Journal of Machine Learning Research 4 (2003) 157-176 Submitted 10/02; Published 6/03

On the Proper Learning of Axis-Parallel Concepts

Nader H. Bshouty BSHOUTY@CS.TECHNION.AC.IL
Department of Computer Science
Technion
Haifa 32000 Israel

Lynn Burroughs LYNNB@CPSC.UCALGARY.CA
Department of Computer Science
University of Calgary
Calgary, Alberta T2N 1N4, Canada

Editor: Dana Ron

Abstract

We study the proper learnability of axis-parallel concept classes in the PAC-learning and exact-learning models. These classes include union of boxes, DNF, decision trees and multivariate polynomials.

For *constant*-dimensional axis-parallel concepts C we show that the following problems have time complexities that are within a polynomial factor of each other.

1. C is α-properly exactly learnable (with hypotheses of size at most α times the target size) from membership and equivalence queries.
2. C is α-properly PAC learnable (without membership queries) under any product distribution.
3. There is an α-approximation algorithm for the MINEQUIC problem (given a $g \in C$ find a minimal size $f \in C$ that is logically equivalent to g).

In particular, if one has polynomial time complexity, they all do. Using this we give the first proper-learning algorithm of constant-dimensional decision trees and the first negative results in proper learning from membership and equivalence queries for many classes.

For axis-parallel concepts over a nonconstant dimension we show that with the equivalence oracle $(1) \Rightarrow (3)$. We use this to show that (binary) decision trees are not properly learnable in polynomial time (assuming P$\neq$NP) and DNF is not s^{ε}-properly learnable ($\varepsilon < 1$) in polynomial time even with an NP-oracle (assuming $\Sigma_2^P \neq P^{NP}$).

Keywords: PAC learning, exact learning, axis-parallel objects, minimizing formula size, Boolean formulas.

1. Introduction

We study the proper learnability of axis-parallel concept classes in the PAC-learning model and in the exact-learning model with membership and equivalence queries. A class $\mathcal{N}_m^n$-$\mathcal{P}\Phi$ of axis-parallel concepts is a class of Boolean formulas $\phi(T_1, T_2, \ldots, T_t)$ where ϕ is from a class of Boolean formulas Φ (such as monotone clauses, decision trees, etc.) and $\{T_i\}$ are boxes in $\mathcal{N}_m^n$, where $\mathcal{N}_m = \{0, \ldots, m-1\}$, that satisfy a certain property $\mathcal{P}$ (such as disjointness, squares, etc.). These classes include union of boxes, union of disjoint boxes, XOR of boxes, decision tree partition, and for the Boolean case $\mathcal{N}_2^n$, they include DNF, decision trees, disjoint DNF and multivariate polynomials.

The term α-proper learning refers to learning where the final hypothesis and the intermediate hypotheses used by the learner for equivalence queries, have size (number of boxes T_i) at most α times the size of the target formula. A class is properly learnable if it is 1-learnable. Table 1 summarizes the results for the n-dimensional Boolean case.

	Upper Bound			**Lower Bound**		
	Type	**Complexity**	**Source**	**Type**	**Condition**	**Source**
DNF	Nonproper	$2^{\tilde{O}(n^{1/3})}$	Klivans and Servedio (2001)	Proper	P$\neq$NP	Pillaipakkamnatt & Raghavan (1996)
	Nonproper	NP-oracle	Bshouty et al. (1996)	s^{ε}-Proper	$\Sigma_2^P \neq P^{NP}$	[ours]
				$o(\sqrt{n}/\log n)$-Proper		Hellerstein and Raghavan (2002)
CDNF	Nonproper	poly(n)	Bshouty (1995)			
	Proper	Σ_4^P-oracle	Hellerstein et al. (1996)			
	Proper	poly(n)	OPEN			
Disj-DNF	Nonproper	poly(n)	Bergadano et al. (1996)	Proper	P$\neq$NP	OPEN
	Proper	Σ_4^P-oracle	OPEN			
DT	Nonproper	poly(n)	Bshouty (1995)	Proper	P$\neq$NP	[ours]
	Proper	Σ_4^P-oracle	OPEN			
MP	Nonproper	poly(n)	Bergadano et al. (1996)	Proper	P$\neq$NP	OPEN
	Proper	Σ_4^P-oracle	OPEN			
MMP	Proper	poly(n)	Schapire and Sellie (1993)			

Table 1: Result summary for the Boolean domain ($m = 2$).

Hellerstein et al. (1996) show that proper learnability of a class C, using a polynomial number of membership and equivalence queries is possible in a machine with unlimited computational power if and only if C has *polynomial certificates*. They also show that if C has a polynomial certificate then C is properly learnable using an oracle for Σ_4^P (the class Σ_4^P contains all languages of the form $\{s \mid \exists w \forall x \exists y \forall z\, \phi(s,w,x,y,z)\}$ where ϕ is a predicate computable in time polynomial in $|s|$). They then give a polynomial-size certificate for CDNF (a polynomial-size DNF that has a polynomial-size CNF). This implies that CDNF is properly learnable using an oracle for Σ_4^P. For DNF, decision trees (DT), disjoint DNF (DNF where the conjunction of every two terms is 0) and multivariate polynomials with nonmonotone terms (MP), it is not known whether they have polynomial certificates. Therefore it is not known if they are properly learnable. Pillaipakkamnatt and Raghavan (1996) showed that DNF is not properly learnable unless P=NP. On the other hand, Bshouty et al. (1996) show that any circuit is (nonproperly) learnable with equivalence queries only and the aid of an NP-oracle. The best algorithm today for learning DNF runs in time $2^{\tilde{O}(n^{1/3})}$ (Klivans and Servedio, 2001). Recently, Hellerstein and Raghavan (2002) show that DNF has no polynomial certificate and is not $f(n)$-properly learnable for any $f(n) = o(\sqrt{n}/\log n)$.

CDNF, decision trees, disjoint DNF and multivariate polynomials over $\mathcal{N}_2^n$ are (nonproperly) learnable in polynomial time from membership and equivalence queries (Bshouty, 1995; Bergadano et al., 1996; Beimel et al., 2000). Multivariate polynomials with monotone terms (MMP) are properly learnable (Schapire and Sellie, 1993).

In this paper we use a new technique for finding negative results for learning from membership and equivalence queries (see Theorem 5). We use Theorem 5 and the result of Zantema and Bodlaender (2000) to show that if a decision tree over $\mathcal{N}_2^n$ is properly learnable from membership and equivalence queries then P=NP. We then use the result of Umans (1999) and show that if DNF over $\mathcal{N}_2^n$ is s^ε-properly learnable with an NP-oracle, where s is the size of the DNF, then $\Sigma_2^P = P^{NP}$ (the class Σ_2^P contains languages of the form $\{x \mid \exists y \forall z\, \phi(x,y,z)\}$ where ϕ is a predicate computable in time polynomial in $|x|$). We show our results are still true even if the learner can use other oracles such as subset, superset, disjointness, etc. Therefore, if P$\neq$NP then decision trees and DNF are not properly learnable from membership and equivalence queries (and all the other oracles defined in Subsection 2.3).

We then consider classes over $\mathcal{N}_m^n$ where the dimension n is constant. Table 2 summarizes our results for axis-parallel classes over $\mathcal{N}_m^n$ for a constant dimension n.

	Learnable in time poly$(\log m)$	**Not Learnable if P$\neq$NP**
Union of t Boxes	$\log t$-Proper	Proper
Disjoint Union Boxes	dim=2 Proper	dim>2 Proper
Decision Tree	Proper	
XOR **of Boxes**	α-Proper	OPEN

Table 2: Our results for constant dimension (n constant).

For axis-parallel classes over a constant dimension we show that these classes have polynomial certificates. Therefore by the result of Hellerstein et al. (1996), they are properly learnable from membership and equivalence queries using the Σ_4^P oracle. We further investigate the learnability of these classes and show that an NP-oracle is sufficient for proper learnability. We also show that the following problems have time complexities within a polynomial factor (in the size of the target) of each other.

1. C is α-properly exactly learnable from membership and equivalence queries.

2. C is α-properly PAC learnable (without membership queries) under any product distribution.

3. There is an α-approximation algorithm for the MINEQUIC problem (given a $g \in C$ find a minimal size $f \in C$ that is equivalent to g).

4. C is exactly learnable with a learning algorithm that uses all the queries (membership and nonproper equivalence, subset, superset, etc.) and outputs a hypothesis that has size at most α times the target size.

