Computational Limitations on Learning from Examples

LEONARD PITT

University of Illinois, Urbana-Champaign, Urbana, Illinois

AND

LESLIE G. VALIANT

Harvard University, Cambridge, Massachusetts

Abstract. The computational complexity of learning Boolean concepts from examples is investigated. It is shown for various classes of concept representations that these cannot be learned feasibly in a distribution-free sense unless R = NP. These classes include (a) disjunctions of two monomials, (b) Boolean threshold functions, and (c) Boolean formulas in which each variable occurs at most once. Relationships between learning of heuristics and finding approximate solutions to NP-hard optimization problems are given.

Categories and Subject Descriptors: F.1.1 **[Computation by Abstract Devices]**: Models of Computation—*relations among models*; F.1.2 **[Computation by Abstract Devices]**: Modes of Computation—*probabilistic computation*; F.1.3 **[Computation by Abstract Devices]**: Complexity Classes—*reducibility and completeness*; I.2.6 **[Artificial Intelligence]**: Learning—*concept learning*; *induction*

General Terms: Experimentation, Theory, Verification

Additional Key Words and Phrases: Distribution-free learning, inductive inference, learnability, NP-completeness

1. *Introduction*

The problem of inductive inference can be approached from several directions, and among these the viewpoint of computational complexity, namely, the study of inherent computational limitations, appears to be a promising one. An important instance of inference is that of learning a concept from examples and counterexamples. In [32] and [33] it was shown that certain classes of concepts represented as Boolean expressions could be learned in a very strong sense: Whatever the probability distributions from which the examples and counterexamples were drawn, an expression from these classes distinguishing the examples from the

This work was done while L. Pitt was at Harvard University, supported by Office of Naval Research grant N00014-85-K-0445. The work of L. G. Valiant was supported in part by Office of Naval Research grant N00014-85-K-0445, a fellowship from the Guggenheim Foundation, National Science Foundation grant DCR-83-02385, and by the Mathematical Sciences Research Institute, Berkeley, Calif., under grant DAAG 29-85-K-013.

Authors' addresses: L. Pitt, Department of Computer Science, University of Illinois, Urbana-Champaign, Urbana, IL 61801; L. G. Valiant, Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138.

counterexamples with controllable error could be inferred feasibly. The purpose of the current paper is to indicate limits beyond which learning in this strong sense cannot be expected. We show that for some richer classes of Boolean concepts, feasible learning for all distributions is impossible unless R = NP. The proofs indicate infeasibility for particular distributions only. Learnability for natural subclasses of distributions is not necessarily excluded.

A *concept* is defined in terms of its component *features*, which may themselves be concepts previously learned or primitive sensory inputs. For example, the concept *elephant* may be described in terms of the features {earsize, color, size, has-trunk, ...}. We assume that our domain of discourse consists of n relevant features, which we denote by the variables $x_1, x_2, \ldots, x_n$. Each feature variable x_i may take on a different range of values. Thus the feature *color* may be assigned to any of the values {red, blue, ...}. In this paper we are concerned only with *Boolean* features, however. This restriction is not a limitation, as all negative results for the Boolean case hold for the more general case of multivalued features.

We assume each object in our world is represented by some assignment of the feature variables $\{x_i\}$ to either 0 or 1. Thus each object is simply a vector $\vec{x} \in \{0, 1\}^n$. A *concept* C is a subset of the 2^n possible vectors. If $\vec{x} \in C$, then we say that $\vec{x}$ is an example of the concept, or a *positive example.* Alternatively, if $\vec{x} \notin C$, then we say that $\vec{x}$ is a *negative example.* It is possible and often important to allow incompletely specified feature vectors also [32]. Since the negative results of this paper hold even in the completely specified case, we restrict ourselves to this simpler case here.

The all-zero vector and all-one vector will be denoted by $\vec{0}$ and $\vec{1}$, respectively. We let $\vec{1}_{i,j,\ldots}$ denote the vector that is set to 1 *only* at the positions $i, j, \ldots$. Similarly, $\vec{0}_{i,j,\ldots}$ denotes the vector which is 0 only at the positions $i, j, \ldots$.

There are many different ways in which a concept may be represented. For example, a Boolean formula f over the feature variables $\vec{x}$ represents the concept $C = \{\vec{x} : f(\vec{x}) = 1\}$. Similarly, a Pascal program, circuit, system of linear equations, etc., may be viewed as a representation of a concept.

We are interested in algorithms that are capable of learning from examples whole classes of concepts, since any single concept is trivially learnable by an algorithm that has a "hardcoded" description of that concept. Whether or not a given class of concepts is learnable may depend on the knowledge representation chosen. Hence it is only meaningful to discuss classes of *programs* as being learnable. By a *program* we mean any algorithmic specification with binary output (i.e., *a recognition algorithm*). A program may be an explicit procedure or some knowledge representation with an associated evaluation procedure. A class F of such programs (whether Boolean circuits, formulas, etc.) represents a class $\mathscr{C}$ of concepts if for each $C \in \mathscr{C}$ there is an $f \in F$ that is a recognition algorithm for C.

Let T be a parameter representing the size of the program $f \in F$ to be learned. T will depend on the representation and the total number of available features. For example, for a Boolean formula f over n variables, $T(f) = \max\{n, size(f)\}$, where $size(f)$ is the smallest number of symbols needed to write the formula f.

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

Definition 1.1. Let F be some class of programs representing concepts. Then F is *learnable from examples* iff there exists a polynomial p and a (possibly randomized) learning algorithm A that accesses f only via EXAMPLES(f) such that $(\forall f \in F)(\forall D^+, D^-)(\forall \epsilon > 0)$, the algorithm A halts in time $p(T(f), 1/\epsilon)$ and outputs a program $g \in F$ that with probability at least $1 - \epsilon$, has the following properties:

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

and

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

This definition can be understood as follows: We think of nature as providing examples of the concept to be learned according to some unknown probability distribution for which we can make no assumptions. Since there may be very bizarre examples of the concept that occur with low probability, it is unreasonable to expect the learning algorithm to produce a program that correctly classifies all examples. Hence a successful program g is one that agrees with the unknown rule f on *most* of the distribution. That is, the probability that the program g incorrectly classifies either a positive or a negative example is at most ϵ in either case. A second source of error is introduced by the possibility that the particular sequence of examples provided by nature is highly unrepresentative. In this case it is reasonable that the program g be highly inaccurate. We require that this occur with probability at most ϵ. Hence with probability at least $1 - \epsilon$, g will satisfy the inequalities of Definition 1.1. A further requirement of the definition is that the run time of the algorithm be polynomial in the size of the program to be learned, as well as in the error parameter $1/\epsilon$.

In applications it may be advantageous to allow the learning algorithm to choose a program from a different class from the one from which the input is taken. For example, the underlying rule might be a Boolean formula, but we may be satisfied if the learning algorithm produces a Pascal program that can adequately discriminate between positive and negative examples. The above definition can be extended as follows:

Definition 1.2. Let F and G be classes of programs representing concepts. Then F is *learnable from examples by* G iff there exists a polynomial p and a (possibly randomized) learning algorithm A that accesses f only via EXAMPLES(f) such that $(\forall f \in F)(\forall D^+, D^-)(\forall \epsilon > 0)$, the algorithm A halts in time $p(T(f), 1/\epsilon)$ and outputs a program $g \in G$ that with probability at least $1 - \epsilon$ has the following properties:

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

and

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

Thus "F is learnable by F" is equivalent to "F is learnable." In general, the larger the class F is, the harder the learning task, since the domain of possible programs is much more varied. On the other hand, for fixed F, the larger the class G is, the easier the learning task. Thus we have, immediately from the definitions:

If F is learnable by G, and $G \subseteq H$, then F is learnable by H. In particular, even if F is not learnable, if F is learnable by G and G is learnable, then $F \cup G$ is learnable. Also, it is easy to see that if F is learnable by G, and G is learnable by F, then F is learnable and G is learnable.

In Section 3 we shall see some examples of F and G where F is not learnable, but $F \cup G$ is learnable. Results such as these are reminiscent of the often cited problem-solving heuristic: Try a different viewpoint or richer representation.

