who have corrected many errors and suggested many improvements.

References

[A86] D. Aldous.
On the Markov chain simulation method for uniform combinatorial distributions and simulated annealing.
University of California at Berkeley Statistics Department,
technical report number 60, 1986.

[AV79] D. Angluin, L.G. Valiant.
Fast probabilistic algorithms for Hamiltonian circuits and matchings.
Journal of Computer and Systems Sciences,
18, 1979, pp. 155-193.

[BI88] G.M. Benedek, A. Itai.
Learnability by fixed distributions.
Proceedings of the 1988 Workshop on Computational Learning Theory,
Morgan Kaufmann Publishers, 1988, pp. 80-90.

[BEHW86] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth.
Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension.
Proceedings of the 18th A.C.M. Symposium on the Theory of Computing,
1986, pp. 273-282.

[C52] H. Chernoff.
A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations.
Annals of Mathematical Statistics,
23, 1952, pp. 493-509.

[EHKV88] A. Ehrenfeucht, D. Haussler, M. Kearns. L.G. Valiant.
A general lower bound on the number of examples needed for learning.
Proceedings of the 1988 Workshop on Computational Learning Theory,
Morgan Kaufmann Publishers, 1988, pp. 139-154.

[GJ79] M. Garey, D. Johnson.
Computers and intractability: a guide to the theory of NP-completeness.
Freeman, 1979.

[G89] M. Gereb-Graus.
Complexity of learning from one-sided examples.
Harvard University, unpublished manuscript, 1989.

[H88] D. Haussler.
Quantifying inductive bias: AI learning algorithms and Valiant's model.
Artificial Intelligence,
36(2), 1988, pp. 177-221.

[HKLW88] D. Haussler, M. Kearns, N. Littlestone, M. Warmuth.
Equivalence of models for polynomial learnability.
Proceedings of the 1988 Workshop on Computational Learning Theory,
Morgan Kaufmann Publishers, 1988, pp. 42-55, and
University of California at Santa Cruz Information Sciences Department,
technical report number UCSC-CRL-88-06, 1988.

[HSW88] D. Helmbold, R. Sloan, M. Warmuth.
Bootstrapping one-sided learning.
Unpublished manuscript, 1988.

[KLPV87a] M. Kearns, M. Li, L. Pitt, L.G. Valiant.
On the learnability of Boolean formulae.
Proceedings of the 19th A.C.M. Symposium on the Theory of Computing,
1987, pp. 285-295.

[KLPV87b] M. Kearns, M. Li, L. Pitt, L.G. Valiant.
Recent results on Boolean concept learning.
Proceedings of the 4th International Workshop on Machine Learning,
Morgan Kaufmann Publishers, 1987, pp. 337-352.

[KV89] M. Kearns, L.G. Valiant.
Cryptographic limitations on learning Boolean formulae and finite automata.
Proceedings of the 21st A.C.M. Symposium on the Theory of Computing,
1989, pp. 433-444.

[L88] N. Littlestone.
Learning quickly when irrelevant attributes abound: a new linear threshold algorithm.
Machine Learning,
2(4), 1988, pp. 245-318.

[N87] B.K. Natarajan.
On learning Boolean functions.
Proceedings of the 19th A.C.M. Symposium on the Theory of Computing,
1987, pp. 296-304.

[PV88] L. Pitt, L.G. Valiant.
Computational limitations on learning from examples.
Journal of the A.C.M.,
35(4), 1988, pp. 965-984.

[PW88] L. Pitt, M.K. Warmuth.
Reductions among prediction problems: on the difficulty of predicting automata.
Proceedings of the 3rd I.E.E.E. Conference on Structure in Complexity Theory,
1988, pp. 60-69.

[R87] R. Rivest.
Learning decision lists.
Machine Learning,
2(3), 1987, pp. 229-246.

[S89] R. Schapire.
The strength of weak learnability.
Machine Learning,
5(2), 1990, pp. 197-227.

[S90] H. Shvayster.
A necessary condition for learning from positive examples.
Machine Learning,
5(1), 1990, pp. 101-113.

[V84] L.G. Valiant.
A theory of the learnable.
Communications of the A.C.M.,
27(11), 1984, pp. 1134-1142.

[V85] L.G. Valiant.
Learning disjunctions of conjunctions.
Proceedings of the 9th International Joint Conference on Artificial Intelligence,
1985, pp. 560-566.