There are some surprising results that follow from this. The first is (1)$\Rightarrow$(2). It is known that (proper) learnability from equivalence and membership queries implies (proper) learnability in the PAC model with membership queries (Angluin, 1987). Here we show that in the case of finite-dimensional space and for the product distribution we can change a learner that depends on membership queries, to a learner that learns without membership queries. Another surprising result that we show from this is: a decision tree over any constant dimension is *properly learnable* from membership and equivalence queries. This contrasts with the Boolean case for which proper learning is NP-hard. Then we show that decision trees over $\mathcal{N}_m^n$ for constant n are properly PAC learnable under any distribution.

Our result also shows that union of disjoint DNF in two dimensions has a polynomial-time proper-learning algorithm. On the other hand, union of boxes and disjoint union of boxes over dimensions greater than two are properly learnable if and only if P=NP. Union of boxes is $\log t$-properly learnable where t is the number of boxes, and XOR of boxes is α-properly learnable for some constant α.

All the results in the literature for domains of constant dimension are for nonproper learning of the above classes in the exact-learning model and there were no negative results for proper learning of these classes from membership and equivalence queries.

Chen and Maass (1994) give a proper exact learning of one box from equivalence queries. Beimel and Kushilevitz (1998) show that $\mathcal{N}_m^n$ disjoint DNF is (nonproperly) learnable from membership and equivalence queries, for any dimension n. The output hypothesis is represented as a $\mathcal{N}_m^n$ multiplicity automaton. Since $\mathcal{N}_m^n$ multiplicity automaton contains the class of $\mathcal{N}_m^n$ multivariate polynomials (Beimel et al., 2000), the class of $\mathcal{N}_m^n$ multivariate polynomials is (nonproperly) learnable in polynomial time from membership and equivalence queries. Bshouty et al. (1998) give a learning algorithm that $O(d \ln t)$-properly learns a union of t boxes in d-dimensional space. This result is also implied by our work.

There are many algorithms in the literature that learn a union of boxes in constant-dimensional space (Chen and Homer, 1996; Maass and Warmuth, 1998), and even any combination of thresholds in constant-dimensional space from equivalence queries only (Ben-David et al., 1997; Bshouty, 1998). All of these algorithms are nonproper and return hypotheses that may be arbitrarily large.

2. Preliminaries

In this section we give some definitions and notation that we will use in the rest of the paper. We also give some preliminary lemmas that will be used in subsequent sections.

2.1 Learning Models

The learning criteria we consider are *exact learning* and *PAC learning*.

In the exact-learning model there is a function f called the *target function* $f : \mathcal{N}_m^n \to \{0,1\}$ (where $\mathcal{N}_m = \{0,1,\ldots,m-1\}$), which has a formula representation in a class C of formulas defined over the variable set $V_n = \{x_1,\ldots,x_n\}$. The goal of the learning algorithm is to halt and output a formula $h \in C$ that is logically equivalent to f.

To gain information about f, an exact-learning algorithm might make a *membership query* by sending an assignment $a \in \mathcal{N}_m^n$ to a *membership oracle* MQ_f which returns the value $\mathrm{MQ}_f(a) = f(a)$. The learning algorithm may also perform an *equivalence query* by sending a hypothesis $h \in C$ to an *equivalence oracle* EQ_f which returns either "YES", signifying that h is logically equivalent to f, or a *counterexample* b such that $h(b) \neq f(b)$.

We say that a class C of Boolean functions is α*-properly exactly learnable* in polynomial time from membership and equivalence queries if there is a polynomial-time algorithm $\mathcal{A}$ such that for any $f : \mathcal{N}_m^n \to \{0,1\}$ in C,

- $\mathcal{A}$ makes a polynomial number of membership and equivalence queries (polynomial in n, $\log m$ and $|f|$),
- all hypotheses $h \in C$ that $\mathcal{A}$ uses for equivalence queries, have size at most α times the size of f,
- $\mathcal{A}$ outputs a hypothesis $h \in C$ that is logically equivalent to f, and has size at most α times the size of f.

If $\alpha = 1$, we omit the α and simply say that C is *properly exactly learnable*.

The PAC-learning model is as follows. There is a distribution D defined over the domain $\mathcal{N}_m^n$. The goal of the learning algorithm is to halt and output a formula h that is ε-close to f with respect to the distribution D, that is,

$$\Pr_D[f(x) = h(x)] \geq 1 - \varepsilon.$$

We say that h is an ε-approximation of f with respect to the distribution D. In the PAC or *example query* model, the learning algorithm asks for an example from the *example oracle*, and receives an example $(a, f(a))$ where a is chosen from $\mathcal{N}_m^n$ according to the distribution D.

We say that a class of Boolean functions C is α*-properly PAC learnable* under the distribution D in polynomial time if there is an algorithm $\mathcal{A}$, such that for any $f \in C$ over V_n and any ε and δ, algorithm $\mathcal{A}$ runs in polynomial time, asks a polynomial number of queries (polynomial in n, $\log m$, $1/\varepsilon$, $1/\delta$ and the size of the target function) and with probability at least $1 - \delta$ outputs a hypothesis $h \in C$ that is an ε-approximation of f with respect to the distribution D. The size of h is at most α times the size of f. It is known (Angluin, 1987) that if a class C is α-properly exactly learnable in polynomial time from equivalence queries (and membership queries) then it is α-properly PAC learnable (with membership queries) in polynomial time under any distribution D.

We say that a distribution D is a *product* distribution over $\mathcal{N}_m^n$ if

$$D(x_1, \ldots, x_n) = D_1(x_1) D_2(x_2) \cdots D_n(x_n)$$

where each D_i is a distribution over $\mathcal{N}_m$.

2.2 Axis-Parallel Concept Classes

A *Boolean function* over $\mathcal{N}_m^n$ is a function $f : \mathcal{N}_m^n \to \{0,1\}$ on variables $V_n = \{x_1, \ldots, x_n\}$. The elements of $\mathcal{N}_m^n$ are called *assignments*. For an assignment $a \in \mathcal{N}_m^n$, the i^{th} entry of a will be denoted a_i, where $a_i \in \mathcal{N}_m$. This is the assignment for variable x_i. Our results easily extend to Boolean functions over $\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n}$, but we will continue to use $\mathcal{N}_m^n$ for its notational simplicity.

An $\mathcal{N}_m^n$ *literal* is a function with either of the following forms.

$$[x_i \geq a] = \begin{cases} 1 & \text{if } x_i \geq a \\ 0 & \text{otherwise,} \end{cases} \qquad [x_i < a] = \begin{cases} 1 & \text{if } x_i < a \\ 0 & \text{otherwise,} \end{cases}$$

where $i \in \{1, \ldots, n\}$ and $a \in \mathcal{N}_m \cup \{m\}$. The literal on the right is the negation of the one on the left. A $\mathcal{N}_m^n$ *monotone literal* refers to one of type $[x_i \geq a]$. If $m = 2$, we have the familiar Boolean variables $x_i, \bar{x}_i$ and the constants 0,1.

An $\mathcal{N}_m^n$ *term* is a product (conjunction) of $\mathcal{N}_m^n$ literals. For example, if $n = 3$, then

$$T = [x_1 \geq 2] \wedge [x_1 < 5] \wedge [x_2 \geq 9]$$

is an $\mathcal{N}_{11}^3$ term. Note that the conjunction of a term with literals of the form $[x_i \geq 0]$ or $[x_i < m]$, is logically equivalent to the term itself. Thus we may write

$$T = [2 \leq x_1 < 5] \wedge [9 \leq x_2 < 11] \wedge [0 \leq x_3 < 11],$$

where $[a \leq x_i < b] \stackrel{def}{=} [x_i \geq a][x_i < b]$. Therefore, every term $\mathcal{N}_m^n$ term can be written as

$$T_i = \bigwedge_{k=1}^{n} [a_{i,k} \leq x_k < b_{i,k}],$$

where $a_{i,k}, b_{i,k} \in \mathcal{N}_m \cup \{m\}$.

Geometrically, each term T corresponds to a box in n-dimensional space. That is, the points $x \in \mathcal{N}_m^n$ on which $T(x) = 1$ fall within an n-dimensional box whose sides are parallel to the axes. For this reason, the classes built from these terms will be called *axis-parallel concept classes*.

An $\mathcal{N}_m^n$ *monotone term* is an $\mathcal{N}_m^n$ term that uses only monotone $\mathcal{N}_m^n$ literals. An $\mathcal{N}_m^n$ *DNF* is a disjunction of $\mathcal{N}_m^n$ terms, and a $\mathcal{N}_m^n$ *monotone DNF* is a disjunction of $\mathcal{N}_m^n$ monotone terms. An $\mathcal{N}_m^n$ *multivariate polynomial* is a sum of terms (mod 2), and an $\mathcal{N}_m^n$ *disjoint DNF* is an $\mathcal{N}_m^n$ DNF where the conjunction of every two terms is 0 (geometrically, the terms are represented by n-dimensional boxes that do not overlap).

