On Learning Embedded Symmetric Concepts

Avrim Blum*
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
avrim@cs.cmu.edu

Prasad Chalasani
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
chal@cs.cmu.edu

Jeffrey Jackson†
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
jcj@cs.cmu.edu

Abstract

An embedded symmetric concept is a concept symmetric on the set of relevant variables. The class of such concepts is a natural extension of several fundamental classes including Boolean thresholds, parity, and monotone monomials. In this paper, we characterize which types of embedded symmetric concepts are learnable in a representation-dependent sense, and which are not. In particular, we show that among "reasonable" symmetric concept subclasses only a very small set—constant, conjunction, disjunction, parity, consensus, and their negations—are PAC-learnable in a representation-dependent sense given standard complexity assumptions. We also address the problem of prediction, and provide a weak hardness result for prediction based on the difficulty of predicting a subclass of DNF formulas. On the positive side, we present prediction methods for certain subclasses of these concepts, including the mod$p$ concepts. We also show that many classes of embedded symmetric concepts have the property that over the uniform distribution, they are either strongly learnable or else indistinguishable from random.

In addition, we define a more general class of multi-symmetric concepts and present an exact learning algorithm with membership queries for this class.

*Supported in part by an NSF Postdoctoral Fellowship.
†Supported in part by NSF Grant CCR-9119319.

ACM COLT '93 /7/93/CA, USA
© 1993 ACM 0-89791-611-5/93/0007/0337...$1.50

1 Introduction

Symmetric concepts depend only on the number of variables set to 1. We call a concept an *embedded symmetric concept* if it is symmetric on the subset of variables that are relevant. (One can think of such a concept as a symmetric concept defined on a smaller domain, embedded into a larger one.) Embedded symmetric concepts are in some sense very simple. A key feature that makes them interesting is that "learning is as easy as finding the relevant variables." That is, if the set of relevant variables is known then such concepts are easy to learn in either an exact or approximate sense. On the other hand, the learning situation changes dramatically if the set of relevant variables is unknown to the learner.

The class C_{sym} of embedded symmetric concepts, in addition to being a natural "semantically defined" class, includes many of the fundamental concept classes studied in computational learning theory. For example, conjunctions, disjunctions, Boolean thresholds and parity functions are all subclasses of C_{sym}. Bshouty, Hancock, and Hellerstein [BHH92] recently extended earlier results on learning read-once formulas to show that such concepts are learnable with membership queries over a basis including all symmetric functions. This suggests that useful generalizations of learning results may be possible by considering symmetric concepts as a class rather than individually.

Kearns, Li, Pitt, and Valiant [KLPV87] show that Boolean thresholds cannot be PAC-learned (in terms of Boolean thresholds) if $RP \neq NP$. One of our results is a characterization of which subclasses of embedded symmetric concepts are learnable and which are not under this same complexity assumption. We show that embedded symmetric concept classes based on conjunction, disjunction, parity, consensus, their negations, and the constant functions are essentially[1] the only embedded symmetric concept classes which are PAC-learnable in terms of embedded symmetric concepts. We also give a weak representation-independent hardness result, showing that the class C_{sym} is as hard for prediction as a sub-

[1]The "essentially" deals with the difficulty of non-uniform classes such as "monomials ∪ {parity with 57 relevant variables}".

class of DNF formulas—the functions of $\log(n)$ relevant variables—for which prediction seems difficult.

In addition to the above negative results, by restricting the concept class in various ways we are able to give several positive results. We present an algorithm for predicting mod$p$ functions. We also show how to predict concepts which, when viewed as a unary function on the sum of their inputs, change values only a constant number of times or which are periodic with period a constant prime. Additionally, we show that weak learnability of certain subclasses of C_{sym} with respect to the uniform distribution implies strong learnability with respect to the uniform distribution.

Finally, we consider a generalization of embedded symmetric concepts which we call multi-symmetric concepts. A multi-symmetric concept c is defined by specifying (a) a collection of disjoint subsets of the input bits, and (b) a Boolean function f which takes as its arguments the number of 1's in each subset. The value of c on input x is the value of f when given the number of 1's in x in each set. Let C_{multi} be the representation class of these concepts in which the function f is represented by a lookup table. We give an algorithm for C_{multi} that learns with equivalence and membership queries and runs in time polynomial in n and the size of c's smallest C_{multi} representation.

Note that multi-symmetric concepts are different from read-once formulas because f takes in the number of 1's in each set, not just a bit defined by some other function g. For instance, if f has two inputs, it might output 1 exactly when the two inputs are equal (corresponding to a concept c that specifies two disjoint sets of inputs, and outputs "positive" when an instance has the same number of 1's in each set).

2 Definitions and Notation

A *concept* c is a Boolean function on an *instance space* X. A *concept class* C is a set of concepts. An *instance* x is an element of the instance space (here $\{0,1\}^n$), and we use x_i to denote the ith bit of x. We will also think of an instance as an assignment of values to the n variables $v_1, v_2, \ldots, v_n$; so in this case, x_i represents the value assigned to v_i by x. A pair $\langle x, c(x)\rangle$ is called an *example* (or for emphasis a *labeled example*) of the concept $c \in C$; the $c(x)$ portion is called the *label*. An example labeled 1 (resp. 0) is called a *positive* (resp. *negative*) example.