We also consider the situation in which, owing to the application area, a classification error in one direction (false positive or false negative) would be catastrophic, and we would like the learning algorithm to produce a rule that completely avoids this type of error. NFP and NFN abbreviate "no false positive" and "no false negative," respectively:

Definition 1.3. Let F be a class of programs representing concepts. Then F is *NFP-learnable* (*respectively NFN-learnable*) *from examples* iff there exists a polynomial p and a (possibly randomized) learning algorithm A that accesses f only via EXAMPLES(f) such that $(\forall f \in F)(\forall D^+, D^-)(\forall \epsilon > 0)$, the algorithm A halts in time $p(T(f), 1/\epsilon)$ and outputs a program $g \in F$ that with probability at least $1 - \epsilon$ has the following properties:

$$\sum_{g(\vec{x})=0} D^+(\vec{x}) < \epsilon \qquad \text{and} \qquad \sum_{g(\vec{x})=1} D^-(\vec{x}) = 0$$

$$\left(\text{respectively, } \sum_{g(\vec{x})=0} D^+(\vec{x}) = 0 \text{ and } \sum_{g(\vec{x})=1} D^-(\vec{x}) < \epsilon\right).$$

We similarly define "*F is NFP-learnable by G*" and "*F*" *is NFN-learnable by G*."

Note that the algorithms of [32] and [33] for k-CNF (k-DNF) are NFP (NFN) in the stronger sense that the probability that $\sum D^-(\vec{x}) = 0$ ($\sum D^+(\vec{x}) = 0$) is 1 rather than $1 - \epsilon$. Here we use this weaker definition, since our negative results (Section 6) hold even for this weaker notion.

Also note that for all of the classes of concepts we consider, and for all classes where testing the consistency of a program with an example can be done in polynomial time, there is a trivial polynomial-time algorithm for learning the class of concepts according to the above definition if it is assumed that P = NP: The algorithm simply takes sufficiently many randomly generated examples from D^+ and D^- and then, using an oracle for NP, tests whether any concept representation of a certain size in the class is consistent with the generated examples. The number of examples required can be deduced easily from [10]. Thus if P = NP, then most reasonable concept classes will be learnable. To prove nonlearnability we must therefore rely on the ubiquitous assumption that P $\neq$ NP. Actually, since our learning algorithms are allowed to be randomized, our results will rely on the assumption that R $\neq$ NP, where R is the class of sets accepted in random polynomial time [19].[1]

One of the main techniques in this paper is to reduce known NP-complete problems to the problem of finding a representation of a concept that is consistent with certain sample data. It is then shown that, if the class of concepts is learnable,

[1] A set S is accepted in random polynomial time iff there exists a randomized algorithm A such that on all inputs A is guaranteed to halt in polynomial time, and such that, if $x \notin S$, $A(x) =$ "no" and if $x \in S$, then $\Pr[A(x) = \text{"yes"}] \geq \frac{1}{2}$.

the NP-complete problem can be solved in random polynomial time. Thus all of our nonlearnability results are of the form; "F is not learnable unless R = NP." For the rest of the paper we drop the phrase "unless R = NP," and simply write "*F is not learnable.*"

The rest of this paper is organized as follows: In Section 2 we briefly review some previous work relative to this model of concept learning. In Sections 3–5 we prove that some restricted classes of Boolean functions are not learnable. We show in Section 6 that even learning of heuristics (rules that account for *some* of the positive examples) may be intractable, and we conclude with a brief review and some open problems in Section 7. In the remainder of this section we define most of the concept classes with which we are concerned.

A CNF formula is a Boolean formula in conjunctive normal form: a product of clauses, where each clause is a sum of literals. A literal is either a variable x_i or its negation $\overline{x_i}$. For example: $(x_1 + \overline{x_7})(x_3 + x_6 + x_9)(\overline{x_2} + \overline{x_4} + x_8)$.

A DNF formula is a Boolean formula in disjunctive normal form: a sum of terms, where each term is a product of literals. For example, $x_1x_3\overline{x_4} + x_2x_9 + \overline{x_6}x_7x_8$.

A monomial is a single term; a product of literals.

For each integer $k \geq 1$ we define the following classes of Boolean formulas (and corresponding concept classes):

k-CNF. The class of CNF formulas with at most k literals in each clause.

k-DNF. The class of DNF formulas with at most k literals in each term.

k-clause-CNF. The class of CNF formulas with at most k clauses.

k-term-DNF. The class of DNF formulas with at most k terms.

μ-formulas. The class of arbitrary Boolean formulas that contain each variable at most once.

monotone formulas. The class of formulas containing only positive literals.

We are also interested in *threshold* operators that are not usually concisely representable as Boolean formulas. For example, a common decision rule might be: It is an elephant iff it has at least 5 of the following 12 features: trunk, big ears, large, grey, ... While we might also allow weights to be given to each feature reflecting its relative importance, it turns out that intractability appears already in the {0, 1} case:

Boolean threshold functions. The class of concepts represented by zero–one linear inequalities: that is, for each concept C there is a vector $\vec{c} \in \{0, 1\}^n$ and an integer $y \geq 0$ such that $C = \{\vec{x}: \vec{c} \cdot \vec{x} \geq y\}$.

2. *Previous Work*

Machine learning has been studied from a number of different vantage points. As a branch of artificial intelligence, it has been investigated widely, although usually in a less formal setting. (See [26] for a survey.) Among more formal approaches, the literature of inductive inference [7, 8] includes investigations into the inference of recursive functions, automata, formal languages, etc. Work has been done on (1) probabilistic aspects of inference [15, 16, 23, 27, 28, 30, 34], (2) approximate inference [11, 13, 29, 31], and (3) inference within the limitations of feasible computation [1–4, 20]. However, what distinguishes the approach taken here is the definition of learnability (introduced in [32]) which, by incorporating all three of the aspects above, allows some nontrivial program classes to be learnable in a strong but plausible sense.

In [32] it was shown that k-CNF is NFP-learnable (and hence learnable) from positive examples only. The main idea was that the formula can be constructed from a conjunct of a set of clauses whose cardinality is at most polynomial in the size of the variable set. Each such clause can be included (with high likelihood) in the conjunct being learned, provided that it does not violate any of a polynomial number of positive samples generated by the underlying distribution. An immediate consequence, by a kind of duality (see Section 3.4), is that k-DNF is NFN-learnable from negative examples only [33]. Since every monomial is a 1-CNF formula, monomials are NFP-learnable from positive examples only. Note that a monomial is a 1-term-DNF expression. We show in the next section that the sum of any other fixed number of monomials is not learnable.

In [33] it is shown that "heuristic monomials," or rules of thumb for DNF expressions to be learned, are difficult to find. We discuss this further and extend the results in Section 6.

Recent work [9, 10] adopts essentially the same definitions of learnability, and applies them to the learning of geometric concepts. It is shown that orthogonal rectangles and other geometric concepts in E^n (n-dimensional Euclidean space) are learnable in time polynomial in n and an error parameter. It is also shown that linear separators in E^n are learnable, which contrasts with our results in Section 5 that Boolean threshold functions (the discrete version of linear separators) are not learnable. It has been shown (N. Megiddo, private communication) that the learnability (in the sense of this paper) of concepts defined by unions of two half spaces in E^n would imply R = NP.

Learning from a "minimally adequate teacher" who can provide counterexamples to hypotheses and answer membership queries is investigated in [6]. It is shown that this model allows the learning of regular sets in polynomial time, whereas results from [1] and [20] showed that this was not possible from examples only unless P = NP.

In [5] and [25] it is shown that finding a minimum simple disjunction (i.e., a 1-CNF formula) consistent with given data is NP-hard. Their proof is by reduction from the Set Covering problem [18]. It is also deduced in [5] that finding a minimum size DNF formula is NP-hard for various size measures (e.g., the number of literals in the formula). The essential reasoning is that simple disjunctions cannot be abbreviated by introducing conjunctions.

Finally, the results from [21] on constructing functions that appear random to polynomial time computations give rise to a concept class that is not learnable assuming the existence of one-way functions.

3. *Hard-to-Learn Boolean Formulas*

In this section we show that k-term-DNF and k-clause-CNF, both in the monotone, and unrestricted case, are not learnable (i.e., unless R = NP). This is perhaps surprising in light of the results in [32] and [33] outlined in Section 2 that k-CNF and k-DNF are learnable. We further discuss the relationships between these problems later.