An $\mathcal{N}_m^n$ *decision tree* ($\mathcal{N}_m^n$ DT) is a full binary tree (that is, every parent node has two children) whose nodes are labeled with $\mathcal{N}_m^n$ literals and whose leaves are labeled with constants from $\{0,1\}$. Each decision tree T represents a function $f_T : \mathcal{N}_m^n \to \{0,1\}$. To compute $f_T(a)$ we start from the root of the tree T: if the root is labeled with the literal l and $l(a) = 1$, then $f_T(a) = f_{T_R}(a)$ where T_R is the right subtree of the root (that is, the subtree of the right child of the root with all its descendents). Otherwise, $f_T(a) = f_{T_L}(a)$ where T_L is the left subtree of the root. If T is a leaf then $f_T(a)$ is the label of this leaf.

In general, for every set of Boolean functions Φ (e.g., XOR, OR, etc.) and *property* function $\mathcal{P}_t : (\mathcal{N}_m^n\textit{-term})^t \to \{0,1\}$ that is computable in polynomial time, (e.g., $\mathcal{P}_t(T_1, \ldots, T_t) = 1$ if $T_1, \ldots, T_t$ are pairwise disjoint) we can build a concept over $\mathcal{N}_m^n$ as follows. We define the axis-parallel concept class $\mathcal{P}\Phi[\mathcal{N}_m^n]$ to be the set of all $\phi(T_1, \ldots, T_t)$ where $\phi \in \Phi$ and $\{T_i\}$ are $\mathcal{N}_m^n$ terms with $\mathcal{P}_t(T_1, \ldots, T_t) = 1$. If $\mathcal{P}_t \equiv 1$ then we write $\Phi[\mathcal{N}_m^n]$.

For example, let $\Phi = \{x_1, x_1 \vee x_2, x_1 \vee x_2 \vee x_3, \cdots\}$ be the set of monotone clauses and define the property function $\mathcal{P}_t$ such that $\mathcal{P}_t(T_1, \ldots, T_t) = 1$ if and only if $T_i \wedge T_j = 0$ for every $1 \leq i < j \leq t$. Then $\mathcal{P}\Phi[\mathcal{N}_2^n]$ is the set of disjoint DNF (in the Boolean domain) and $\mathcal{P}\Phi[\mathcal{N}_m^2]$ is the set of unions of disjoint rectangles in two-dimensional space.

The size of a formula $f \in \mathcal{P}\Phi[\mathcal{N}_m^n]$, which we will denote $size_{\mathcal{P}\Phi}(f)$, will be defined to be the smallest number of $\mathcal{N}_m^n$ terms $T_1, \ldots, T_t$ such that for some $h \in \Phi$, $f \equiv h(T_1, \ldots, T_t)$ and $\mathcal{P}_t(T_1, \ldots, T_t) = 1$. For the class of decision trees, the size will be the minimum number of non-leaf nodes used by an $\mathcal{N}_m^n$ decision tree equivalent to f.

Our main result uses a technique in which we compress the domain of a function in an attempt to extract its main features. That is, we will map a function $f : \mathcal{N}_m^n \to \{0,1\}$ to $f' : \mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n} \to \{0,1\}$ for which each $m_i \leq m$. We need such a mapping (called a projection) to retain its relative ordering of points. A *monotone projection* from $\mathcal{N}_{m'}$ to $\mathcal{N}_m$ is a function $M : \mathcal{N}_{m'} \cup \{m'\} \to \mathcal{N}_m \cup \{m\}$ such that for every $i, j \in \mathcal{N}_{m'}$ where $i < j$ we have $M(i) \leq M(j)$ and $M(m') = m$. Extending this to n dimensions, a monotone projection $\mathcal{M} : (\mathcal{N}_{m'} \cup \{m'\})^n \to (\mathcal{N}_m \cup \{m\})^n$ is $\mathcal{M} = (M_1, \ldots, M_n)$ where each $M_i : \mathcal{N}_{m'} \cup \{m'\} \to \mathcal{N}_m \cup \{m\}$ is a monotone projection. We say that $C = \mathcal{P}\Phi[\mathcal{N}_m^n]$ is *closed* *under monotone projection* if for any monotone projection $\mathcal{M}$ whenever $\mathcal{P}(T_1,\dots,T_t)=1$ we also have $\mathcal{P}(T_1\mathcal{M},\dots,T_t\mathcal{M})=1$. A class with the property that the boxes be equilateral is an example of a class not closed under monotone projection. Notice that if $f=\phi(T_1,\dots,T_t)\in\Phi[\mathcal{N}_m^n]$ then $f\mathcal{M}=\phi(T_1\mathcal{M},\dots,T_t\mathcal{M})\in\Phi[\mathcal{N}_{m'}^n]$. If the class $C=\mathcal{P}\Phi[\mathcal{N}_m^n]$ is closed under monotone projection then $f\mathcal{M}\in\mathcal{P}\Phi[\mathcal{N}_{m'}^n]$.

To map functions on compressed domains to functions on larger domains, we define the *dual monotone projection* . For a monotone projection $M:\mathcal{N}_{m'}\cup\{m'\}\to\mathcal{N}_m\cup\{m\}$ define the dual monotone projection $M^\star:N_m\cup\{m\}\to\mathcal{N}_{m'}\cup\{m'\}$ where $M^\star(y)$ is the minimal x such that $M(x)\geq y$. Since $M(m')=m$, the dual monotone projection is well defined. For a monotone projection $\mathcal{M}=(M_1,\dots,M_n)$ we define $\mathcal{M}^\star=(M_1^\star,\dots,M_n^\star)$.

The monotone projection that we will use in this paper is the *lattice projection* . Since it will be used for classes of constant dimension, we will use d for the dimension n. A *lattice* in $\mathcal{N}_m^d$ is $L=L_1\times\cdots\times L_d$ where each $L_i\subseteq\mathcal{N}_m$. Notice that a lattice represents a subsampling of the domain, in which only certain rows (those in L_i) are represented for each dimension i.

Let $f\in\phi(T_1,\dots,T_t)\in\Phi[\mathcal{N}_m^d]$ where $\phi\in\Phi$ and

$$T_i=\bigwedge_{k=1}^{d}[a_{i,k}\leq x_k<b_{i,k}].$$

Let $L=L_1\times\cdots\times L_d$ be a lattice in $\mathcal{N}_m^d$. For $a,b\in\mathcal{N}_m$ define

$$\lfloor a\rfloor_{L_i}=\max(\{x\in L_i\mid x\leq a\}\cup\{0\}),$$

$$\lceil b\rceil_{L_i}=\min(\{x\in L_i\mid x\geq b\}\cup\{m\}).$$

Note that $\lfloor a\rfloor_{L_i}$ is a monotone projection, while $\lceil b\rceil_{L_i}$ is its dual monotone projection. For an assignment $v\in\mathcal{N}_m^d$ we define $\lfloor v\rfloor_L=(\lfloor v_1\rfloor_{L_1},\dots,\lfloor v_d\rfloor_{L_d})$. Define

$$T_i^L=\bigwedge_{k=1}^{d}[\lceil a_{i,k}\rceil_{L_k}\leq x_k<\lceil b_{i,k}\rceil_{L_k}]$$

and $f^L=\phi(T_1^L,\dots,T_t^L)$. We call f^L the *lattice projection* of f on L. Some useful properties of the lattice projection are given by the next lemma.

Lemma 1

1. $f^L(u)=f(\lfloor u\rfloor_L)=f^L(\lfloor u\rfloor_L)$.
2. $size_{\mathcal{P}\Phi}(f^L)\leq size_{\mathcal{P}\Phi}(f)$.
3. *Let* $\mathcal{P}\Phi[\mathcal{N}_m^d]$ *be closed under monotone projection. For any function* $f\in\mathcal{P}\Phi[\mathcal{N}_m^d]$ *and lattice* L *we have* $f^L\in\mathcal{P}\Phi[\mathcal{N}_m^d]$.
4. *Let* $f=\phi(T_1,\dots,T_t)$ *where* $\phi\in\Phi$ *and* $T_i=\wedge_{k=1}^d[a_{i,k}\leq x_k<b_{i,k}]$. *If for every* i *and* k *we have* $a_{i,k},b_{i,k}\in L_k$ *then* $f^L\equiv f$.