Let $[k]$ represent the set $\{0,\ldots,k\}$. A concept c over $\{0,1\}^n$ is *symmetric* if there is a function $f_n : [n] \rightarrow \{0,1\}$ such that $c(x) = f_n(\sum_i x_i)$ for all $x \in \{0,1\}^n$. In other words, c is a function of only the number of 1's in the input. A concept c is *embedded symmetric* if there exists some $R \subseteq \{1,\ldots,n\}$ and a function $f_r : [r] \rightarrow \{0,1\}$ such that $|R| = r$ and $c(x) = f_r(\sum_{i\in R} x_i)$ for all $x \in \{0,1\}^n$. For instance, if c is a conjunction of r variables, then R is the set of the r relevant variables and f_r is the function that outputs 0 on inputs $0, 1, \ldots, r-1$ and 1 on input r. C_{sym} is the class of all embedded symmetric concepts. The variables indexed by the elements of R are the *relevant* variables of the concept, and we frequently treat R as the set of these variables rather than a set of indices. We call f the *basis* function of the (embedded) symmetric concept, and a subscript to f will indicate the size of its domain. A basis function f (relevant set R) is *feasible* for a set of examples if there is some embedded symmetric concept with basis f (relevant set R) consistent with the examples. A *pattern* of a basis function f is a boolean string that represents the values of f at consecutive elements of its domain. For example, if $f(i) = 0$, $f(i+1) = 0$, and $f(i+2) = 1$ then 001 is a pattern of f and we will say that the pattern begins at i. Every pattern of length $r+1$ defines in a natural way a basis function f_r.

A *basis family* F is a set of basis functions f_r, one for each $r \in Z^+$. At times we refer to a basis function f_r as a *basis function on r relevant variables*. We consider only basis families F which have a polynomial-time representation; that is, there is a polynomial-time Turing machine M_F such that for all $f_r \in F$ and all $i \in [r]$, $M_F(1^r, i) = f_r(i)$.

Consensus is the function which has value 1 when all its inputs are zero or all its inputs are one, and value 0 otherwise. The functions conjunction, disjunction, parity, consensus, their negations, and the two constant functions each define a basis family in a natural way. We define the set of *simple* basis functions to be the collection of all the functions in all ten of these basis families. All other basis functions are *non-simple*. If an embedded symmetric concept has a (non-)simple basis function then the concept is (non-)simple. A basis family F is simple if every basis function $f_r \in F$ is simple. F is *polynomially non-simple* if there is a fixed integer polynomial q such that for each $f_r \in F$ there is a non-simple function $f_t \in F$ for $r \leq t \leq q(r)$.

The *concept class based on family* F, denoted C_F, is the set of all embedded symmetric concepts with basis $f_r \in F$ for some r. C_{simple} is the class of all embedded symmetric concepts having a simple basis function.

Every concept c can be represented by a collection of disjoint subsets $S_1, S_2, \ldots, S_k$ of $\{1,\ldots,n\}$ and a function $f : [r_1] \times [r_2] \times \ldots \times [r_k] \rightarrow \{0,1\}$ where $r_i = |S_i|$, with $c(x) = f(\sum_{i\in S_1} x_i, \sum_{i\in S_2} x_i, \ldots, \sum_{i\in S_k} x_i)$ for all $x \in \{0,1\}^n$. If c is defined by S and f as above, the *multi-symmetric* representation of c is any reasonable representation of the sets S_i along with a lookup table with $N = (r_1+1)(r_2+1)\ldots(r_k+1)$ entries of the form $\langle w_1, w_2, \ldots, w_k, b\rangle$, where $f(w_1, w_2, \ldots, w_k) = b$. The class of these representations is denoted C_{multi}.

An algorithm A (possibly randomized) *probably approximately correctly (PAC) learns* a concept class C if there exists a polynomial p such that for any $\epsilon, \delta > 0$, any $c \in C$ and any distribution D on labeled examples of c, in time $p(n, 1/\epsilon, 1/\delta)$ A produces with probability at least $1-\delta$ a concept $h \in C$ such that $\Pr_D[h(x) \neq c(x)] \leq$

ϵ. We will also refer to classes PAC-learnable with respect to a particular distribution D. If the hypothesis concept h is allowed to come from a concept class C' (that may or may not equal C) then A is said to *PAC learn C in terms of C'*. If C can be learned in terms of polynomial-time algorithms, then C is *polynomially predictable* (which we refer to simply as predictable).

3 Learning Simple Symmetric Concepts

We will later show that PAC-learning C_F is NP-hard for every polynomially non-simple family F even if learning is in terms of C_{sym}. First, we establish a positive result for representation-dependent learning of C_{simple}.

Theorem 1 *C_{simple} is PAC-learnable.*

Proof: It is sufficient to find a simple embedded symmetric concept consistent with any set of examples produced by a target concept $c \in C_{simple}$. There are standard learning algorithms for the constant, conjunction, disjunction, and parity concept classes and their negations ([Val84], folklore). We give an algorithm for the consensus class below; an algorithm for negated consensus follows immediately. The algorithm for learning C_{simple} then consists of running each of these subclass algorithms in turn until one returns a consistent hypothesis.

Learning consensus is a variation on monomial learning. Given a set of examples of a consensus concept, all variables in the relevant set R are assigned the same value in every positive example. So if variable v_i is relevant, the set of variables which have the same value as v_i in every positive example form a set $R_i \supseteq R$. Thus the consensus concept c_i defined by R_i will be consistent with all of the negative examples as well as the positive ones.

A simple consensus algorithm, then, consists of selecting in turn each variable v_i and testing the corresponding c_i for consistency with the negative examples. This process stops when a consistent c_i is found or after all variables v_i have been tried, in which case the examples are not consistent with any consensus concept. This procedure runs in polynomial time. ■

4 Representation-Dependent Hardness of Learning Non-simple Symmetric Concepts

We now prove that no class C_F defined by a polynomially non-simple basis family F can be PAC-learned in terms of C_F unless $RP = NP$. We then extend this to show that such C_F cannot even be learned in terms of the entire class C_{sym}. As in Kearns et al. [KLPV87], the results follow by showing that a related consistency problem is NP-hard.

Definition 2 *The* F-consistency (F-Con) *problem is: Given a set of examples, is there a symmetric concept $c \in C_F$ such that for each example $\langle x, l(x)\rangle$ in the set, $c(x) = l(x)$?*

In the remainder of this section, we show that F-Con is NP-hard for every polynomially non-simple F, and then relate this result to the learnability of embedded symmetric concepts.

4.1 Hardness of F-Consistency

The one-in-three 3SAT question is: given a 3CNF formula, is there an assignment which satisfies exactly one literal in each clause of the 3CNF? This problem is NP-hard [GJ79]. We will show how to reduce one-in-three 3SAT to F-Con for any polynomially non-simple basis family F.