In order to prove that these classes are not learnable, we first define a particular generalization of the Graph k-Colorability problem [18].

The k-NM-Colorability problem is described (using the notational conventions of [18]) by:

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

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

Note that if every $c \in C$ has size 2, then this is simply the Graph k-Colorability problem, where the vertex set is S and the edge set is the set of pairs in C.

LEMMA 3.1. *For all integers $k \geq 2$, k-NM-Colorability is NP-complete.*

PROOF. Clearly k-NM-Colorability is in NP, and since Graph k-Colorability is NP-complete for every $k \geq 3$ ([18]), we need only show that 2-NM-Colorability is NP-hard. In fact, 2-NM-Colorability is exactly the Set-Splitting problem, which is also NP-complete [18]. □

3.1 LEARNING k-TERM-DNF IS HARD. We are now ready to prove one of our main results. The construction in the proof below will be used again in Sections 4 and 6.

THEOREM 3.2. *For all integers $k \geq 2$, k-term-DNF is not learnable.*

PROOF. We reduce k-NM-Coloring to the k-term-DNF learning problem. Let (S, C) be an instance of k-NM-Coloring. We construct a k-term-DNF learning problem, as follows:

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

If $S = \{s_1, s_2, \ldots, s_n\}$, then we have n feature variables $\{x_1, x_2, \ldots, x_n\}$ for the learning problem. The set of positive examples will be the vectors $\{\vec{p}_i\}_{i=1}^n$, where $\vec{p}_i = \vec{0}_i$, the vector with feature $x_i = 0$ and all other features set to 1. The distribution D^+ will be uniform over these n positive examples, with each $\vec{p}_i$ occurring with probability $1/n$. We'll form $|C|$ negative examples, $\{\vec{n}_i\}_{i=1}^{|C|}$, each occurring with probability $1/|C|$ in the distribution D^-. For each constraint $c_i \in C$, if $c_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_m}\}$, then the vector $\vec{n}_i = \vec{0}_{i_1, i_2, \ldots, i_m}$ is a negative example. Thus the constraint $\{s_1, s_3, s_8\}$ gives rise to the negative example $\vec{0}_{1,3,8} = \langle 01011110111 \cdots \rangle$.

CLAIM 3.3. *There is a k-term-DNF formula consistent with all of the positive and negative examples above iff (S, C) is k-NM-Colorable.*

PROOF OF CLAIM

($\Leftarrow$) Assume (S, C) is k-NM-Colorable by a coloring χ: $S \rightarrow \{1, 2, \ldots, k\}$ that uses every color at least once. Let f be the k-term-DNF expression $f = T_1 + T_2 + \cdots + T_k$, where the ith term T_i is defined by

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

In other words, the ith term is the conjunction of all variables x_j for which the corresponding element s_j is not colored i.

Then the positive example $\vec{p}_j$ clearly satisfies the term T_i, where $\chi(s_j) = i$. Thus all of the positive examples satisfy the formula f.

Now suppose that some negative example $\vec{n}_i = \vec{0}_{i_1, i_2, \ldots, i_m}$ satisfies f. Then $\vec{n}_i$ satisfies some term T_j. Then every element of $\{s_{i_1}, s_{i_2}, \ldots, s_{i_m}\}$ must be colored with color j; otherwise, T_j would not be satisfied. But then the constraint c_i associated with the negative example $\vec{n}_i$ is not satisfied by the coloring, since all of the members of c_i are colored the same color. Thus no negative example satisfies the formula f.

($\Rightarrow$) Suppose that $T_1 + T_2 + \cdots + T_k$ is a k-term-DNF formula that is satisfied by all of the positive examples and no negative example. Then note without loss of generality, each T_i is a product of positive literals only: If T_i contains two or more negated variables, then none of the positive examples can satisfy it (since they all have only a single "0"), so it may be eliminated. If T_i contains exactly one negative literal $\overline{x_j}$, then it can be satisfied by at most the single positive example $\vec{p}_j$, so T_i can be replaced with $T_i' = \prod_{j \neq i} x_j$, which is satisfied by only the vectors $\vec{p}_j$ and the vector $\vec{1}$, neither of which are negative examples.

Now color the elements of S by the function χ: $S \rightarrow \{1, 2, \ldots, k\}$ defined by $\chi(s_i) = \min\{j: \text{literal } x_i \text{ does not occur in term } T_j\}$.

Note that χ is well defined: Since each positive example satisfies the formula $T_1 + T_2 + \cdots + T_k$, each positive example $\vec{p}_i$ must satisfy some term T_j. But each term is a conjunct of only positive literals; therefore, for some j, x_i must not occur in term T_j. Thus each element of S receives a color. Furthermore, if χ violates some color constraint $c_i = \{s_{i_1}, s_{i_2}, \ldots\}$, then all of these elements are colored by the same color j, and then, by the definition of χ, none of the literals $\{x_{i_1}, x_{i_2}, \ldots\}$ occur in term T_j, and thus the negative example associated with c_i satisfies T_j, contradicting the assumption that none of the negative examples satisfy the formula $T_1 + T_2 + \cdots + T_k$. This completes the proof of the claim. □

Now to complete the proof of Theorem 3.2, we observe that, if there is a learning algorithm for k-term-DNF, it can be used to decide k-NM-Colorability in random polynomial time as follows: Given an instance (S, C) of k-NM-Colorability, form the distributions D^+ and D^- as above, and choose $\epsilon < \min\{1/|S|, 1/|C|\}$.

If (S, C) is k-NM-Colorable, then by Claim 3.3 there exists a k-term-DNF formula consistent with the positive and negative examples, and thus, with probability at least $1 - \epsilon$, the learning algorithm must produce a formula that is consistent with *all* of the examples (by the choice of the error bound ϵ), thereby giving a k-NM-coloring for (S, C). Conversely, if (S, C) is not k-NM-Colorable, then by Claim 3.3 there does not exist a consistent k-term-DNF formula, and the learning algorithm must either fail to produce a hypothesis within its allotted time, or else produce one that is not consistent with at least one example. In either case this can be observed, and it may be determined that no k-NM-Coloring is possible. □

Note that the construction gives rise to a *monotone* k-term-DNF formula that is hard to learn, even when the learning algorithm is allowed to choose among nonmonotone k-term-DNF formulas. Thus we have the following stronger result.

COROLLARY 3.4

— *For all integers $k \geq 2$, monotone k-term-DNF is not learnable.*
— *For all integers $k \geq 2$, monotone k-term-DNF is not learnable by k-term-DNF.*

3.2 k-TERM-DNF AND GRAPH COLORING APPROXIMATIONS.

As shown, k-term-DNF, even in the simple monotone case, is not learnable because it is NP-hard to determine whether there is a k-term-DNF formula consistent with a collection of positive and negative examples. We are prompted to ask whether k-term-DNF (of n variables) is learnable by $(k + 1)$-term-DNF or $f(k, n)$-term-DNF for some reasonably slowly growing function f.

The difficulty of learning k-term-DNF stems from the NP-hardness of a generalization of the Graph k-Colorability problem. In fact, we could have used Graph k-Colorability directly to obtain the same result for $k \geq 3$. (However, we shall need the NP-hardness of 2-NM-Colorability in a later section also.)

It is shown in [17] that for every $\epsilon > 0$, unless P = NP, no polynomial time algorithm can approximate the fewest number of colors needed to color a graph within a constant factor of $2 - \epsilon$. In the paper, we see that this is achieved by showing that for all $k \geq 6$, unless P = NP, no polynomial time algorithm exists that outputs "yes" on input a k-colorable graph, and outputs "no" on input a graph that needs at least $2k - 4$ colors. It follows immediately from the proof of Theorem 3.2 and this result that

COROLLARY 3.5. *For all integers $k \geq 6$, (monotone) k-term-DNF is not learnable by $(2k - 5)$-term-DNF.*

We can strengthen this result by exploiting the fact that 2-NM-Colorability is NP-hard. In [17] the hardness of Graph 3-Colorability was used to obtain the result mentioned above. By employing the same techniques, but using 2-NM-Colorability, it is easy to show that

COROLLARY 3.6. *For all integers $k \geq 4$, (monotone) k-term-DNF is not learnable by $(2k - 3)$-term DNF.*

There is currently a large separation between the lower bounds for approximate coloring, and the upper bounds achieved by the best approximate coloring algorithms. Even for 3-colorable graphs, the best approximation algorithm known only guarantees that the coloring found uses at most $3\sqrt{n}$ colors, where n is the number of vertices in the graph [35]. Should this prove to be a lower bound as well, we would have that (monotone) 3-term-DNF is not learnable by less than $3\sqrt{n}$-term-DNF (with n the number of variables).