Proof. For (1) it is enough to examine the literals. For $f^L(u)$ the literals are $[\lceil a\rceil_{L_k} \le u_k < \lceil b\rceil_{L_k}]$, which (since $\lfloor a\rfloor_{L_k}$ is a monotone projection), is equivalent to

$$\left[\lfloor\lceil a\rceil_{L_k}\rfloor_{L_k} \le \lfloor u_k\rfloor_{L_k} < \lfloor\lceil b\rceil_{L_k}\rfloor_{L_k}\right] = [\lceil a\rceil_{L_k} \le \lfloor u_k\rfloor_{L_k} < \lceil b\rceil_{L_k}]$$

which is the corresponding literal in $f^L(\lfloor u\rfloor)$. Similarly, $f(\lfloor u\rfloor_L)$ has literals $[a \le \lfloor u_k\rfloor_{L_k} < b]$, which (since $\lceil b\rceil_{L_k}$ is a monotone projection), is equivalent to

$$[\lceil a\rceil_{L_k} \le \lceil\lfloor u_l\rfloor_{L_k}\rceil < \lceil b\rceil_{L_k}] = [\lceil a\rceil_{L_k} \le \lfloor u_k\rfloor_{L_k} < \lceil b\rceil_{L_k}]$$

which is also the corresponding literal in $f^L(\lfloor u\rfloor)$.

Since the lattice projection does not increase the number of terms (although it may render some terms redundant), (2) follows.

For item (3), note that the lattice projection does not change the domain. Also, if $\mathcal{P}_t(T_1,\ldots,T_t) = 1$ then $\mathcal{P}_t(T_1^L,\ldots,T_t^L) = 1$. Thus f^L remains in the same class as f.

Item (4) follows from the fact that if $a_{i,k}, b_{i,k} \in L_k$ then $\lceil a_{i,k}\rceil_{L_k} = a_{i,k}$ and $\lceil b_{i,k}\rceil_{L_k} = b_{i,k}$, which implies $T_i^L = T_i$ and hence $f^L = \phi(T_1^L,\ldots,T_t^L) = \phi(T_1,\ldots,T_t) = f$. □

It is not necessary that each dimension have the same size. All the results in this paper are also true for the class $\mathcal{P}\Phi[\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n}]$. In that case a monotone projection is $\mathcal{M} = (M_1,\ldots,M_n)$ where $M_i : N_{m_i'} \to N_{m_i}$ and for $f \in \mathcal{P}\Phi[\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n}]$ we have $f\mathcal{M} \in \mathcal{P}\Phi[\mathcal{N}_{m_1'} \times \cdots \times \mathcal{N}_{m_n'}]$.

Constructiveness Assumption: We assume that there is an algorithm "Construct" such that for any function $\psi : \mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n} \to \{0,1\}$ that is computable in polynomial time, Construct(ψ) runs in time $poly(\prod m_i)$ and returns some formula $f \in \mathcal{P}\Phi[\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n}]$ that is equivalent to ψ, if there exists such a formula, and returns "error" otherwise. Such algorithms exist (and are in fact very trivial) for all the classes presented in this paper.

2.3 Oracles

In addition to the example, membership and equivalence oracles, we will also consider the following oracles defined by Angluin (1987).

- **Subset oracle**. $Sub_f(h)$ for $h \in C$. This oracle returns 'YES' if $h \Rightarrow f$ and returns a counterexample a such that $h(a) = 1$ and $f(a) = 0$ otherwise.

- **Superset oracle**. $Sup_f(h)$ for $h \in C$. This oracle returns 'YES' if $h \Leftarrow f$ and returns a counterexample a such that $h(a) = 0$ and $f(a) = 1$ otherwise.

- **Disjointness oracle**. $Disj_f(h)$ for $h \in C$. This oracle returns 'YES' if $h \wedge f = 0$ and returns a counterexample a such that $h(a) = 1$ and $f(a) = 1$ otherwise.

- **Exhaustiveness oracle**. $Exh_f(h)$ for $h \in C$. This oracle returns 'YES' if $h \vee f = 1$ and returns a counterexample a such that $h(a) = 0$ and $f(a) = 0$ otherwise.

Given a set of oracles O, we say that O is *easy* (resp. NP-easy) for C if every oracle in O can be simulated in polynomial time for C (resp. simulated in polynomial time using an NP-oracle), where the simulation uses the target function f.

Lemma 2 *Let Φ be a set of Boolean functions. For a constant dimension d, membership and equivalence oracles and all the above oracles (subset, superset, etc.) are easy for $\Phi[\mathcal{N}_m^d]$.*

Proof. Suppose we have a target f and wish to simulate an oracle $R_f(h)$. Given a hypothesis h, we take all the literals $[x_iQa]$ for $Q \in \{\geq, <\}$ in the terms of the target f and the formula h. Let A be the set of all the a's in those terms and the two constants 0 and m. Suppose $A = \{0 = a_1 < a_2 < \cdots < a_t = m\}$. Notice that $t \leq 2d(s_h + s_f + 2)$ where s_h and s_f are the number of terms in h and f, respectively. It is easy to see that the functions h and f are constant functions 0 or 1 in each subdomain $[a_{i_1}, a_{i_1+1}) \times \cdots \times [a_{i_d}, a_{i_d+1})$. Thus we only need to compare f and h on a single point in each subdomain. The number of subdomains is $(t-1)^d$ which is polynomial for any constant d. □

2.4 Polynomial Certificates

Following the definition of Hellerstein et al. (1996), the class $C = \mathcal{P}\Phi[\mathcal{N}_m^n]$ has polynomial certificates if for every $f \in C$ of size t there are $q = poly(t,n)$ assignments $A = \{a_1^f, \ldots, a_q^f\}$ such that for every $g \in C$ of size less than t, g is not consistent with f on A. That is, $g(a) \neq f(a)$ for some $a \in A$.

Lemma 3 *Let d be constant. If $C = \mathcal{P}\Phi[\mathcal{N}_m^d]$ is closed under monotone projection then it has polynomial certificates.*

Proof. Let $f = \phi(T_1, \ldots, T_t)$ where $\phi \in \Phi$ and $T_i = \wedge_{k=1}^d [a_{i,k} \leq x_k < b_{i,k}]$. Let $L_k = \{a_{i,k}, b_{i,k} \mid i = 1, \ldots, t\}$ and $L = L_1 \times \cdots \times L_d$. Notice that $|L| \leq (2t)^d = poly(t)$ for constant d. We now show that if $g \in \mathcal{P}\Phi[\mathcal{N}_m^d]$ is consistent with f on L then $size_{\mathcal{P}\Phi}(g) \geq t$.

Since g is consistent with f on L, by Lemma 1 parts 1 and 4, we have $g^L(x) = g(\lfloor x \rfloor_L) = f(\lfloor x \rfloor_L) = f^L(x) = f(x)$. By Lemma 1 part 3 we have $g^L \in \mathcal{P}\Phi[\mathcal{N}_m^d]$. Therefore, by Lemma 1 part 2 we have $size_{\mathcal{P}\Phi}(g) \geq size_{\mathcal{P}\Phi}(g^L) = size_{\mathcal{P}\Phi}(f)$. □

It follows from Hellerstein et al. (1996) that if $C = \mathcal{P}\Phi[\mathcal{N}_m^d]$ is closed under monotone projection then C is learnable from membership and equivalence queries using an oracle for Σ_4^P. In this paper we will show that an NP-oracle is sufficient.

2.5 Approximation Algorithms

We assume the reader is familiar with approximation algorithms, α-approximation and some of the basic concepts in approximation theory and complexity theory. For a problem in which we seek to minimize the size of a formula equivalent to a function f, an α-approximation algorithm (for $\alpha \geq 1$) returns a formula $h \equiv f$ that has size at most α times the size of the smallest formula equivalent to f. Given a class $C = \mathcal{P}\Phi[\mathcal{N}_m^n]$, we define the optimization problem Minimal Equivalent Formula (MINEQUIC) to be the following.

MINEQUIC

Given a formula $f \in C = \mathcal{P}\Phi[\mathcal{N}_m^n]$.
Find a minimal size $h \in C$ that is equivalent to f.

We expect an algorithm for MINEQUIC to run in time polynomial in n and the size of f.

In some cases the function f is given as an n-dimensional $m_1 \times m_2 \times \cdots \times m_n$ matrix and an algorithm is expected to find a minimal size $h \in C$ in time polynomial in $\prod m_i$. We call the matrix the *unary representation* of f. Then MINEQUIC for unary inputs is defined as follows.

<u>MINEQUI C_U</u>

Given an $m_1 \times \cdots \times m_n$ matrix A representing a formula in $C = \mathcal{P}\Phi[\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n}]$.
Find a minimal size $h \in C$ that represents A.

A generalization of this problem is the MINEQUI$^\star C_U$ problem. For MINEQUI$^\star C_U$ the input is given in its unary representation, but the matrix may contain $\star$ entries which denote unspecified values.