A key fact we exploit in the reduction is that non-simple basis functions are characterized by a small set of patterns. In particular, every non-simple basis function must contain one of the patterns given in the definition below.

Definition 3 *The set*

$$\{0100, 1011, 0010, 1101, 0011, 1100\}$$

is the set of non-simple patterns.

Lemma 4 *A basis function f_r on $r \geq 3$ relevant variables is non-simple if and only if it contains at least one of the non-simple patterns.*

Proof: We prove the equivalent claim that a basis function on at least three variables is simple if and only if all of its 4-bit patterns are simple.

The only-if direction is straightforward. The other direction can be proved by induction on r. The base case for $r = 3$ is immediate. Now assume that the proposition holds for all values of r smaller than r_0, let f be a basis function on r_0 relevant variables, and let s be the $(r_0 + 1)$-bit pattern defining f. Then if all of the 4-bit patterns of f are simple, the two basis functions f_R and f_L defined by dropping either the the first or last bit of s, respectively, are simple by the inductive assumption. A case analysis confirms that if f_L and f_R are simple then f must be as well. ■

The key use of the non-simple patterns is that any one will be sufficient to achieve the required reduction. From this it follows that the F-Consistency problem is NP-hard for any F containing only non-simple basis functions and hence for any polynomially non-simple F.

Lemma 5 *F-Con is NP-hard for any basis family F in which every basis function $f_r \in F$ for $r \geq 3$ is non-simple.*[2]

[2] All basis functions on fewer than three relevant variables are simple.

Proof Sketch: (For a full proof, see the Appendix.) As indicated above, the proof is based on reduction from one-in-three 3SAT. Given any 3CNF formula g on n variables and a basis family F we generate a set S_1 of examples on $2n+2$ variables—two variables x_i and $\bar{x}_i$ for each variable v_i in g plus two auxiliary variables. The examples have the property that every feasible relevant set R contains both auxiliaries plus exactly one of each pair of variables $(x_i, \bar{x}_i)$, so the only feasible basis in F is f_{n+2}. A case analysis of each possible non-simple pattern beginning at each possible position within the non-simple function f_{n+2} shows that an additional set S_2 of examples can be efficiently produced such that $S_1 \cup S_2$ achieves the desired reduction. ■

The reduction above can be modified to produce examples on $2n+k$ variables for which every feasible relevant set must contain $n+k$ variables, $k \geq 2$. It is also easy to modify the examples in S_2 so that they apply to f_{n+k} rather than to f_{n+2}. Thus if a basis family F includes simple basis functions but has a non-simple basis function "not too far away" from every simple basis function then F-Con is still hard. That is:

Theorem 6 *F-Con is NP-hard for every polynomially non-simple basis family F.*

4.2 Learning in Terms of Symmetric Concepts

We now relate the hardness of F-Consistency to the learnability of symmetric concepts.

Theorem 7 *C_{sym} is not PAC-learnable by C_{sym} unless $RP = NP$. Furthermore, for any fixed polynomially non-simple family F, C_F is not PAC-learnable either in terms of C_F itself or in terms of C_{sym} unless $RP = NP$.*

Proof: First, it follows immediately from Theorem 6 and Kearns et al. [KLPV87] that if $RP \neq NP$ then C_F is not PAC-learnable in terms of C_F for any polynomially non-simple family F.

To see that C_F is also not learnable in terms of C_{sym} under this assumption, consider another reduction from one-in-three 3SAT. Given 3CNF g and polynomially non-simple F, a set of examples S_1 on $2n+k$ variables is constructed as in the reduction of Lemma 5, establishing that $n+k$ of the variables are relevant, that every feasible relevant set contains exactly one of each pair $(x_i, \bar{x}_i)$, and that $f_{n+k} \in F$ is feasible for S_1. Next, a set of $n+k+1$ labeled examples S_1' is produced in which each instance assigns successively more 1's to the relevant variables. By choosing the labels of the examples in S_1' to be consistent with f_{n+k}, the *only* feasible basis for the set of examples $S_1 \cup S_1'$ is f_{n+k}. Finally, a set S_2 corresponding to the like-named examples of Lemma 5 is created. Now if there is a one-in-three satisfying assignment for g then there is a symmetric concept consistent with $S = S_1 \cup S_1' \cup S_2$ and any such concept has basis in F, while if there is no such assignment then there is no symmetric concept consistent with S.

Now assume that C_F is PAC-learnable in terms of C_{sym} for some polynomially non-simple F. By reasoning similar to Kearns et al. this implies that there is a (possibly randomized) polynomial-time algorithm which can find a symmetric concept consistent with any set of examples for which there is some concept $c \in C_F$ consistent with the examples. But we have just shown that every 3SAT g can be reduced in polynomial time to a set S which is either consistent with some $c \in C_F$ if g has a one-in-three satisfying assignment or is not consistent with any symmetric concept otherwise. Thus one-in-three 3SAT is in RP if C_F is PAC-learnable in terms of C_{sym}.

We immediately also get that C_{sym} is likewise not PAC-learnable in terms of C_{sym} if $RP \neq NP$. ■

5 Representation-Independent Learning

While the theorem above indicates that learning symmetric concepts in terms of symmetric concepts is apparently hard, it may be that C_{sym} can be predicted in terms of some other class. We know of no such general prediction algorithm for C_{sym}. In fact, even predicting C_{sym} with respect to the uniform distribution is an open problem. Fourier techniques, which are often helpful in learning with respect to uniform or product distributions [LMN89, FJS91], do not immediately apply to symmetric concepts because there are symmetric concepts with exponentially small Fourier coefficients throughout. We do, however, have prediction algorithms for several subclasses of C_{sym} which are outlined below. First, we consider a weak representation-independent hardness result which provides some evidence that predicting C_{sym} may be difficult.