On the other hand, suppose that for some $\epsilon > 0$, $g(n)$ is $o(n^{1-\epsilon})$. Then, if we show that monotone k-term-DNF is learnable by $kg(n)$-term-DNF, we would immediately have significantly improved the best known upper bound [24, 35] of $\min\{k(n/\log n), kn^{1-(1/k)}\}$ for the number of colors needed by randomized algorithms to color a k-Colorable graph with n vertices. It is unlikely that such an algorithm will be found easily. The reader should consult [9] for further relationships between approximations for NP-hard optimization problems and learnability.

3.3 k-TERM-DNF IS LEARNABLE BY k-CNF.

The previous section indicates the difficulties involved in learning k-term-DNF. Following an observation of R. Boppana (private communication), we show here that k-term-DNF is learnable by k-CNF. (As a corollary we have that k-term-DNF $\cup$ k-CNF *is* learnable!) There is no paradox here. k-term-DNF is too restrictive of a domain to allow learning, that is, although patterns in the data may be observable, the demand that the learned formula be expressed in k-term-DNF is a significant enough constraint to render the task intractable. A richer domain of representation, k-CNF, allows a greater latitude in expressing the formula learned. Thus the availability to the learner of a variety of knowledge representations is seen to be valuable for learning. Often a change of representation can make a difficult learning task easy.

THEOREM 3.7. *For all integers $k \geq 1$, k-term-DNF is learnable by k-CNF.*

PROOF. Every k-term-DNF formula F_D is logically equivalent to some k-CNF formula F_C whose size is polynomially related to the size of F_D. (If $F_D = \sum_{i=1}^{k} T_i$, then let $F_C = \prod (x_{i_1}, x_{i_2}, \ldots, x_{i_k})$, where the product is over all choices of $x_{i_1} \in T_1$, $x_{i_2} \in T_2, \ldots, x_{i_k} \in T_k$.) Use the algorithm of [32] to learn a k-CNF formula F that is close enough to F_C (according to D^+ and D^-) to satisfy the definition of learnability. □

3.4 DUALITY: k-CLAUSE-CNF. Just as similar techniques showed that both k-CNF and k-DNF were learnable [32, 33], we observe that essentially the same proof shows that both k-term-DNF and k-clause-CNF are not learnable for $k \geq 2$. This follows because k-clause-CNF learnability is easily seen to imply learnability for k-term-DNF.

Assume that k-clause-CNF is learnable by algorithm A. Suppose we are given an instance of k-term-DNF learning, that is, an EXAMPLE box with buttons POS and NEG with underlying distributions D^+_{DNF} and D^-_{DNF}. Now transform the box by switching the labels of POS and NEG, and present to algorithm A this new example box (with distributions $D^+_{\text{CNF}} = D^-_{\text{DNF}}$ and $\text{D}^-_{\text{CNF}} = D^+_{\text{DNF}}$).

In polynomial time, A constructs a k-clause-CNF formula $C = C_1C_2C_3 \cdots C_k$ where each C_i is a sum of literals. Now let the k-term-DNF formula F be defined by

$$F = \overline{C_1C_2 \cdots C_k} = \overline{C_1} + \overline{C_2} + \cdots \overline{C_k}.$$

We have immediately that

$$\sum_{F(\vec{x})=0} D^+_{\text{DNF}}(\vec{x}) = \sum_{C(\vec{x})=1} D^-_{\text{CNF}}(\vec{x}) < \epsilon$$

and

$$\sum_{F(\vec{x})=1} D^-_{\text{DNF}}(\vec{x}) = \sum_{C(\vec{x})=0} D^+_{\text{CNF}}(\vec{x}) < \epsilon;$$

thus we have constructed a k-term-DNF formula for the original input in polynomial time.

We have just proved

THEOREM 3.8. *For all integers $k \geq 2$, k-clause-CNF is not learnable.*

By the same reduction, and by Corollary 3.6, we also have the following corollary:

COROLLARY 3.9

— *For all integers $k \geq 4$, k-clause-CNF is not learnable by $(2k - 3)$-clause-CNF.*
— *For all positive integers k, k-clause-CNF is learnable by k-DNF.*
— *For all positive integers k, k-clause-CNF ∪ k-CNF ∪ k-term-DNF ∪ k-DNF is learnable.*

4. *μ-Formulas Are Not Learnable*

In [32], it was shown that the class of μ-formulas (Boolean formulas in which each variable occurs at most once) are learnable, provided that the learning algorithm has available certain very powerful oracles. Here we show that μ-formulas are not learnable from examples alone. In Section 6 we show that it is difficult to learn even very weak approximations of μ-formulas.

Each μ-formula may be represented in a natural way as a rooted directed tree with variables as leaves, internal nodes labeled with AND, OR, and NOT, and edges directed away from the root. We *do not* assume that the out-degree of these nodes is two, and therefore we may assume that no node has the same label as its immediate ancestor. Observe that all of the NOT labels may be pushed to the leaves of the tree by using De Morgan's laws. This operation can be carried out without increasing the number of edges and may increase the number of nodes by at most n, the number of leaves. We therefore assume without loss of generality that the internal nodes of the tree are all labeled with AND or OR with no node labeled the same as its immediate ancestor, that they have out-degree at least 2, and that the leaves contain literals for the variables. As each variable that occurs in the formula may occur only once, in the tree there will be a single node containing either the variable, or its negation. We say that the variable x_i occurs positively (respectively, negatively) if x_i (respectively, $\overline{x_i}$) is a leaf. It is easy to see that in polynomial time such a tree representation can be constructed from any μ-formula.

We reduce 2-NM-Colorability to the μ-formula learning problem, but first we show that without loss of generality, the 2-NM-Colorability problem satisfies a simple property. Suppose that (S, C) is an instance of 2-NM-Colorability. Let $s \in S$ be given. If there is some $t \in S$ such that $\{s, t\} \in C$, then we say that s has t as a partner. (s may have many partners.) Now notice that without loss of generality we may assume that every $s \in S$ has a partner, that is, that every element of S occurs in some constraint of size 2 enforcing that the two elements be colored differently. For if (S, C) does not satisfy this property, we can simply form the new instance $(S \cup S', C \cup C')$, where $S' = \{s' : s \in S\}$ and $C' = \{\{s, s'\} : s \in S\}$; thus we have added a partner s' for each element s of S, and the partner occurs only in the single constraint $\{s, s'\}$. Now $(S \cup S', C \cup C')$ is 2-NM-Colorable iff (S, C) is 2-NM-Colorable, and every element has a partner.

Now we are ready to prove

THEOREM 4.1. *μ-formulas are not learnable.*

PROOF. We reduce 2-NM-Colorability to μ-formula learning. Let (S, C) be an instance of 2-NM-Colorability. By the remarks above, we may assume that every $s \in S$ has a partner. Let the positive examples be $\{\vec{p}_i\}$ and the negative examples be $\{\vec{n}_i\}$ as in the proof of Theorem 3.2, with the uniform distributions D^+ and D^-.

LEMMA 4.2. *There is a μ-formula consistent with all $\{\vec{p}_i\}$ and $\{\vec{n}_i\}$ iff (S, C) is 2-NM-Colorable.*

The nonlearnability of μ-formulas follows from Lemma 4.2, for as in the proof of Theorem 3.2, if there is an algorithm for learning μ-formulas in random polynomial time, then, by choosing error parameter ϵ smaller than both $1/|C|$ and $1/|S|$, the algorithm can be used to decide the NP-complete 2-NM-Colorability problem in random polynomial time.

We prove Lemma 4.2

($\Leftarrow$) By Claim 3.3, (S, C) is 2-NM-Colorable implies there is a 2-term-DNF formula consistent with all of $\{\vec{p}_i\}$ and $\{\vec{n}_i\}$. By that construction, the 2-term-DNF formula contains each variable at most once; therefore, it is a μ-formula.