<u>MINEQUI$^\star C_U$</u>

Given an $m_1 \times \cdots \times m_n$ matrix A with entries $0, 1$ and $\star$, representing a function in $C = \mathcal{P}\Phi[\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_n}]$.
Find a minimal size $h \in C$ that is equivalent to f on the specified values.

Another problem that is related to the latter problem is the minimal-size consistency problem.

<u>MINEQUI$^\star C$</u>

Given a set $S = \{(a_1, f(a_1)), \cdots, (a_t, f(a_t))\}$ where $f \in C = \mathcal{P}\Phi[\mathcal{N}_m^n]$.
Find a minimal-size $h \in C$ that is consistent with S. That is, $h(a_i) = f(a_i)$ for all i.

Some of these problems appear in the literature. For example, MINEQUI $\mathcal{N}_m^2$ DNF$_U$ is the task of covering orthogonal polygons by rectangles. MINEQUI $\mathcal{N}_m^2$ Disj-DNF$_U$ is the problem of partitioning orthogonal polygons into rectangles.

Next we show that MINEQUIC and MINEQUIC_U are equivalent for the constant-dimensional domain.

Lemma 4 *Let $C = \mathcal{P}\Phi[\mathcal{N}_m^d]$ for a constant d, be closed under monotone projection. Then*

1. MINEQUIC *has a polynomial-time* α*-approximation algorithm if and only if* MINEQUIC_U *has a polynomial-time* α*-approximation algorithm.*

2. MINEQUI$^\star C$ *has a polynomial-time* α*-approximation algorithm if and only if* MINEQUI$^\star C_U$ *has a polynomial-time* α*-approximation algorithm.*

Proof. We prove (1). The proof of (2) is similar. Let $\mathcal{A}$ be a polynomial-time α-approximation algorithm for MINEQUIC_U. We will describe an approximation algorithm $\mathcal{B}$ for MINEQUIC. Let $f \in C$ be represented as $\hat{\phi}(\hat{T}_1, \ldots, \hat{T}_{t'})$ where $\hat{\phi} \in \Phi$ and $\hat{T}_i = \bigwedge_{k=1}^{d}[a_{i,k} \le x_k < b_{i,k}]$. Algorithm $\mathcal{B}$ builds the sets

$$L_k = \{a_{i,k}, b_{i,k} \mid i = 1, \ldots, t', k = 1, \ldots, d\} = \{c_{1,k} < c_{2,k} < \ldots < c_{m_k,k}\}$$

and defines the lattice $L = L_1 \times \cdots \times L_d$. By Lemma 1 we have $f^L(x) = f(x) = f(\lfloor x \rfloor_L)$ which implies that all features of f are captured by the lattice. Now $\mathcal{B}$ builds an $m_1 \times \cdots \times m_d$ matrix A that stores the values of f on the lattice. That is, $A[i_1, \ldots, i_d] = f(c_{i_1,1}, \cdots, c_{i_d,d})$. Matrix A represents a function $f\mathcal{M}$ in which $\mathcal{M}$ compresses the domain of f by mapping each $c_{i_k,k}$ in $\mathcal{N}_m$ to its ordinal $i_k \in \{1, \ldots, m_k\}$. That is, $\mathcal{M}(c_{i_1,1}, \ldots, c_{i_d,d}) = (i_1, \ldots, i_d)$. Clearly this is a monotone projection, as is its inverse $\mathcal{M}^-(i_1, \ldots, i_d) = (c_{i_1,1}, \ldots, c_{i_d,d})$. The size of matrix A is at most $\prod m_i \le (2t')^d$ which is polynomial for constant d. Algorithm $\mathcal{B}$ now uses $\mathcal{A}(A)$ to find a $\phi \in \Phi$ and $T_1, \ldots, T_t$ such that

$$\mathcal{P}_t(T_1, \ldots, T_t) = 1, \qquad f\mathcal{M} = \phi(T_1, \ldots, T_t) \qquad \text{and} \qquad t \le \alpha \cdot size_{\mathcal{P}\Phi}(f\mathcal{M}).$$

Algorithm $\mathcal{B}$ then uses the inverse $\mathcal{M}^-$ and returns

$$g = \phi(T_1\mathcal{M}^-, \dots, T_t\mathcal{M}^-).$$

Since C is closed under monotone projection and $\mathcal{M}^-$ is a monotone projection we know that $\mathcal{P}_t(T_1\mathcal{M}^-, \dots, T_t\mathcal{M}^-) = 1$. Therefore, $g \in \mathcal{P}\Phi[\mathcal{N}_m^d]$. We also have by Lemma 1,

$$size_{\mathcal{P}\Phi}(g) \le t \le \alpha \cdot size_{\mathcal{P}\Phi}(f\mathcal{M}) \le \alpha \cdot size_{\mathcal{P}\Phi}(f).$$

Finally, we have

$$g(x) = \phi(T_1\mathcal{M}^-, \dots, T_t\mathcal{M}^-) = (f\mathcal{M})(\mathcal{M}^-(x)) = f(\lfloor x \rfloor_L) = f(x).$$

For the other direction, suppose that we have an algorithm for MINEQUIC. Given an $m_1 \times \cdots \times m_d$ matrix A, by the constructiveness assumption we can build a formula $f \in \mathcal{P}\Phi[\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_d}]$ that represents A in time $poly(\prod m_i)$ and then using the algorithm for MINEQUIC we get the desired representation. □

3. Approximation Algorithms and Learning

In this section we show the connection between approximation and learning.

Theorem 5 *If C is α-properly exactly learnable from a set O of oracles and O is easy (resp., NP-easy) for C then* MINEQUIC *has an α-approximation algorithm (resp., with the aid of an NP-oracle).*

Proof. Let $\mathcal{A}$ be a learning algorithm that uses the oracles in O to learn a hypothesis of size at most α times the size of the target. Since O is easy for C, all the oracles in O can be simulated in polynomial time for C. So we can run algorithm $\mathcal{A}$ and simulate all the oracles in polynomial time. Since the learning algorithm is α-proper, the output hypothesis has size less than α times the size of the target. □

The next theorem follows from Lemma 4 and the Occam Theorem (Blumer et al., 1987), which shows that one can PAC learn by taking a sufficiently-large (but polynomial) sample and finding a hypothesis consistent with the sample.

Theorem 6 *Let $C = \mathcal{P}\Phi[\mathcal{N}_m^d]$ for a constant d be closed under monotone projection. The following three problems have time complexities within a polynomial factor (in the target size) of each other.*

1. *C is α-properly PAC learnable.*
2. *There is an α-approximation algorithm for* MINEQUI$^\star C$.
3. *There is an α-approximation algorithm for* MINEQUI$^\star C_U$.

We now demonstrate a connection between α-proper learning in the PAC and exact models, and α-approximation of MINEQUIC, for classes of constant dimension.

Theorem 7 *Let $C = \mathcal{P}\Phi[\mathcal{N}_m^d]$ for a constant d be closed under monotone projection. The following problems have time complexities within a polynomial factor of each other.*

1. *C is α-properly exactly learnable from membership and equivalence queries.*

2. *C is α-properly PAC learnable (without membership queries) under any product distribution.*

3. *There is an α-approximation algorithm for the* MINEQUIC *problem.*

Proof. We first show $(1) \equiv (3)$. By Theorem 5 and since by Lemma 2 the oracles are easy, we have $(1) \Rightarrow (3)$. Now we show $(3) \Rightarrow (1)$. Let $\mathcal{A}$ be an α-approximation algorithm for MINEQUIC. Let $f = \phi(T_1, \ldots, T_t)$ be the target function where $\phi \in \Phi$, $f \in \mathcal{P}\Phi[\mathcal{N}_m^d]$ and

$$T_i = \bigwedge_{k=1}^{d} [a_{i,k} \le x_k < b_{i,k}].$$

We now give a learning algorithm that uses a technique similar to one used by Bshouty et al. (1998). However, where that learning algorithm was nonproper, ours is α-proper. This learning algorithm works as follows. At each stage it holds d sets $L_1, \ldots, L_d$ where $L_k \subseteq (\{a_{i,k}, b_{i,k} \mid i = 1, \ldots, t\} \cup \{0\})$. L_k is initially $\{0\}$. The elements of L_k are sorted in an increasing order:

$$L_k = \{c_{1,k} < c_{2,k} < \cdots < c_{i_k,k}\}.$$

From the lattice $L = L_1 \times \cdots \times L_d$, the learning algorithm then builds the function f^L. It saves a table A of size $\prod_{k=1}^{d}(|L_k|+1)$ where $A[j_1, \ldots, j_d] = f(c_{j_1,1}, \ldots, c_{j_d,d})$. The table can be filled in using the membership oracle. Now, f^L can be defined using this table by $f^L(u) = f(\lfloor u \rfloor_L) = A[j_1, \ldots, j_d]$ where $\lfloor u_i \rfloor_{L_i} = c_{j_i}$ for $i = 1, \ldots, d$. Since $f^L \in \mathcal{P}\Phi[\mathcal{N}_m^d]$ (the class is closed under monotone projection), we can use algorithm $\mathcal{A}$ and construct a formula h that is equivalent to f^L and has size at most α times the size of f^L. Since the size of f^L is at most the size of f, the size of h is at most α times the size of f.