5.1 A Representation-Independent Hardness Result

A variable v_i is *relevant* in a concept c if there exist instances x and y that differ in only their ith bits and $c(x) \neq c(y)$. The class C_{log} of *$\log n$-relevant-variable* concepts is the class of all concepts c which have $\log n$ relevant variables. This class is a subclass of the set of polynomial-size DNF formulas since the truth-table defining such a concept has only n entries. It is also a subclass for which predictability seems difficult. We now show that C_{log} is predictable if C_{sym} is predictable.

Let c be a $\log n$-relevant-variable concept. From each instance $x \in \{0,1\}^n$ generate a new instance $x' \in \{0,1\}^{n^2}$ which replicates each x_i n times. Consider a set V' of n^2 variables corresponding to the bits of x', and let v_i' represent the set of n variables in V' associated with x_i. Let R be the relevant variables of c, and assume without loss of generality that $R = \{1, \ldots, \log n\}$. Define the relevant variables $R' \subset V'$ to be the set containing one of the variables in v_1', two of the variables in v_2', and in general 2^{i-1} variables in v_i', $1 \leq i \leq \log n$. Then there is an embedded symmetric concept c' with relevant set

R' such that for all x, $c(x) = c'(x')$. Thus the examples x' represent examples of a symmetric concept, so a prediction algorithm for C_{sym} can be used to predict C_{log}.

5.2 Constant Flips

The number of times that the value of a basis function changes as its input increases is termed the number of *flips* of the basis function and of its corresponding symmetric concept. It is not hard to see that the class of symmetric concepts based on the set of all basis functions having at most some constant k number of flips is predictable, since if there are k flips in embedded symmetric concept c with basis f then there must be some degree k polynomial P in the sum of the relevant variables such that $\text{sign}(P) = f$ In other words, the concept can be written as a linear threshold function over the set of conjunctions of $\leq k$ variables. Since thresholds can be learned in time polynomial in the number of variables, the result follows.

5.3 mod$p$ Concepts

The mod$p$ concepts have value 0 when the sum of the relevant variables is congruent to 0mod$p$, and value 1 otherwise. We show that the class of all such concepts for any prime p can be learned in the mistake bound model [Lit88] with at most n mistakes. The idea is to set up a linear system much as is done in the standard algorithm for learning parity, except in this case only negative examples are incorporated into the system. That is, if c is a mod$p$ concept then there is some 0-1 vector w such that for every negative instance x, $(w \cdot x) \text{mod} p = 0$. w defines the relevant variables of c in the obvious way, so learning the function reduces to learning w, which in turn reduces to obtaining a set of n linearly independent (mod$p$) negative instances.

At any point in the execution of the algorithm, the label of a test instance is predicted by first checking to see if the instance is a linear combination of a set S of linearly independent negative instances. If so, predict negative, otherwise positive. If the prediction is wrong (which can only occur if the prediction is positive), add this instance to S.

We can generalize this result as follows. Let m be a product of distinct primes p_i, $1 \leq i \leq k$, and let c be a mod$m$ concept. Then $c(x) = 0$ if and only if instance x assigns 0modp_i 1's to the relevant variables for each p_i. Thus to learn the class of mod$m$ concepts for such an m we maintain a collection of sets S_i of instances, one set for each p_i. A test instance is labeled negative if for each i it is a linear combination modp_i of the instances in S_i, otherwise it is labeled positive. Each instance which is incorrectly labeled positive is added to the S_i's which predicted positive. This procedure makes at most kn mistakes.

5.4 Concepts with Constant Prime Period

A symmetric function f has *prime period* if there is some prime p and some pattern s with bits $s_0, \ldots, s_{p-1}$ such that the pattern of f is simply an appropriate number of repetitions of s, truncating the final s as needed. The mod$p$ concepts in the previous subsection are clearly a special case of these. As in that subsection, suppose w is the 0-1 vector defining the relevant variables. Then it is not hard to see that the polynomial equation

$$P^+(x) \equiv \prod_{i<p, s_i=1} (w \cdot x - i) = 0 \text{mod} p$$

is satisfied by an instance x if and only if it is positive. The negative instances are captured by a similar polynomial. Thus, the learning problem is reduced to one of solving polynomial equations of degree at most $(p-1)$ for w. For constant p, we can linearize these equations in the standard way (i.e., introduce new variables that represent products of the x_i), and then efficiently PAC-learn or predict, along the lines of the standard algorithms for learning and predicting parity.

6 On Learning over the Uniform Distribution

We do not know whether or not the class of symmetric concepts is learnable or predictable over the uniform distribution (without queries). However, we can show the following interesting fact. Let $C_{sym,unbiased}$ be the class of symmetric concepts such that the probability a random example is positive is $\frac{1}{2}$. Then, if $C_{sym,unbiased}$ is weakly predictable over the uniform distribution, it is also strongly learnable from the uniform distribution. In fact, all naming-invariant subclasses of $C_{sym,unbiased}$ have this property as well. (A class is naming-invariant if it is closed under permutations of variable names [Kea89].) That is, if a naming-invariant $C \subseteq C_{sym,unbiased}$ is weakly-learnable over the uniform distribution then it is also strongly learnable.

We prove our result by showing how to find all the relevant variables of the target concept. It is clear that if we can do so, we can then fill in a lookup table from random examples to achieve arbitrarily low error.

Theorem 8 *If $C \subseteq C_{sym,unbiased}$ is naming-invariant and weakly-learnable over the uniform distribution then it is also strongly learnable over the uniform distribution.*

Proof. Let $\mathcal{A}$ be an algorithm that weak-learns the class C over the uniform distribution. For simplicity, we will combine the confidence and error parameters and just say that for some $m = poly(n)$, with probability at least $\frac{1}{2} + \frac{1}{P(n)}$ (for some polynomial P), $\mathcal{A}$ will predict correctly on the next instance after seeing m random labeled examples. If S is a set of $m+1$ examples, we say that $\mathcal{A}$ *succeeds* on S if after seeing the first m labeled

examples and the final unlabeled instance x, $\mathcal{A}$ correctly predicts the label of x.