($\Rightarrow$) Suppose there is some μ-formula f consistent with the examples $\{\vec{p}_i\}$ and $\{\vec{n}_i\}$. Now consider the tree T for f that has only AND and OR internal nodes, as discussed above.

CLAIM. *Each variable x_i occurs positively in the tree T.*

Suppose that x_i either occurs negatively in T, or fails to occur. Then consider the element s_i of S with which x_i is associated. s_i has a partner s_j. Since every positive example satisfies f, the positive example $\vec{p}_j$ satisfies f. If x_i fails to occur in T, then it does not matter what value it has; so if $\vec{p}_j = \vec{0}_j$ satisfies f, so too does the vector $\vec{0}_{i,j}$, which is a negative example due to the constraint $\{s_i, s_j\} \in C$. Similarly, if x_i occurs negatively in f, then since $\vec{p}_j$ satisfies f, so too will the same vector with x_i set to 0 instead of 1, and thus the negative example $\vec{0}_{i,j}$ satisfies f. Therefore, every variable occurs positively in f, proving the claim.

Now there are two cases to consider, depending on the label of the root of the tree. In each case we show how a 2-NM-Coloring may be found from T.

Case 1. The root node is labeled OR.

Then f is equivalent to the formula $f_1 + f_2 + \cdots + f_k$, where f_i is the subformula computed by the subtree of f with f's ith child as root.

Let $L = f_1$ and $R = f_2 + \cdots + f_k$. Color each element s_i with color C_L iff x_i occurs in formula L; otherwise, color s_i with color C_R. We show that this is a legitimate 2-NM-Coloring:

Suppose a constraint $c_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_m}\}$ is violated. Then all of $s_{i_1}, s_{i_2}, \ldots, s_{i_m}$ are colored the same color, and all of $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$ occur in the same subformula, say L without loss of generality. Then since formula R contains only positive literals, and does not contain the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$, it follows that $\vec{n}_i = \vec{0}_{i_1, i_2, \ldots, i_m}$ satisfies formula R, and therefore satisfies f, a contradiction.

Case 2. The root node is labeled AND.

Since each variable x_i occurs positively, there must be an OR on the path from the root to each x_i; otherwise the positive example $\vec{p}_i = \vec{0}_i$ could not satisfy f. Thus there are $k \geq 2$ OR nodes that are children of the root AND node (by the normal form assumption that all nodes have out-degree at least 2). We divide the subtree beneath the ith OR into two groups, L_i and R_i, where L_i is the function computed by the leftmost subtree of the ith OR, and R_i is the function computed by the OR of the remaining branches of the ith OR.

Thus $f = (L_1 + R_1)(L_2 + R_2) \cdots (L_k + R_k)$. Then let $f' = L + R$, where $L = L_1 L_2 \cdots L_k$ and $R = R_1 R_2 \cdots R_k$. We have that if f' is satisfied by some vector, then f is also. Therefore no negative example satisfies f'. Now color s_i with color C_L iff x_i occurs in formula L, and color it C_R, otherwise. By the same argument as in Case 1, if some coloring constraint is violated, then all of the elements of the constraint occur in the same subformula, and then the other subformula is satisfied by the negative example associated with the given constraint. This completes the proof of Lemma 4.2 and Theorem 4.1. □

5. *Boolean Threshold Functions*

A Boolean threshold function (defined in Section 1) may be thought of intuitively as follows. Among the set of n features $\{x_i\}$ there is some *important subset* Y for the concept to be learned. There is also a critical threshold k such that whenever an example $\vec{x}$ has at least k of the features of Y set to 1, it is a positive example; otherwise, it is a negative example. We write this rule as $\mathrm{Th}_k(\vec{y})$, where $\vec{y}$ is the characteristic vector for the set Y, that is, $y_i = 1$ iff the ith feature is in the set Y.

Thus if $\vec{x}$ is a positive example, then it satisfies $\vec{x} \cdot \vec{y} \geq k$, and if $\vec{x}$ is a negative example, then it satisfies $\vec{x} \cdot \vec{y} < k$. (Where $\cdot$ is the "dot product" of the two vectors: $\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i y_i$.)

To show that Boolean threshold functions are not learnable, we reduce Zero-One Integer Programming (ZIP) to the learning problem. ZIP is the following NP-complete problem [18, p. 245]:

Instance: A set of s pairs $\vec{c}_i$, b_i and the pair $\vec{a}$, B, where $\vec{c}_i \in \{0, 1\}^n$, $\vec{a} \in \{0, 1\}^n$, $b_i \in [0, 1\}$, and $0 \leq B \leq n$.
Question: Does there exist a vector $\vec{d} \in \{0, 1\}^n$ *such that* $\vec{c}_i \cdot \vec{d} \leq b_i$ for $1 \leq i \leq s$ and $\vec{a} \cdot \vec{d} \geq B$?

Given an instance of ZIP, we construct a Boolean threshold learning problem. We have $2n$ features $x_1, x_2, \ldots, x_{2n}$. We sometimes write a vector of length $2n$ as the concatenation of two vectors $\vec{u}$, $\vec{v}$ of length n, and denote this by $(\vec{u}, \vec{v})$.

There are two positive examples, $\vec{p}_1 = (\vec{0}, \vec{1})$ and $\vec{p}_2 = (\vec{a}, \vec{1}_{1,2,\ldots,n-B})$.

There are two types of negative examples. First, for each of the vectors $\vec{c}_i$, $1 \leq i \leq s$, from the ZIP instance, we define the negative example $(\vec{c}_i, \vec{1}_{1,2,\ldots,n-b_i-1})$. Second, for $1 \leq i \leq n$ we define the negative example $\vec{0}, \vec{0}_i)$.

We claim that there is a solution to the ZIP instance iff there is a Boolean threshold function consistent with the given examples. If our claim is true, then any learning algorithm can be used to decide the ZIP problem in random polynomial time by letting D^+ and D^- be uniform over the positive and negative examples, respectively, and choosing $\epsilon < 1/(s + n)$.

To prove the claim, suppose that z is a solution to the ZIP instance, and let $\vec{y} = (\vec{z}, \vec{1})$. It is easily verified that the threshold function $\text{Th}_n(\vec{y})$ is consistent with all of the positive and negative examples above.

On the other hand, suppose that Y is a set with characteristic vector $\vec{y} = (\vec{z}, \vec{w})$, and k is a positive integer such that the rule $\text{Th}_k(\vec{y})$ is consistent with the positive and negative examples defined above. We show that $\vec{z}$ is a solution to the ZIP instance.

The facts that $\vec{p}_1 = (\vec{0}, \vec{1})$ is a positive example and that, for all i, $(\vec{0}, \vec{0}_i)$ is a negative example give rise (respectively) to the following two inequalities:

$$k \leq (\vec{0}, \vec{1}) \cdot (\vec{z}, \vec{w}) \leq n. \tag{1}$$

$$(\forall i) \qquad (\vec{0}, \vec{0}_i) \cdot (\vec{z}, \vec{w}) < k. \tag{2}$$

By (1) and (2), and the fact that $(\vec{0}, \vec{0}_i)$ differs from $(\vec{0}, \vec{1})$ in only the position $n + i$, it follows that $\vec{w} = \vec{1}$. Substituting $\vec{1}$ for $\vec{w}$ in (2), we conclude that $k > n - 1$, and by (1), $k = n$.

Now observe that since $\vec{p}_2 = (\vec{a}, \vec{1}_{1,2,\ldots,n-B})$ is a positive example,

$$n \leq (\vec{a}, \vec{1}_{1,2,\ldots,n-B}) \cdot (\vec{z}, \vec{1})$$

and thus

$$\vec{a} \cdot \vec{z} \geq B. \tag{3}$$

Also, since for each i, $(\vec{c}_i, \vec{1}_{1,2,\ldots,n-b_i-1})$ is a negative example,

$$(\vec{c}_i, \vec{1}_{i,2,\ldots,n-b_i-1}) \cdot (\vec{z}, \vec{1}) < n,$$

and hence

$$(\forall i) \qquad \vec{c}_i \cdot \vec{z} \leq b_i. \tag{4}$$

Inequalities (3) and (4) assert that $\vec{z}$ is indeed a solution to the ZIP instance, proving our claim, and the following.