After we construct $h \equiv f^L(x)$, we ask the equivalence query EQ($h(x)$). Let v be the counterexample. That is, $f^L(v) = h(v) \ne f(v)$. Then by Lemma 1 $f(\lfloor v \rfloor_L) = f^L(\lfloor v \rfloor_L) = f^L(v) \ne f(v)$. Intuitively, since $f(\lfloor v \rfloor_L) \ne f(v)$, the straight line that connects the two points v and $\lfloor v \rfloor_L$ hits the "boundary" of f. Now we give an algorithm to find a point close to this boundary and using this point we will add a new point to the lattice L that is equal to one new $a_{i,j}$ or $b_{i,j}$.

The algorithm works as follows. It first finds the smallest value k (or some k using binary search) such that

$$f(\lfloor v_1 \rfloor_{L_1}, \ldots, \lfloor v_{k-1} \rfloor_{L_{k-1}}, v_k, v_{k+1}, \ldots, v_d) \ne f(\lfloor v_1 \rfloor_{L_1}, \ldots, \lfloor v_{k-1} \rfloor_{L_{k-1}}, \lfloor v_k \rfloor_{L_k}, v_{k+1}, \ldots, v_d).$$

Such a k exists since $f(\lfloor v \rfloor_L) \ne f(v)$. Then (again with a binary search) the algorithm finds an integer c such that $\lfloor v_k \rfloor_{L_k} < c \le v_k$ where

$$f(\lfloor v_1 \rfloor_{L_1}, \ldots, \lfloor v_{k-1} \rfloor_{L_{k-1}}, c-1, v_{k+1}, \ldots, v_d) \ne f(\lfloor v_1 \rfloor_{L_1}, \ldots, \lfloor v_{k-1} \rfloor_{L_{k-1}}, c, v_{k+1}, \ldots, v_d).$$

Now we prove the following.

Claim 8 *We have $c \in \{a_{i,k}, b_{i,k}\}$ for some i, and $c \notin L_k$.*

Proof of Claim 8. Let $\theta(x_k) = f(\lfloor v_1 \rfloor_{L_1}, \ldots, \lfloor v_{k-1} \rfloor_{L_{k-1}}, x_k, v_{k+1}, \ldots, v_d)$. Here v_i are constant so the function θ is a function on one variable x_k. So $\theta(x_k) = \phi(\hat{T}_1, \ldots, \hat{T}_m)$ where each $\hat{T}_i$ is either 0, 1 or

$[a_{i,k} \le x_k < b_{i,k}]$. Since $\theta(c-1) \neq \theta(c)$ we must have some i where either $a_{i,k} \le c$ and $a_{i,k} > c-1$, or $c-1 < b_{i,k}$ and $c \ge b_{i,k}$. In the first case $c = a_{i,k}$ and in the second case $c = b_{i,k}$. Thus $c \in \{a_{i,k}, b_{i,k}\}$.

Now notice that $\lfloor v_k \rfloor_{L_k} \neq v_k$ and $\lfloor v_k \rfloor_{L_k} < c \le v_k$. Therefore, c was not in L_k. □

We add c to L_k and update the table by adding all the missing values $f(v)$ where $v \in L_1 \times \cdots \times L_d$. We now show that this algorithm runs in polynomial time. Notice that the number of equivalence queries is at most

$$\sum_{k=1}^{d} |\{a_{i,k}, b_{i,k} \mid i = 1, \ldots, t\} \cup \{0\}| \;\le\; (2t+1)d,$$

and the number of membership queries is at most the size of the table which is $(2t+1)^d$ plus the number of membership queries needed for the binary search. We do one binary search for each equivalence query. Therefore the algorithm uses at most $(2t+1)^d + (2t+1)d^2 \log m$ membership queries.

We now give the proof that (2) is equivalent to (3).

To prove (2) $\Rightarrow$ (3), we use the following standard argument (see for example Haussler et al., 1991). Suppose C is α-properly PAC learnable under any product distribution. We show that (3) is true for MINEQUIC_U. Then by Lemma 4 the result follows. Let A be an $m_1 \times \cdots \times m_d$ matrix, an instance for MINEQUIC_U. Define the product distribution $D(i_1, \ldots, i_d) = 1/(\prod_{i=1}^{d} m_i)$ (uniform over $\mathcal{N}_{m_1} \times \cdots \times \mathcal{N}_{m_d}$). We now run algorithm $\mathcal{A}$ with error $\varepsilon = 1/(2\prod_{i=1}^{d} m_i)$. The hypothesis we get is consistent with A with probability at least $1-\delta$ and has size at most α times the target size.

To prove (3) $\Rightarrow$ (2), let $\mathcal{A}$ be an α-approximation algorithm for MINEQUIC. Let $r(1/\varepsilon, 1/\delta)$ be the number of examples needed to learn C, assuming we have unlimited computational power. This r is polynomial and can be upper bounded by the VC-dimension Theorem (Blumer et al., 1989). Let $\mathcal{B}$ be a polynomial-time (nonproper) PAC-learning algorithm for C under any distribution D. Such an algorithm exists (Bshouty, 1998). The idea of the proof is very simple. Since we cannot use membership queries, we learn a nonproper hypothesis $\hat{h}$ that is close to the target function f and then use $\hat{h}$ for membership queries. We do that by first running $\mathcal{B}$ to nonproperly learn the target function with a small error. Algorithm $\mathcal{B}$ will output some hypothesis $\hat{h}$. Then we use the hypothesis $\hat{h}$ to simulate membership queries of f. We show that if the distribution is the product distribution then with high probability $\hat{h}$ simulates membership queries of f.

We define the following algorithm to learn f.

Proper_Learning
1. Run $\mathcal{B}$ with $\hat{\varepsilon} = \delta/(3r^d)$ where $r = r(1/\varepsilon, 3/\delta)$ and $\hat{\delta} = \delta/3$.
   Let $\hat{h}$ be the output hypothesis.
2. Get $r = r(1/\varepsilon, 3/\delta)$ examples $(x^{(1)}, f(x^{(1)})), \ldots, (x^{(r)}, f(x^{(r)}))$.
3. Define $L_i = \{x_i^{(j)} \mid j = 1, \ldots, r\}$ and $L = L_1 \times \cdots \times L_d$.
4. Define a $\hat{m}_1 \times \cdots \times \hat{m}_d$ matrix A with $\hat{m}_i = |L_i|$ and $A[i_1, \ldots, i_d] = \hat{h}(x_1^{(i_1)}, \ldots, x_d^{(i_d)})$.
5. Run $\mathcal{A}$ on A and let $h : \mathcal{N}_{\hat{m}_1} \times \cdots \times \mathcal{N}_{\hat{m}_d} \to \{0,1\}$ be the output.
6. Define $g = h\mathcal{M}^\star$ where $\mathcal{M}^\star(x) = \mathcal{M}^{-1}(\lfloor x \rfloor_L)$ and $\mathcal{M}^{-1}(x_1^{(i_1)}, \ldots, x_d^{(i_d)}) = (i_1, \ldots, i_d)$.
7. Output (g)

In algorithm **Proper_Learning**, step 1 learns some function $\hat{h}$. Steps 2-3 take examples and build a lattice L. Since we do not have membership queries to find the value of the target on this lattice we use instead $\hat{h}$ to find the values. Steps 4-6 build a consistent hypothesis.

We now show that with probability at least $1-\delta$ all of the membership queries that are simulated by $\hat{h}$ give a correct answer. Notice that since the distribution is the product distribution, $x_1^{(i_1)},\ldots,x_d^{(i_d)}$ are chosen independently. Therefore,

$$\Pr[\hat{h}(x_1^{(i_1)},\ldots,x_d^{(i_d)}) \neq f(x_1^{(i_1)},\ldots,x_d^{(i_d)})] \leq \hat{\varepsilon}$$

and

$$\Pr[\exists x \in L\ :\ \hat{h}(x) \neq f(x)] \leq \hat{\varepsilon}|L| \leq \hat{\varepsilon} r^d = \frac{\delta}{3}.$$

Since the learning algorithms $\mathcal{B}$ and $\mathcal{A}$ also have failure probability at most $\delta/3$, we are guaranteed that algorithm **Proper_Learning** succeeds with probability at least $1-\delta$. This completes the proof of Theorem 7. □

As a corollary, it follows that for classes with constant dimension that are closed under monotone projection, a Σ_4^P oracle is not required for proper learning. An NP-oracle is sufficient.