The basic idea of the proof is this. The probability $\mathcal{A}$ succeeds on a random set S of $m+1$ examples is $\geq \frac{1}{2} + \frac{1}{P(n)}$. Suppose, however, before feeding the examples to $\mathcal{A}$, we corrupt them by replacing the first i bits of each example with random bits. As we increase i to n, at some point $\mathcal{A}$'s performance must degrade, and this will occur at a relevant variable. Since the concept is symmetric, all relevant variables are "equally relevant" so we can find the other relevant variables through similar means.

More specifically, given a set S of $m+1$ examples and a set I of indices, let $corrupt(S, I)$ be the set of examples S but where for each x in S, x_i has been replaced by a random bit for each $i \in I$. Let p_i be the probability $\mathcal{A}$ succeeds on $corrupt(S, \{1, \ldots, i\})$ when S is randomly drawn. Let p_0 be the probability $\mathcal{A}$ succeeds on S. We know $p_0 \geq \frac{1}{2} + \frac{1}{P(n)}$. In addition, $p_n = \frac{1}{2}$ because the first m examples give no information (the label of x is 50/50 1 or 0 independent of x) and the test example is a random string independent of its label as well.[3]

So, some p_{i-1} and p_i must differ by at least $\frac{1}{nP(n)}$. By repeatedly drawing new sets S we can estimate each p_i to sufficient accuracy so that with high probability, all pairs (p_{i-1}, p_i) that appear to differ by at least $\frac{1}{2nP(n)}$ are truly different and we do find one such pair. If (p_{i-1}, p_i) is such a pair, it must be that x_i is relevant because if x_i is irrelevant, the distributions $corrupt(S, \{1, \ldots, i-1\})$ and $corrupt(S, \{1, \ldots, i\})$ over random S are the same. Thus, we find one relevant variable.

We can find the remaining relevant variables as follows. We pick a random permutation ϕ on the indices and repeat the above procedure with the bits of each example rearranged according to ϕ. Since the class C is naming-invariant, these examples are consistent with a concept in C. So we will again find some relevant variable, which will be a random one independent of x_i. If we repeat this $O(n \log(n/\delta))$ times, we will with probability at least $1 - \delta$ find all the relevant variables. As noted above, once we have the relevant variables, we can easily learn to any desired accuracy. ■

This result extends to all naming invariant subclasses of C_{sym} having the property that every concept in the class has the *same* probability of a random example being positive (but not necessarily $\frac{1}{2}$), if we replace the notion of weak learning by the following. Say p is the probability a random example is positive for concepts in the class; then, the probability of correct prediction should be at least $[\max\{p, 1-p\}](1 + 1/poly(n))$. That is, a weak learner must do noticeably better than either constant hypothesis **1** or **0**. The reason this works is that $\max\{p, 1-p\}$ is the largest value possible for p_n in the proof above. Of course, one could improve this further by noticing that if p is sufficiently close to 1 or 0, then simply hypothesizing one of the two constant functions *is* strong learning.

[3] It is important that we corrupt the test example in addition to the "training examples" because some concept classes may be weakly learnable even with zero training examples.

7 Learning Multi-Symmetric Functions

Recall that a multi-symmetric concept c on n variables is specified by a collection of disjoint subsets $\{S_1, S_2, \ldots, S_k\}$ of $\{1, 2, \ldots, n\}$ and a function f such that the value of c on an instance x is $f(w_1, w_2, \ldots, w_k)$ where w_i is the number of bits of S_i that are 1 in x. The representation class C_{multi} represents such concepts by using a lookup table for f. The lookup table has $N = (|S_1|+1)(|S_2|+1)\ldots(|S_k|+1)$ entries of the form $\langle w_1, w_2, \ldots, w_k, b\rangle$, where $f(w_1, w_2, \ldots, w_k) = b$. While a concept may have several multi-symmetric representations, it has a unique representation with minimal N, and this is the representation we assume from now on.

We show that C_{multi} can be exactly identified with membership and equivalence queries in time polynomial in n and N.

We give here a high level view of the algorithm with further details in the appendix. We first introduce some useful notation. For clarity, we will use upper-case letters to denote instances in $\{0,1\}^n$. For an instance $X \in \{0,1\}^n$ and a subset S of $\{1, 2, \ldots, n\}$, let X^S denote a vector in $\{0, 1, *\}^n$, whose ith component equals X_i if $i \in S$ and equals $*$ otherwise. We write $|X|$ to denote the number of ones in a vector $X \in \{0, 1, *\}^n$.

The algorithm maintains a set $R \subseteq \{1, 2, \ldots, n\}$ of indices found to be relevant so far. It also maintains a partitioning of R into *blocks* $B_1, B_2, \ldots, B_u$. At all times, the algorithm maintains the invariant that each of the B_i is a union of one or more of the (disjoint) sets $S_1, S_2, \ldots, S_k$. When the algorithm terminates, R will contain k blocks and each block equals a different set S_i.

With the current R and current partitioning $B_1, B_2, \ldots, B_u$, the algorithm constructs a hypothesis based on the assumption that R is in fact the set of all the relevant indices, and that the blocks B_i are equal to single sets S_j. The algorithm builds a table with $M = (|B_1|+1)(|B_2|+1)\ldots(|B_u|+1)$ entries of the form $\langle a_1 a_2 \ldots a_u, b\rangle$. For each such entry, the algorithm sets the b value equal to the answer obtained by making a membership query X where for each i, $|X^{B_i}| = a_i$. In all membership queries X, the bit X_j is set to 0 for $j \notin R$. Thus the hypothesis-building portion of the algorithm makes M membership queries and it is easy to see that $M \leq N$.

Once the hypothesis is built, the algorithm makes an equivalence query with this hypothesis. If the equivalence query thus constructed is answered "yes", then the algorithm halts. Otherwise a counterexample Z is obtained which contradicts some table entry. In other

words, we now know two instances Y and Z such that $|Y^{B_i}| = |Z^{B_i}|$ for all i, and yet Y and Z are classified differently. Clearly this means either that some B_i is in fact a union of two or more of the sets S_i, or that there are relevant indices outside R. To discover which of these is the case, the algorithm makes a sequence (described below) of membership queries "walking" from Y toward Z until a vector is found whose classification differs from that of Y.