THEOREM 5.1. *Boolean threshold functions are not learnable.*

6. *Learning Heuristic Rules*

As we have seen in the previous sections, there are various natural classes of formulas for which the problem of finding a good rule is hard. In cases such as these, as well as cases in which there may be no rule of the form we seek that is consistent with the observed data, we may wonder whether we can learn *heuristic* rules for the concept—rules that account for some significant fraction of the positive examples, while avoiding incorrectly classifying most of the negative examples.

For example, since we know of no learning algorithm for CNF and DNF in general, and we do have a learning algorithm for 1-term-DNF, perhaps we can find a single monomial that covers half of the positive examples while avoiding error on all but $1 - \epsilon$ of the negative examples whenever such a monomial exists.

The following definition is meant to capture the notion of learning heuristics:

Definition 6.1. Let h be a function of n, the number of features, such that $(\forall n)\ 0 \le h(n) \le 1$, and let F and G be classes of programs representing concepts. Then F is *h-heuristically learnable from examples by G* iff there exists a polynomial p and a (possibly randomized) learning algorithm A that accesses f only via EXAMPLES(f) such that $(\forall f \in F)(\forall n)(\forall D^+, D^-)(\forall \epsilon > 0)$, the algorithm A halts in time $p(T(f), 1/\epsilon)$ and outputs a program $g \in G$ that with probability at least $1 - \epsilon$, has the following properties:

$$\sum_{g(\vec{x})=0} D^+(\vec{x}) < 1 - h(n)$$

and

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

If no $g \in G$ with these properties exists, then A may output anything.

Note that the program g found must be correct on the fraction $h(n)$ of the positive examples. If $F = G$, then we write "F is h-heuristically learnable," and observe that the last sentence of the definition is redundant. The definition extends in the natural way to "F is h-heuristically NFP-learnable by G," and "F is h-heuristically NFP-learnable," by modifying the second inequality to $\sum_{g(\vec{x})=1} D^-(\vec{x}) = 0$.

6.1 HEURISTIC MONOMIALS ARE NOT LEARNABLE.

In [33], it was shown that the class DNF is not $\frac{1}{2}$-heuristically NFP-learnable by monomials; that is, it is NP-hard to determine whether there is a monomial m such that at least half of the positive examples (weighted by D^+) and none of the negative examples satisfy m, given that there is some DNF rule that accurately classifies all of the examples. (Actually, the results in [33] also imply that DNF is not c-heuristically NFP-learnable by monomials for any fixed rational $0 < c < 1$.)

Here we slightly strengthen the result by removing the "NFP" condition and show that DNF is not c-heuristically learnable by monomials.

The reduction is from the Independent Set (IS) problem [18]:

Instance: Graph G with vertex set V and edge set E and integer k.

Question: Does there exist an Independent Set of G of size at least k, that is, a subset $V' \subseteq V$ such that $|V'| \geq k$ and for no pair of elements $x, y \in V'$ is it the case that $(x, y) \in E$?

A simple reduction shows that for any rational $0 < c < 1$, the following c-IS problem is also NP-complete:

Instance: Graph G with vertex set V and edge set E.

Question: Does there exist an Independent Set of G of size at least $c\,|V|$?

We reduce c-IS to the problem of learning c-heuristic monomials for DNF. Let $G = (V, E)$ be an instance of c-IS. With each vertex v_i associate a feature x_i. Let the set of positive examples be $\{\vec{0}_i\}_{i=1}^{n}$. For each edge $(i, j) \in E$ define the negative example $\vec{0}_{i,j}$. Let the distributions D^+ and D^- be uniform over these positive and negative examples, respectively. Let $\epsilon < 1/|E|$.

Clearly there is a DNF formula for this concept: the n-term formula such that the ith term is satisfied only by the vector $\vec{0}_i$.

If there is a learning algorithm A that could determine whether there exists (and produce) a monomial m that is satisfied by the fraction c of the positive examples, and by at most ϵ of the negative examples, then m is satisfied by *none* of the negative examples, by choice of ϵ.

Without loss of generality, m contains only positive literals; otherwise at most one positive example could satisfy m. Now observe that the set of positive examples that satisfy m are those vectors $\vec{0}_i$ such that the literal x_i does not occur in m. Furthermore, if both the positive examples $\vec{0}_i$ and $\vec{0}_j$ satisfy m, then $\vec{0}_{i,j}$ also satisfies m and is therefore not a negative example; that is, $(i, j) \notin E$.

Thus $\{v_i : x_i \text{ does not occur in } m\}$ is an independent set of V and is of size at least $cn = c\,|V|$. Conversely, if there is an independent set I of V of size $c\,|V|$, then it is easily verified that the monomial.

$$\prod_{v_i \notin I} x_i$$

is satisfied by at least the fraction c of the positive examples, and none of the negative examples. Therefore, learning algorithm A solves the c-IS problem in random polynomial time. Furthermore, note that the construction was from a monotone DNF formula; so we have proved

THEOREM 6.1. *Monotone (and hence unrestricted) DNF is not c-heuristically learnable by monomials for any rational c with* $0 < c < 1$.

To further illustrate the difficulty of finding heuristic monomials, let *opt* be the largest number of positive examples that satisfy a monomial that is not satisfied by more than ϵ of the negative examples. Then for any constant $c < 1$, the problem of learning a monomial that is satisfied by $c \cdot opt$ positive examples and at most ϵ of the negative examples is as hard as approximating the maximum independent set to within a constant factor c. It has been shown [18] that if this can be done, then the size of the maximum independent set can be approximated arbitrarily closely.

Finally, we note that a straightforward reduction may be employed to show that for any fixed positive integer k and rational c such that $0 < c < 1$, DNF is not c-heuristically learnable by k-term-DNF.

6.2 Heuristic μ-Formulas Are Not NFP-Learnable.

We now prove a rather strong lower bound for heuristic NFP-learnability of μ-formulas. We show

THEOREM 6.2. *μ-formulas (of N variables) are not $\exp(-N^{1/3})$-heuristically NFP-learnable.*

In other words, if the formula sought is not to be satisfied by any negative examples, then it is NP-hard to find a μ-formula that is satisfied by even an exponentially vanishing fraction of the positive examples (when in fact there is some μ-formula that is consistent with all of the positive and negative examples).

The proof of Theorem 6.2 will be very similar to that of Theorem 4.1. Given a 2-NM-Colorability instance (S, C), we show how to construct a μ-formula learning problem (a set of positive and negative examples with distributions D^+ and D^-) such that if a learning algorithm A produces a μ-formula that is satisfied by no negative example, and at least $\exp(-N^{1/3})$ of the positive examples (according to the distribution D^+), then this formula may be used to find (in polynomial time) a 2-NM-Coloring of (S, C).

PROOF. Let (S, C) be an instance of 2-NM-Colorability, with $|S| = n$, and recall that we may assume without loss of generality that every $s \in S$ has a partner. Then we define a learning problem with $N = n^3$ feature variables, which we think of as the concatenation of n^2 groups of n feature variables each. Each example vector x has length n^3, and we write it as the concatenation of n^2 vectors $\{\vec{b}_i\}$ each of length n. Thus $\vec{x} = (\vec{b}_1\ \vec{b}_2, \ldots, \vec{b}_{n^2})$. We call each of these subvectors of length n a *block*. The feature variable $x_{i,j}$ is the jth feature of the ith block, thus $\vec{b}_i = x_{i,1}x_{i,2}, \ldots, x_{i,n}$ for each i, $1 \le i \le n^2$. Whereas in our previous reduction, the element s_j had associated with it the feature variable x_j, here it is associated with the jth element of *each* block, that is, with each element of $\{x_{i,j}: 1 \le i \le n^2\}$.

We define n^{n^2} positive examples: Every vector $\vec{x}$ for which $(\forall i)(\exists j)\ \vec{b}_i = \vec{0}_j$. Thus each block of any positive example looks like some positive example from the reduction of Theorem 3.2.

We define many negative examples: Any vector where for some block i we have that $\vec{b}_i = \vec{n}_j$, where $\vec{n}_j$ is defined as in the reduction for Theorem 3.2—all 1's except at the positions corresponding to the elements of the jth constraint $c_j \in C$.

Let D^+ be the uniform distribution on the positive examples, generated by choosing, independently at random, a single position in each block to be set to 0. D^- is an almost uniform distribution, generated by randomly choosing a block i, randomly choosing a constraint c_j, setting $\vec{b}_i = \vec{n}_j$, and then setting each of the other blocks randomly.