Corollary 9 *Let $C = \mathcal{P}\Phi[\mathcal{N}_m^d]$ for a constant d be closed under monotone projection. Then C is properly learnable from membership and equivalence queries using an oracle for NP.*

Proof. Since MINEQUIC_U can be solved in polynomial time using an NP-oracle, by Theorem 7, C is properly learnable from membership and equivalence queries using an oracle for NP. □

Let $C = \mathcal{P}\Phi[\mathcal{N}_m^d]$ for a constant d be closed under monotone projection. Consider these problems:

1. C is exactly learnable with a learning algorithm that uses oracles that are easy for C, and outputs a hypothesis of size at most α times the target size.

2. C is exactly learnable with a learning algorithm that uses equivalence queries only. The hypotheses given to the equivalence oracle may have size larger than α times the target size, but the output hypothesis has size at most α times the target size.

Since all the oracles for $C = \mathcal{P}\Phi[\mathcal{N}_m^d]$ are easy, both problems give an α-approximation algorithm for MINEQUIC. Therefore they are equivalent to the problems in Theorem 7.

This shows that a negative result for the α-approximation of MINEQUIC will give a negative result for the α-proper learnability of C from all of the oracles mentioned in Section 2.3.

4. Positive and Negative Results for Proper Learning

In this section we will prove positive and negative results for the α-proper learning of different axis-parallel classes.

4.1 Decision Trees

In this subsection we give our results for decision trees.

We first show the following.

Lemma 10 *The class of decision trees is easy for all the oracles.*

Proof. For any $\phi : \{0,1\}^2 \to \{0,1\}$, and decision trees T_f for f and T_g for g, let $\phi(f,g)(x) = \phi(f(x),g(x))$ (for example, ϕ may compute the XOR of the two trees). We show that for any such ϕ, there is a polynomial-time algorithm $\mathcal{A}$ such that $\mathcal{A}$ can decide in polynomial time if $\phi(f,g) \equiv 0$ and if $\phi(f,g) \not\equiv 0$ then $\mathcal{A}$ finds an assignment x_0 such that $\phi(f(x_0),g(x_0)) = 1$.

The algorithm takes the decision tree T_f and replaces each leaf v in T_f by a decision tree $T_g^v \equiv T_g$. It then takes each leaf u in T_g^v and labels it with $\phi(l_v,l_u)$ where l_v is the label of v in T_f and l_u is the label of u in T_g^v. It is easy to see that this new tree computes $\phi(f,g)$. We will call this tree T'.

Each path in the tree T' from its root to a leaf labeled with 1, defines a term. If all such terms are identically 0 then there is no assignment that gives value 1 in T' and therefore $\phi(f,g) \equiv 0$. Otherwise, there is a term that is 1 for some assignment x_0 and then the algorithm returns x_0. □

Theorem 11 *If there is a proper-learning algorithm for $\mathcal{N}_2^n$ decision trees from membership and equivalence queries (and other oracles) then $P = NP$.*

Proof. Decision tree is easy for all the oracles. Then, this result follows from Theorem 5 because MINEQUI $\mathcal{N}_2^n$ decision tree is NP-complete (Zantema and Bodlaender, 2000). □

Theorem 12 *There is a proper-learning algorithm for $\mathcal{N}_m^d$ decision trees over constant dimension d, from membership and equivalence queries.*

Proof. We describe an algorithm for MINEQUIC_U where C is the class of $\mathcal{N}_m^2$ decision trees. The algorithm easily generalizes to other constant dimensions d. Let A be an $m \times m$ binary matrix. Each node v of the decision tree for A partitions a submatrix of A into two submatrices, which are passed to the left and right subtrees v. The partitioning continues until each submatrix contains only 0s, or only 1s, and the corresponding path in the decision tree terminates with a leaf. Define $S(a,b,x,y)$ to be the size (number of non-leaf nodes) of a minimum decision tree that partitions the submatrix with rows a through x and columns b through y. Then

$$S(a,b,x,y) = \begin{cases} 0 & \text{if the submatrix is all 0 or all 1,} \\ 1+\min\left\{ \begin{array}{l} \min_{a<i\leq x}(S(a,b,i-1,y)+S(i,b,x,y)), \\ \min_{b<j\leq y}(S(a,b,x,j-1)+S(a,j,x,y)) \end{array} \right\} & \text{otherwise.} \end{cases}$$

There are at most $O(m^4)$ submatrices, so the number of subproblems is polynomial, and $S(0,0,m-1,m-1)$ can be computed in time $O(m^5)$. The subproblems also provide information to build the minimal tree. For general d, the algorithm has time-complexity $O(dm^{2d+1})$ which is polynomial for constant d. □

Theorem 13 *There is a proper-PAC-learning algorithm for $\mathcal{N}_m^d$ decision trees for constant d.*

Proof. The same algorithm above will also solve MINEQUI$^\star C_U$. By Theorem 6 the result follows. □

4.2 DNF and Union of Boxes

In this subsection we give the results for DNF, union of boxes and disjoint union of boxes. We first prove the following.

Theorem 14 *There is an* $\varepsilon < 1$ *such that: If the class* $\mathcal{N}_m^n$ *DNF is* s^ε*-properly learnable with membership and equivalence oracles (and all the other oracles) where* s *is the size of the DNF, then* $\Sigma_2^P = P^{NP}$*.*

Proof. The oracles are NP-easy for $\mathcal{N}_2^n$ DNF and approximating MINEQUI $\mathcal{N}_2^n$ DNF within s^ε is Σ_2^P-hard (Umans, 1999). Then the result follows from Theorem 5. □

For union of boxes in a constant dimension we have the following.

Theorem 15 *For union of boxes over constant dimension* d *(*$\mathcal{N}_m^d$ *DNF) we have*

1. *There is a* $O(d\log t)$*-proper-learning algorithm for union of boxes over dimension* d *from membership and equivalence queries, where* t *is the optimal number of boxes required.*
2. *There is an* α *such that there is an* α*-proper-learning algorithm for union of boxes over dimension 2 from any set of oracles if and only if* $P = NP$*.*

Proof. Part 1 uses the fact that MINEQUI $\mathcal{N}_m^d$ DNF_U reduces to an instance of Set Cover of size $(2t)^{2d}$ as described by Bshouty et al. (1998), allowing the $O(\ln(2t)^{2d})$-approximation algorithm for Set Cover to be used. Part 2 uses the result that MINEQUI $\mathcal{N}_m^2$ DNF_U is NP-complete and does not admit an approximation scheme unless P=NP (Berman and DasGupta, 1992). Then both parts follow from Theorem 7 and Lemma 4. □

For union of disjoint boxes ($\mathcal{N}_m^n$ disjoint DNF) we have the next theorem.

Theorem 16 *We have*

1. *There is a proper-learning algorithm for union of disjoint boxes over dimension 2 from membership and equivalence queries.*
2. *There is a proper-learning algorithm for union of disjoint boxes over dimension 3 from membership and equivalence queries if and only if* $P = NP$*.*

Proof. This follows from Theorem 7 and Lemma 4, and the fact that MINEQUI $\mathcal{N}_m^d$ disjoint DNF_U is the same as the problem of partitioning a (set of) orthogonal polygons into a minimum number of boxes. This problem is in P for dimension $d = 2$ (Lipski Jr. et al., 1979), and NP-complete for dimension $d = 3$ (Dielissen and Kaldewaij, 1991). □

4.3 Multivariate Polynomials and XOR of Boxes

In this subsection we investigate the learnability of multivariate polynomials.

For multivariate polynomials with monotone terms we have the next theorem.

Theorem 17 *For any constant* d*, there is a proper-learning algorithm for* $\mathcal{N}_m^d$ *monotone multivariate polynomial from membership and equivalence queries.*

Proof. We give an algorithm to optimally solve the MINEQUI C_U problem where C is the class of $\mathcal{N}_m^2$ monotone multivariate polynomials (XOR of monotone rectangles). A monotone term $[x_1 \geq a][x_2 \geq b]$ in the polynomial is a rectangle with lower left corner (a,b), and covering all the points (c,d) with $(a,b) \prec (c,d)$ (that is, $a < c$ and $b \leq d$ or $a \leq c$ and $b < d$). Any nonzero input matrix $A = (a_{ij})$ over $\{0,1\}$ will have a point $p = (c,d)$ such that $A[p] = 1$ and $A[q] = 0$ for all $q \prec p$.