For $i \in \{1, 2, \ldots, n\}$, $X \in \{0,1\}^n$, we write $X[i]$ to denote a vector in $\{0,1\}^n$ identical to X everywhere except at index i. The vectors $X[ij]$ and $X[ijk]$ are defined similarly. Then we can define the sequence of membership queries as follows. Let X denote the previous membership query made, where initially, we set $X = Y$. Subsequently, the next vector is constructed from X as follows. There are two cases to consider:

(a) For all $i = 1, 2, \ldots, u$, $X^{B_i} = Z^{B_i}$. In this case, pick the smallest index j where X and Z differ (this j must be outside of R), and define the next vector to be $X[j]$. If the classification of this new vector differs from that of X, this clearly means that there are relevant indices (in particular, j) outside R. In this case, the procedure Grow(X, X_j, R) makes $O(n^2)$ membership queries, finds a set R' that strictly contains R, and guarantees that $R' - R$ is the union of one or more of the sets S_i. The algorithm then redefines R to be R'. We then repeat the entire procedure.

(b) Otherwise, let s be the smallest index such that $X^{B_s} \neq Z^{B_s}$. Then pick a pair $j, k \in B_s$ such that $X_j = 1, Z_j = 0$, and $X_k = 0, Z_k = 1$. Such a pair must exist since $|X^{B_s}| = |Z^{B_s}|$. The next membership query is then $X[jk]$. Note that $|X[jk]^{B_i}| = |X^{B_i}|$ for all i. Thus if the classification of this new vector differs from that of X, it must be the case that j and k belong to different sets S_i, so B_s is a union of two or more of the sets S_i. In this case, the procedure Split(X, B_s, j, k) makes $O(n^3)$ membership queries and finds a partition of B_s into two non-empty sub-blocks, each of which is guaranteed to be a union of one or more of the sets S_i. Thus R will now have one more block than before, and the entire procedure is repeated.

From the preceding description, it is clear that each time an equivalence query is answered with a counterexample, one of the procedures Grow or Split is invoked, and the number of blocks in R increases by 1. Because of the invariants maintained by Grow and Split, this means that there will be no more than k equivalence queries before the algorithm finds the right hypothesis. So from the preceding remarks, the total number of membership queries is $O(k(n^3 + N))$.

Thus we are able to show:

Theorem 9 *There is an exact learning algorithm for learning multi-symmetric concepts over k disjoint sets that makes at most k equivalence queries and $O(k(n^3 + N))$ membership queries, where N is the number of entries in the smallest table specifying the concept.*

A detailed description of the procedures Grow and Split is in the Appendix.

8 Further Research

There are a number of questions left open by this research. Ultimately, we would like to know whether or not the class of embedded symmetric concepts is predictable. We do not have an answer to this question even when it is restricted to predicting with respect to the uniform distribution. In the mean time, it may be helpful to find additional subclasses of C_{sym} which are predictable.

Another interesting question concerns the learning of a generalization of multi-symmetric concepts which drops the pairwise-disjoint requirement on the relevant variable subsets. Call such concepts *overlapping symmetric*. While we can learn a representation of such concepts using the algorithm for multi-symmetric concepts, in the worst case this may require super-polynomial time in the size of target overlapping symmetric concept. Is there a polynomial-time algorithm for this class?

References

[BHH92] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. Learning boolean read-once formulas with arbitrary symmetric and constant fan-in gates. In *Fifth Annual Workshop on Computational Learning Theory*, pages 1–15, 1992.

[FJS91] Merrick L. Furst, Jeffrey C. Jackson, and Sean W. Smith. Improved learning of ac^0 functions. In *Fourth Annual Workshop on Computational Learning Theory*, pages 317–325, 1991.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[Kea89] Michael Kearns. *The Computational Complexity of Machine Learning*. PhD thesis, Harvard University Center for Research in Computing Technology, May 1989. Technical Report TR-13-89. Also published by MIT Press as an ACM Distinguished Dissertation.

[KLPV87] Michael Kearns, Ming Li, Leonard Pitt, and Leslie Valiant. On the learnability of boolean formulae. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 285–295, 1987.

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[LMN89] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. In *30th Annual Symposium on Foundations of Computer Science*, pages 574–579, 1989.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

Appendix

Proof of Lemma 5:

We reduce from one-in-three 3SAT. Given a 3CNF g on n variables, we first create a set S_1 of examples on $2n+2$ variables y_1, y_2, x_i, and $\bar{x}_i$, $i = 1, \ldots, n$, such that any symmetric concept consistent with the examples must have exactly $r = n + 2$ relevant variables and such that exactly one variable in each pair $p_i = (x_i, \bar{x}_i)$ is relevant in any feasible relevant set. We then create additional examples S_2 which further restrict the feasible relevant set, so that there is a symmetric concept consistent with the entire set of examples $S_1 \cup S_2$ if and only if there is a one-in-three satisfying assignment for g.

We now show how to create S_1. By assumption, F is a non-simple basis family, so for all $r \geq 3$ f_r has a non-simple pattern. The form of the reduction depends on which non-simple pattern(s) are contained in f_r; we consider each possibility in turn.

First, consider the pattern 0100, and for the moment assume this pattern occurs at the beginning of the basis function (i.e, $f_r(0) = 0$, $f_r(1) = 1$, etc.). An instance is represented by the set of variables which have value 1. For example, $\{x_2, x_3\}$ represents the instance 01100 when n is 5. The examples

$$\langle\{\}, 0\rangle, \langle\{y_1\}, 1\rangle, \langle\{y_1, y_2\}, 0\rangle$$

establish that y_1 and y_2 are both relevant and furthermore that any feasible basis f must have $f(0) = 0$, $f(1) = 1$, and $f(2) = 0$. Now the n examples

$$\langle\{x_i, \bar{x}_i\}, 1\rangle$$

require that every feasible relevant set R must contain exactly one variable of the pair $p_i = (x_i, \bar{x}_i)$. So every feasible R contains exactly $n+2$ relevant variables, and clearly any concept defined by f_r along with any feasible R is consistent with these examples.