Note that the distributions D^+ and D^- are polynomially generable, that is, there is an algorithm that produces examples for each of these distributions in random polynomial time.

LEMMA 6.3. *If there is a 2-NM-Coloring of (S, C) then there is a μ-formula f that is satisfied by* all *of the positive examples, and* none *of the negative examples.*

PROOF. If $\chi: S \to \{1, 2\}$ is a 2-NM-Coloring of (S, C), then define the μ-formula f by

$$f = \prod_{i=1}^{n^2} (T_{i,1} + T_{i,2}),$$

where

$$T_{i,1} = \prod_{\chi(s_j)\neq 1} x_{i,j} \quad \text{and} \quad T_{i,2} = \prod_{\chi(s_j)\neq 2} x_{i,j}.$$

f is a μ-formula, because the variable $x_{i,j}$ may only appear in either the subformula $T_{i,1}$ or the subformula $T_{i,2}$. If it occurs in $T_{i,1}$ then $\chi(s_j) \neq 1$, and if it appears in $T_{i,2}$ then $\chi(s_j) \neq 2$. It then cannot appear in both subformulas; otherwise, the element s_j receives no color by the coloring χ.

Suppose that $\vec{p}$ is a positive example, and thus for some function $p: \{1, 2, \ldots, n^2\} \to \{1, 2, \ldots, n\}$ the ith block of $\vec{p}$ is exactly the vector $\vec{0}_{p(i)}$. Then for each i, $\vec{p}$ satisfies at least one of $\{T_{i,1}, T_{i,2}\}$: If $\chi(s_{p(i)}) = 1 \neq 2$, then the literal $x_{i,p(i)}$ appears in $T_{i,2}$ and therefore not in $T_{i,1}$, and since all other variables of the form $x_{i,j}, j \neq p(i)$ are set to 1 in $\vec{p}$, and these are the only other variables that can occur in $T_{i,1}$, we have that $\vec{p}$ satisfies $T_{i,1}$. Similarly, if $\chi(s_{p(i)}) = 2$, then $\vec{p}$ satisfies $T_{i,2}$. Since $\vec{p}$ satisfies either $T_{i,1}$ or $T_{i,2}$ for each i, $\vec{p}$ satisfies f.

Now if $\vec{n}$ is a negative example, then by definition, there is some block $\vec{b}_i$ and constraint $c_j = \{s_{j_1}, s_{j_2}, \ldots, s_{j_k}\}$ such that $\vec{b}_i = \vec{0}_{j_1,j_2,\ldots,j_k}$. If $\vec{n}$ satisfies f, then $\vec{n}$ satisfies $T_{i,1}$ or $T_{i,2}$. By definition, we have that either all of the literals $\{x_{i,j_1}, x_{i,j_2}, \ldots, x_{i,j_k}\}$ occur in $T_{i,1}$ or they all occur in $T_{i,2}$. Then we must have that all of the elements $\{s_{j_1}, s_{j_2}, \ldots, s_{j_k}\}$ are colored with color 2 or 1, respectively, and thus χ does not satisfy all of the constraints. Hence, $\vec{n}$ cannot satisfy f. This completes the proof of Lemma 6.3.

LEMMA 6.4. *From a μ-formula f that is satisfied by more than $(n-1)^{n^2}$ of the positive examples, and not by any negative examples, in polynomial time a 2-NM-Coloring of (S, C) can be found.*

PROOF. We say that block i is *covered* iff for each j, $1 \leq j \leq n$, there is some positive example $\vec{p}$ such that the ith block $\vec{b}_i = \vec{0}_j$, and $\vec{p}$ satisfies f. Clearly there must be some block i that is covered; otherwise, f is satisfied by at most $(n-1)^{n^2}$ positive examples. Let i be a covered block.

CLAIM. *In the tree T for the function f, all variables $\{x_{i1}, x_{i,2}, \ldots, x_{i,n}\}$ from the covered ith block occur positively.*

To see that the claim is true, recall that each element s_j has a partner s_k in the 2-NM-Colorability instance. With $\{s_j, s_k\}$ as a constraint, we have that *any* vector for which the ith block $\vec{b}_i = \vec{0}_{j,k}$ is a negative example. Since there is some positive example $\vec{p}$ satisfying f such that the ith block $\vec{b}_i = \vec{0}_j$ (block i is covered), if $x_{i,k}$ occurs negatively in the tree, or not at all, then by simply setting $x_{i,k}$ to 0 in $\vec{p}$, we have an example that still satisfies f, and in which $\vec{b}_i = \vec{0}_{j,k}$, a negative example. This contradicts the assumption that no negative example satisfies f.

The rest of the proof of Lemma 6.4 now follows similarly to that of Lemma 4.2. We sketch the argument.

Case 1. The root node is labeled OR.

Then $f = L + R$, where L is the function computed by the left branch of the root node, and R is the function computed by the OR of the remaining branches of the root node. Consider only the variables $\{x_{i,j}\}$ from the covered block i. Color s_j with color C_L iff the literal $x_{i,j}$ appears in formula L; otherwise, color s_j with color C_R.

If there is some constraint $c_j = \{s_{j_1}, s_{j2}, \ldots, s_{j_k}\}$ whose elements are all colored identically (without loss of generality suppose it is C_L), then all of the literals $x_{i,j_1}, x_{i,j_2}, \ldots, x_{i,j_k}$ occur in formula L. Then these variables do not occur in R, and therefore the vector $\vec{n}$ that has these variables set to 0, all other variables from block i set to 1, and all other variables set to 0 or 1 depending on whether they occur negatively or positively in f must satisfy R (and hence f). But $\vec{n}$ is a negative example, since the ith block corresponds to a coloring constraint. This contradicts the assumption that no negative examples satisfy f.

Case 2. The root node is labeled AND.

First note that none of the children of the root AND node in the formula tree T can be literals from block i. For if $x_{i,j}$ is a child of the root, then any positive example satisfying the formula must have a "1" in position $x_{i,j}$, and there is then no positive example satisfying f such that the ith block $\vec{b}_i = \vec{0}_j$, contradicting the fact that block i is covered.

Although there may be some literals from some block other than i attached directly to the AND, the path from the AND to any literal of block i must contain an OR node. Thus there is at least one OR as an immediate descendant of the top AND. Suppose there are $k \geq 1$ ORs that are immediate descendants of the top AND and whose subtree contain some literal from block i. For each such OR, we divide the subtrees below the OR into two groups, L_j and R_j, $(1 \leq j \leq k)$, where L_j is the function computed by the subtree beneath the leftmost branch of the jth OR, and R_j is the function computed by the OR of the subtrees beneath the remaining branches of the jth OR. Further, let X be the function computed by the AND of the remaining subtrees.

Now the function f can be written as

$$f = (L_1 + R_1)(L_2 + R_2) \cdots (L_k + R_k)X,$$

where X contains no literals from block i. (X may be empty.)

Consider the formula $f' = L_1 L_2 \cdots L_k X + R_1 R_2 \cdots R_k X$. Then any example satisfying f' also satisfies f, and therefore no negative example satisfies f'. Further note that in f' each variable of block i occurs only once, although variables of other blocks may occur twice if they are in X. Now we observe that the argument for Case 1 applies to f', since the argument only used the facts that all literals from block i are positive, which is true here due to the claim; that all negative examples were avoided, which again is true for f'; and that all literals from block i occur only once.

This completes the proof of Lemma 6.4. □

We are now ready to finish the proof of Theorem 6.2. Suppose that there is an algorithm A that on input EXAMPLES of some μ-formula of N variables, in polynomial time and with high probability found a μ-formula that was satisfied by greater than the fraction $\exp(-N^{1/3})$ of the positive examples and was not satisfied by any negative example. Then given any instance of a 2-NM-Coloring problem (S, C) with $|S| = n$, use the reduction above to obtain a μ-formula learning problem to present to algorithm A with $N = n^3$ variables, and distributions D^+ and D^-. Then by Lemma 6.3, if (S, C) is 2-NM-Colorable, then there is some μ-formula that is consistent with all positive and negative examples, and thus in polynomial time algorithm A finds some μ-formula f that is satisfied by no negative examples, and by more than

$$\exp(-N^{1/3})n^{n^2} = \exp(-n)n^{n^2} > (1 - (1/n))^{n^2}n^{n^2} = (n - 1)^{n^2}$$

positive examples. By Lemma 6.4, in random polynomial time, A can be used to find a 2-NM-Coloring of (S, C).