Then an optimal cover must use the monotone rectangle $R = [x_1 \geq c][x_2 \geq d]$ to cover p. We update the matrix by setting $A[i,j] = 1 - A[i,j]$ for all points (i,j) covered by R. Repeating the process until A is the zero matrix will yield the minimum number of rectangles. This algorithm for the 2-dimensional case generalizes to d dimensions for any constant d. $\square$

Before giving our result for multivariate polynomials with nonmonotone terms, we consider first "almost monotone" multivariate polynomials. An almost-monotone multivariate polynomial is a sum of terms that contain literals of the form $[x_i \geq \alpha]$ for $i = 1, \ldots, d$ and $[x_1 < \beta]$ (only x_1 may be negated).

Lemma 18 *There is a proper-learning algorithm for $\mathcal{N}_m^d$ almost-monotone multivariate polynomial from membership and equivalence queries.*

Proof. We consider the two dimensional case and note that it generalizes to d dimensions. An almost-monotone rectangle $[x_1 \geq a][x_1 < a'][x_2 \geq b]$ covers all points (c,d) with $a \leq c < a'$ and $d \geq b$. Suppose that we have two points (a,b), (a',b) such that $A[c,d] = 0$ for all (c,d) with $d < b$, $A[c,b] = 0$ for $c = a'$ and all $c < a$, and $A[c,b] = 1$ for $a \leq c < a'$. Then for $f = T_1 + T_2 + \cdots + T_t$ that represents A, we have $f = [x_2 \geq b]f$ and therefore without loss of generality, no term T_i contains a literal $[x_2 \geq d]$ with $d < b$.

Now consider the row of A that corresponds to $x_2 = b$. Let k be the number of transition points a for which $A[a,b] \neq A[a+1,b]$. Then f must have at least $\lceil k/2 \rceil$ rectangles T_i covering this row (since a rectangle has two vertical edges, it can cover just two transitions). So without loss of generality we may assume that consecutive 1s are covered by their own rectangle, and the points $(a,b) \leq (c,b) < (a',b)$ are covered by $R = [x_1 \geq a][x_1 < a'][x_2 \geq b]$ in an optimal cover.

So the algorithm works as follows. It finds the two points (a,b) and (a',b) as described above. It adds rectangle $R = [x_1 \geq a][x_1 < a'][x_2 \geq b]$ to the cover. It then toggles the matrix entries that are covered by R, and recurses. $\square$

Theorem 19 *There is a 2^{d-1}-proper-learning algorithm for $\mathcal{N}_m^d$ multivariate polynomial from membership and equivalence queries.*

Proof. To prove the theorem we show that every $\mathcal{N}_m^n$ multivariate polynomial f containing t terms can be written as a sum (XOR) of at most $2^{d-1}t$ almost-monotone $\mathcal{N}_m^n$ terms. Thus the algorithm to find an optimal-size almost-monotone $\mathcal{N}_m^n$ multivariate polynomial, is a 2^{d-1}-approximation algorithm for finding a $\mathcal{N}_m^d$ multivariate polynomial of minimum size. Let $f = T_1 + \cdots + T_t$ be a $\mathcal{N}_m^d$ multivariate polynomial. We change each term to a sum of almost-monotone terms as follows:

$$\begin{aligned} T_i &= \prod_{k=1}^{d} [x_k \geq a_{i,k}][x_k < b_{i,k}] \\ &= [x_1 \geq a_{i,1}][x_1 < b_{i,1}] \prod_{k=2}^{d} ([x_k \geq a_{i,k}] + [x_k \geq b_{i,k}]) \\ &= \sum_{c_2 \in \{a_{i,2}, b_{i,2}\}} \cdots \sum_{c_d \in \{a_{i,d}, b_{i,d}\}} [x_1 \geq a_{i,1}][x_1 < b_{i,1}][x_2 \geq c_2][x_3 \geq c_3] \cdots [x_d \geq c_d] \end{aligned}$$

So f is a sum of at most $2^{d-1}t$ almost-monotone $\mathcal{N}_m^n$ terms. $\square$

5. Conclusion and Open Problems

We have shown that for axis-parallel concept classes C with constant dimension, α-proper learning in the exact model with membership and equivalence queries, and α-proper PAC learning with just example queries, are roughly as hard as finding an α-approximation algorithm for the MINEQUIC problem. This allows us to apply some positive and negative results in approximation to the learning of axis-parallel concepts. For axis-parallel concept classes C with variable dimension, we show that an α-proper-exact-learning algorithm for C can be used to give an α-approximation algorithm for MINEQUIC. Thus any negative results for the α-approximation of MINEQUIC, gives a negative result for α-proper exact learning of C.

Several problems remain open. It is unknown whether one can properly-learn Disj-DNF with variable dimension, given the aid of a Σ_4^P oracle. We also do not yet know of any value of α such that Disj-DNF for dimension greater than two is not α-properly learnable unless $P = NP$. For multivariate polynomials with variable dimension, it remains open whether there is an α such that MP is not α-properly learnable unless $P = NP$.

Acknowledgments

While doing this research, Nader H. Bshouty was supported by the fund for promotion of research at the Technion, grant 120-025, and Lynn Burroughs was supported by an NSERC PGS-B Scholarship, an Izaak Walton Killam Memorial Scholarship, and an Alberta Informatics Circle of Research Excellence (iCORE) Fellowship. Part of this research was done while Nader H. Bshouty visited the University of Calgary, Calgary, Alberta, Canada and was supported by NSERC of Canada. We would like to thank the anonymous reviewers for their careful reading of this paper, and their suggestions for improving its readability.

References

D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

A. Beimel, F. Bergadano, N.H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM*, 47(3):506–530, 2000.

A. Beimel and E. Kushilevitz. Learning boxes in high dimension. *Algorithmica*, 22(1/2):76–90, 1998.

S. Ben-David, N.H. Bshouty, and E. Kushilevitz. A composition theorem for learning algorithms with applications to geometric concept classes. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 324–333, 1997.

F. Bergadano, D. Catalano, and S. Varricchio. Learning sat-k-DNF formulas from membership queries. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 126–130, 1996.

P. Berman and B. DasGupta. On approximating the rectilinear polygon cover problems. In *Proceedings of the 4th Canadian Conference on Computational Geometry*, pages 229–235, 1992.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, 1987.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

N. Bshouty. A new composition theorem for learning algorithms. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 583–589, 1998.

N.H. Bshouty. Exact learning Boolean functions via the monotone theory. *Information and Computation*, 123(1):146–153, 1995.

N.H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, 1996.

N.H. Bshouty, P.W. Goldberg, S.A. Goldman, and D.H. Mathias. Exact learning of discretized geometric concepts. *SIAM Journal of Computing*, 28(2):674–699, 1998.

Z. Chen and S. Homer. The bounded injury priority method and the learnability of unions of rectangles. *Annals of Pure and Applied Logic*, 77(2):143–168, 1996.

Z. Chen and W. Maass. On-line learning of rectangles and unions of rectangles. *Machine Learning*, 17(2 and 3):201–223, 1994.

V. Dielissen and A. Kaldewaij. Rectangular partition is polynomial in two dimensions but NP-complete in three. *Information Processing Letters*, 38(1):1–6, 1991.

D. Haussler, M. Kearns, N. Littlestone, and M. Warmuth. Equivalence of models for polynomial learnability. *Information and Computation*, 95(2):129–161, 1991.

L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan, and D. Wilkins. How many queries are needed to learn? *Journal of the ACM*, 43(5):840–862, 1996.

L. Hellerstein and V. Raghavan. Exact learning of DNF formulas using DNF hypotheses. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, 2002.

A. Klivans and R. Servedio. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 258–265, 2001.

W. Lipski Jr., E. Lodi, F. Luccio, C. Mugnai, and L. Pagli. On two-dimensional data organization II. *Fundamenta Informaticae*, 2:245–260, 1979.

W. Maass and M.K. Warmuth. Efficient learning with virtual threshold gates. *Information and Computation*, 141(1):66–83, 1998.

K. Pillaipakkamnatt and V. Raghavan. On the limits of proper learnability of subclasses of DNF formulas. *Machine Learning*, 25(2 and 3):237–263, 1996.

R.E. Schapire and L.M. Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory (COLT)*, pages 17–26, 1993.

C. Umans. Hardness of approximating Σ_2^p minimization problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 465–474, 1999.

H. Zantema and H. Bodlaender. Finding small equivalent decision trees is hard. *International Journal of Foundations of Computer Science*, 11(2):343–354, 2000.