Now assume that the pattern 0100 appears elsewhere in f_r. We need only carefully "pad" each example to achieve the same results. If the pattern begins at position j, $1 < j < n$, then the examples

$$\langle\{x_1, \bar{x}_1, \ldots, x_j, \bar{x}_j\}, 0\rangle, \langle\{x_1, \bar{x}_1, \ldots, x_j, \bar{x}_j, y_1\}, 1\rangle,$$
$$\langle\{x_1, \bar{x}_1, \ldots, x_j, \bar{x}_j, y_1, y_2\}, 0\rangle,$$

establish that y_1 and y_2 are relevant. For each $i > j$, the example

$$\langle\{x_1, \bar{x}_1, \ldots, x_j, \bar{x}_j, x_i, \bar{x}_i\}, 1\rangle$$

establishes that exactly one variable of p_i is relevant. For each $i \leq j$, the examples

$$\langle\{x_1, \bar{x}_1, \ldots, x_{i-1}, \bar{x}_{i-1}, x_{i+1}, \bar{x}_{i+1}, \ldots x_j, \bar{x}_j, y_1\}, 0\rangle$$
$$\langle\{x_1, \bar{x}_1, \ldots, x_{i-1}, \bar{x}_{i-1}, x_{i+1}, \bar{x}_{i+1}, \ldots x_j, \bar{x}_j, y_1, y_2\}, 1\rangle$$

can be used. Once again f_r is consistent with these examples.

Next consider the pattern 0010. A technique similar to above constructs examples establishing that y_1, y_2, and at least one variable from each pair p_i are relevant. Let j be any position within f_r at which the pattern begins (so $f_r(j+2) = 1$). Note that $j < n$. Now the examples

$$\langle\{x_1, \bar{x}_1, \ldots, x_j, \bar{x}_j, y_1\}, 0\rangle,$$
$$\langle\{x_1, \bar{x}_1, \ldots, x_j, \bar{x}_j, y_1, y_2\}, 1\rangle,$$
$$\langle\{x_1, \bar{x}_1, \ldots, x_j, \bar{x}_j, x_n, \bar{x}_n\}, 0\rangle,$$
$$\langle\{x_1, \bar{x}_1, \ldots, x_j, \bar{x}_j, y_1, x_n, \bar{x}_n\}, 1\rangle$$

establish that exactly one variable in p_n is relevant. Similar examples complete the construction for 0010.

The construction for 0011 is exactly the same as that for 0010, since the 0010 construction does not rely on the value of the fourth bit. Examples for the complements of each of these patterns are constructed by replacing 0's with 1's and vice versa in the labels of the examples. The examples can obviously be constructed in time polynomial in n.

At this point we have shown how to create examples which make f_r the only feasible basis function in F and which establish that in every feasible R the two y variables and exactly one of each of the variables $(x_i, \bar{x}_i)$ are relevant. Furthermore, every such R is feasible. We now show how to create additional examples S_2 such that there is a relevant set R consistent with the entire set of examples if and only if there is a one-in-three satisfying assignment for the given 3CNF g. The proof again proceeds by considering in turn each non-simple pattern which could appear in f_r.

First, consider the pattern 0100 and let j be any position within f_r at which the pattern begins. We illustrate the reduction by example. Assume that the clause $u = v_1 + \bar{v}_2 + v_3$ occurs within g. Let z_i represent y_{i-1} for $i = 2, 3$ and represent the pair $(x_i, \bar{x}_i)$ for $i \geq 4$. Then corresponding to this clause u we create the example:

$$\langle\{z_2, z_3, \ldots, z_{j+1}, x_1, \bar{x}_2, x_3\}, 1\rangle.$$

This example can be correctly labeled if and only if exactly one of the variables x_1, $\bar{x}_2$, and x_3 is relevant. An example achieving a similar effect is created for each clause in g. It is then straightforward to verify that there is a symmetric concept with basis f_r consistent with these examples exactly when there is a one-in-three assignment satisfying g.

Now consider either the pattern 0010 or 0011. Again assume that one of the clauses of g is u. If the pattern occurs at the beginning of f_r then the examples

$$\langle\{x_1, \bar{x}_2\}, 0\rangle, \langle\{x_1, x_3\}, 0\rangle, \langle\{\bar{x}_2, x_3\}, 0\rangle$$

establish that no more than one of the variables x_1, $\bar{x}_2$, and x_3 is relevant. The example

$$\langle\{y_1, x_1, \bar{x}_2, x_3\}, 1\rangle$$

requires that at least one of the three be relevant. As above, one set of such examples for each clause in g completes the reduction.

If the pattern 0010 or 0011 begins at a point $j > 0$ in f_r then we can use "padded" versions of the examples above to establish that at most one of the three variables corresponding to each clause is relevant. To require that at least one variable is relevant in u we produce an example which establishes that not all three "negations" are relevant:

$$\langle\{z_2, z_3, \ldots, z_j, \bar{x}_1, x_2, \bar{x}_3\}, 0\rangle.$$

Again, the reductions for the complementary patterns are simple modifications of the reductions given, and all reductions can be performed in time polynomial in the size of g. ■

The Procedures Grow and Split.

This section contains descriptions of the procedures Grow and Split used in the algorithm for exact learning of multi-symmetric functions. We introduce some useful notation. Let c be the unknown multi-symmetric concept being learned. For two vectors $X, X' \in \{0,1\}^n$, we write $X \sim X'$ if $c(X) = c(X')$, and write $X \not\sim X'$ otherwise. If in the algorithm, the relation $X \sim X'$ is to be tested, then it is implicit that this is done by making two membership queries, X and X'. We will use the symbol b to stand for 0 or 1, and $\bar{b}$ to stand for $1 - b$. For $j \in \{1, 2, \ldots, n\}$, $b \in \{0,1\}$, $X \in \{0,1,*\}^n$, we say j is a b-index of X if $X_j = b$.