On the other hand, suppose (S, C) is not 2-NM-Colorable. Since the existence of a μ-formula consistent with greater than $(n - 1)^{n^2}$ positive examples and all negative examples implies 2-NM-Colorability of (S, C), algorithm A either fails to produce a formula, or produces one for which the associated coloring defined in the proof above fails to be a legitimate 2-NM-Coloring. Either of these events can be witnessed in polynomial time. Thus we have used A to solve the NP-Complete 2-NM-Coloring problem in random polynomial time.

This completes the proof of Theorem 6.2. □

7. *Conclusion*

We have seen that for some seemingly simple classes of Boolean formulas there are serious limitations to learning from examples alone. In the case of μ-formulas, even finding heuristics appears to be intractable. Although these limitations may suggest that the search for algorithms that learn in a distribution-free sense is too ambitious, there is a growing collection of positive results [9, 10, 22, 32, 33] wherein such learning algorithms are achieved. Moreover, it is difficult to argue for the applicability of results based on assumptions of uniform or normal distributions.

It seems instead that our results point out the importance of the knowledge representation used by the learning algorithm. For example, in trying to learn DNF formulas, we have seen that finding the minimum number of terms within a factor of less than 2 is NP-hard. Indeed, this approximation problem may be much more difficult since the graph-coloring approximation problem is reducible to it. Furthermore, even if learning algorithms were found that inferred formulas that were significantly (though only polynomially) larger than the minimum equivalent formulas, this may have disadvantages in applications where comprehensibility by humans is relevant and small constant-sized conjuncts and disjuncts are called for [14]. But as we have noted, allowing the more flexible representation of the *union* of the classes k-DNF, k-CNF, k-term-DNF, and k-clause-CNF results in a class of learnable formulas.

A number of areas of inquiry remain open. Can CNF (DNF) formulas be learned from examples? Can we say something further about the relationships between learnability and approximations for NP-hard optimization problems? Under reasonable restrictions on the type of example distributions allowed, do some of the hard to learn classes become learnable? Exactly what type of information other than examples would allow for the learnability of these classes?

REFERENCES

1. ANGLUIN, D. On the complexity of minimum inference of regular sets. *Inf. Control 39* (1978), 337–350.
2. ANGLUIN, D. Finding patterns common to a set of strings. *J. Comput. Syst. Sci. 21* (1980), 46–62.
3. ANGLUIN, D. Inductive inference of formal languages from positive data. *Inf. Control 45* (1980), 117–135.
4. ANGLUIN, D. Inference of reversible languages. *J. ACM 29*, 3 (July 1982), 741–765.
5. ANGLUIN, D. Remarks on the difficulty of finding a minimal disjunctive normal form for Boolean functions. Unpublished manuscript.
6. ANGLUIN, D. Learning regular sets from queries and counter-examples. Yale University Tech. Rep. YALEU/DCS/464, 1986.
7. ANGLUIN, D., AND SMITH, C. Inductive inference: Theory and methods. *ACM Comput. Surv. 15*, 3 (Sept. 1983), 237–269.
8. BLUM, L., AND BLUM, M. Toward a mathematical theory of inductive inference. *Inf. Control 28* (1975), 125–155.
9. BLUMER, A., EHRENFEUCHT, A., HAUSSLER, D., AND WARMUTH, M. Classifying learnable geometric concepts with the Vapnik–Chervonenkis dimension. In *Proceedings of the 18th Annual Symposium on the Theory of Computing* (Berkeley, Calif., May 28–30). ACM, New York, 1986, pp. 273–282.

10. BLUMER, A., EHRENFEUCHT, A., HAUSSLER, D., AND WARMUTH, M. Occam's Razor. *Inf. Process. Lett. 24* (1987), 377-380.
11. CASE, J., AND SMITH, C. Comparison of identification criteria for machine inductive inference. *Theoret. Comput. Sci. 25* (1983), 193-220.
12. CHVATAL, V. A greedy heuristic for the set covering problem. *Math. Oper. Res. 4*, 3 (1979), 233-235.
13. DALEY, R. On the error correcting power of pluralism in BC-type inductive inference. *Theoret. Comput. Sci. 24* (1983), 95-104.
14. DIETTERICH, T. C., AND MICHALSKI, R. S. A comparative review of selected methods for learning from examples. In *Machine Learning: An Artificial Intelligence Approach.* Tioga, Palo Alto, Calif., 1983.
15. FREIVALD, R. V. Functions computable in the limit by probabilistic machines. In *Mathematical Foundations of Computer Science (3rd Symposium at Jadwisin near Warsaw,* 1974). Springer-Verlag, New York, 1975.
16. FREIVALD, R. V. Finite identification of general recursive functions by probabilistic strategies. In *Proceedings of the Conference on Algebraic, Arithmetic, and Categorial Methods in Computation Theory.* Akadamie-Verlag, New York, 1979, pp. 138-145.
17. GAREY, M., AND JOHNSON, D. The complexity of near-optimal graph coloring. *J. ACM 23*, 1 (Jan. 1976), 43-49.
18. GAREY, M. AND JOHNSON, D. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.
19. GILL, J. Computational complexity of probabilistic Turing machines. *SIAM J. Comput. 6* (1977), 675-695.
20. GOLD, E. M. Complexity of automaton identification from given data. *Inf. Control 37* (1978), 302-320.
21. GOLDREICH, O., GOLDWASSER, S., AND MICALI, S. How to construct random functions. *J. ACM 33*, 3 (July 1986), 792-807.
22. HAUSSLER, D. Quantifying the inductive bias in concept learning. In *Proceedings of AAAI-86* (Philadelphia, Pa.). Morgan Kaufman, Los Altos, Calif., 1986, pp. 485-489.
23. HORNING, J. J. *A study of grammatical inference.* Ph.D. Dissertation. Computer Science Dept., Stanford Univ., Stanford, Calif., 1969.
24. JOHNSON, D. S. Worst case behaviour of graph coloring algorithms. In *Proceedings of the 5th South-Eastern Conference on Combinatorics, Graph Theory, and Computing.* Utilitas Mathematica, Winnipeg, Canada, 1974, pp. 513-528.
25. LEVIN, L. Universal sorting problems. *Prob. Pered. Inf. 9*, 3 (1973), pp. 115-116.
26. MICHALSKI, R. S., CARBONELL, J. G., AND MITCHELL, T. M. *Machine Learning: An Artificial Intelligence Approach.* Tioga, Palo Alto, Calif., 1983.
27. PITT, L. A characterization of probabilistic inference. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Press, Washington, D.C., 1984, pp. 485-494.
28. PODNIEKS, K. M. Probabilistic synthesis of enumerated classes of functions. *Sov. Math. Dokl. 16* (1975), 1042-1045.
29. ROYER, J. S. On machine inductive inference of approximations. *Inf. Control*, to appear.
30. RUDICH, S. Inferring the structure of a Markov chain from its output. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Press, Washington, D.C., 1985, pp. 321-325.
31. SMITH, C. H., AND VELAUTHAPILLAI, M. On the inference of approximate programs. Tech. Rep. 1427, Dept. of Computer Science, Univ. of Maryland, College Park, Md.
32. VALIANT, L. G. A theory of the learnable. *Commun. ACM 27*, 11 (1984), 1134-1142.
33. VALIANT, L. G. Learning disjunctions of conjunctions. In *Proceedings of the 9th IJCAI* (Los Angeles, Calif., Aug. 1985), vol. 1. Morgan Kaufman, Los Altos, Calif., 1985, pp. 560-566.
34. WIEHAGEN, R., FREIVALD, R., AND KINBER, E. B. On the power of probabilistic strategies in inductive inference. *Theoret. Comput. Sci. 28* (1984), 111-133.
35. WIGDERSON, A. A new approximate graph coloring algorithm. In *Proceedings of the 14th Annual Symposium on the Theory of Computing.* ACM, New York, 1982, pp. 325-329.

RECEIVED AUGUST 1986; REVISED JUNE 1987; ACCEPTED JANUARY 1988