Procedure Grow (X, b, R)

Assumptions (top-level invocation only):
For some $j \notin R$, $X_j = b$ and $X[j] \not\sim X$; R is a union of zero or more sets S_i.

Output (from top-level invocation only):
A set R' where $R \subset R', R \neq R'$, and $R' - R$ is a union of one or more sets S_i.

1. *Let L be the set of $j \notin R$ such that $X_j = b$ and $X[j] \not\sim X$.*
2. *If $L = \phi$ then return R.*
3. *Otherwise, $R \leftarrow R \cup L$, and for each $j \in L$ do*
   $R \leftarrow \mathsf{Grow}(X[j], \bar{b}, R)$.
4. *Return R.*

Lemma 10 *Procedure* Grow *makes $O(n^2)$ membership queries and produces an output set that satisfies the stated properties.*

Proof: Since the concept is multi-symmetric, it is clear that for each S_i, if some b-index of X^{S_i} is added to L, then *all* the b-indices of X^{S_i} will be added to L (and thus to R). We claim that if at some point all the b-indices of X^{S_i} are added to R, then in the next recursive call, all the $\bar{b}$-indices (if any) of X^{S_i} will also be added to R. At some point, suppose for some i, all the b-indices of X^{S_i} are added to R. This means that for some b-index j of X^{S_i}, $X[j] \not\sim X$. In the loop of step 3, eventually procedure $\mathsf{Grow}(X[j], \bar{b}, R)$ is called recursively. It is clear that if there is some $\bar{b}$-index k of X^{S_i} then $X[jk] \sim X \not\sim X[j]$. Therefore k (and all other $\bar{b}$-indices of X^{S_i}) will be added to R in this recursive call. Thus for any i, either all or none of S_i will be included in the final R returned. Since we assumed that to begin with, $X[j] \not\sim X$ for some $j \notin R$, at least one new S_i will be contained in the final R returned. Since Grow is only recursively called when a variable is added to R and each invocation makes at most n membership queries, the number of membership queries made is $O(n^2)$. ■

Procedure Split (X, B, p, q)

Assumptions: *B is a union of one or more sets S_i; $p, q \in B$, $X_p = 1, X_q = 0$, $X[pq] \not\sim X$.*

Output: *Sets G, H such that $G \cup H = B$, $G \cap H = \phi$, and each of G, H is a union of one or more sets S_i.*

1. *If $X[p] \sim X$ then $X \leftarrow X[pq]$ and interchange the values of p and q.*
2. *Let Q_0 be the set of 0-indices j of X^B such that $X[pj] \not\sim X$.*
3. *Let $\tilde{Q}_0$ be the set of 0-indices of X^B other than those in Q_0.*
4. *Let R_1 be the set of 1-indices k of X^B such that for some $j \in Q_0$:*
   (a) $X[jk] \sim X$, and
   (b) for every $\ell \in \tilde{Q}_0$, $X[pjk\ell] \sim X$
5. *Return ($Q_0 \cup R_1$, $B - (Q_0 \cup R_1)$).*

Lemma 11 *Procedure* Split *makes $O(n^3)$ membership queries and outputs a pair of sets with the stated properties.*

Proof: From step 4 it follows that Split makes $O(n^3)$ membership queries. The only indices that the procedure considers are those in B, so we assume all indices we refer to below are in B. Also, when we say an index is a 1-index or a 0-index, it is implicit that it is a 1- or 0-index of the vector X^B. We say an index u is a *partner* of an index v if u, v belong to the same set S_i for some i. Note that if a 0-index u is a partner of a 1-index v then for *any* vector $Y \in \{0,1\}^n$, $Y[uv] \sim Y$. After the reinitialization step 1, we have that p, q are indices in B such that $X_p = 1$, $X_q = 0$, $X[p] \not\sim X$, and $X[pq] \not\sim X$. Thus p and q are *not* partners. We want to show that the procedure returns a partition of B into two non-empty subblocks, one of which, $Q_0 \cup R_1$, contains q, and the other contains p, and that each subblock is a union of one or more sets S_i.

Since $X[pq] \not\sim X$, it follows that $q \in Q_0$ in step 2. By the definition of Q_0, $X[jp] \not\sim X$ for every $j \in Q_0$ so p fails the test 4a for membership in R_1, and therefore $p \notin R_1$. Thus $Q_0 \cup R_1$ is a *nonempty, proper* subset of B. Since the concept is multi-symmetric, for any i, either either *all* the 0-indices (1-indices) of X^{S_i} belong to Q_0 (respectively, R_1), or *none* of the 0-indices of X^{S_i} belong to Q_0 (respectively, R_1). Note that a given 1-index of X^B is a partner of a 0-index of Q_0, or a partner of a 0-index in $\tilde{Q}_0$, or it may not have any 0-index partners. We claim that (a) R_1 contains any 1-index that is a partner of some 0-index in Q_0, and (b) R_1 does *not* contain any 1-index that is a partner of some 0-index in $\tilde{Q}_0$. Combined with the observation that Q_0 (R_1) contains either all or none of the 0-indices (1-indices) in an X^{S_i}, these facts imply that $Q_0 \cup R_1$ is a union of one or more sets S_i, which is exactly what we want to show.

We now argue that R_1 satisfies the above two properties. Suppose u is a 1-index that is a partner of some 0-index j in Q_0. Clearly $X[ju] \sim X$ so u passes test 4a. In addition, for every $\ell \in \tilde{Q}_0$, $X[pju\ell] \sim X[p\ell] \sim X$, by definition of $\tilde{Q}_0$, so u passes test 4b also. Thus u is included in R_1. Suppose on the other hand that u is a 1-index that is a partner of some 0-index ℓ in $\tilde{Q}_0$. Then for *any* $j \in Q_0$, $X[pju\ell] \sim X[pj] \not\sim X$, by definition of Q_0, so the test 4b fails for $k = u$. Therefore u is not included in R_1. ■