SEPARATING DISTRIBUTION-FREE AND MISTAKE-BOUND LEARNING MODELS OVER THE BOOLEAN DOMAIN

AVRIM L. BLUM*

Abstract. Two of the most commonly used models in computational learning theory are the distribution-free model in which examples are chosen from a fixed but arbitrary distribution, and the absolute mistake-bound model in which examples are presented in an arbitrary order. Over the Boolean domain $\{0,1\}^n$, it is known that if the learner is allowed unlimited computational resources then any concept class learnable in one model is also learnable in the other. In addition, any polynomial-time learning algorithm for a concept class in the mistake-bound model can be transformed into one that learns the class in the distribution-free model.

This paper shows that if one-way functions exist, then the mistake-bound model is strictly harder than the distribution-free model for polynomial-time learning. Specifically, given a one-way function, we show how to create a concept class over $\{0,1\}^n$ that is learnable in polynomial time in the distribution-free model, but not in the absolute mistake-bound model. In addition, the concept class remains hard to learn in the mistake-bound model even if the learner is allowed a polynomial number of membership queries.

The concepts considered are based upon the Goldreich, Goldwasser and Micali random function construction [9] and involve creating the following new cryptographic object: an exponentially long sequence of strings $\sigma_1, \sigma_2, \ldots, \sigma_r$ over $\{0,1\}^n$ that is hard to compute in one direction (given σ_i one cannot compute σ_j for $j < i$) but is easy to compute *and even make random-access jumps* in the other direction (given σ_i and $j > i$ one *can* compute σ_j, even if j is exponentially larger than i). Similar sequences considered previously [6, 7] did not allow random-access jumps forward without knowledge of a seed allowing one to compute backwards as well.

Key words. Machine learning theory, learning models, one-way functions

AMS subject classifications. 68Q25, 68T05, 94A60

1. Introduction.

Two of the most popular theoretical models for learning from examples are the distribution-free model, also known as the *probably-approximately-correct* (PAC) or "Valiant-style" learning model [17, 25], and the absolute mistake-bound model [21]. In both models, the goal of a learner is to approximately infer some unknown target concept from positive and negative examples of that concept. In the distribution-free model, an adversary chooses a distribution over the labeled examples from which the learner may sample. The learner, from a polynomial (in relevant parameters discussed in Section 2) number of samples, must produce a hypothesis that agrees with the target concept over most of the distribution. In the mistake-bound model, the adversary instead actually chooses the order in which examples appear. Here, the learner sees unlabeled examples, and after each must predict whether it is positive or negative before being told its true classification. The job of the learner in the mistake-bound model is to make a polynomially bounded number of mistakes.

In this paper, we will consider the "representation-independent" versions of the distribution-free and mistake-bound models, also known as the "prediction" model [12, 13], in which the learner's hypotheses need not be of the same form as the target concept. The mistake-bound model is equivalent to Angluin's equivalence query model [3, 21] when query hypotheses are similarly not restricted.

It is known that any algorithm for learning a concept class in the mistake-bound model can be converted to one that will learn the class in the distribution-free model

* School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213-3891. Email: avrim@cs.cmu.edu. This work was done while the author was at the MIT Lab for Computer Science and supported by an NSF Graduate Fellowship, NSF grant CCR-8914428 and the Siemens Corporation.

[3, 18]. If computational considerations are ignored, then over the Boolean domain $\{0,1\}^n$ the converse holds as well. Any concept class over $\{0,1\}^n$ learnable with polynomial sample size (and *not* necessarily polynomial time) in the distribution-free model can also be learned with a polynomial mistake bound (in not necessarily polynomial time) in the mistake-bound model [3, 21]. At the heart of the proof for this last direction is the use in the mistake-bound model of the "halving algorithm". This algorithm enumerates all concepts in the class, predicts according to majority vote, and when a mistake is made throws out the concepts that predicted incorrectly. The equivalence of learnability in the *non*-computational setting has led researchers to wonder (eg. [16]) whether the models remain equivalent when computation *is* limited—especially in light of recent work showing that the distribution-free model is equivalent to a seemingly much easier form of learning known as "weak-learning" [24].

In this paper we show that if we consider only algorithms that run in polynomial time, then if one-way functions exist, the distribution-free and mistake-bound models over $\{0,1\}^n$ are *not* equivalent. Specifically, we create a concept class starting from a pseudorandom bit generator [6, 7, 26], which is learnable in polynomial time in the distribution-free model but not in the mistake-bound model. The existence of pseudorandom bit generators has been proven equivalent to the existence of one-way functions [20, 15, 11]. The concept class we create remains hard to learn in the mistake-bound model even if the learning algorithm is allowed membership queries—the ability to ask whether examples of its own choosing belong to the target concept. So, for instance, the position of this concept class contrasts with that of deterministic finite automata (DFAs) which *are* learnable in a mistake-bound model with membership queries [2] but not in the distribution-free model [19].

The concept class created is based upon the Goldreich, Goldwasser, and Micali random function construction [9] and involves creating the following new cryptographic object: an exponentially long sequence of strings $\sigma_1, \sigma_2, \ldots, \sigma_r$ over $\{0,1\}^n$ that is hard to compute in one direction (given σ_i one cannot compute σ_j for $j < i$) but is easy to compute *and even make random-access jumps* in the other direction (given σ_i and $j > i$ one *can* compute σ_j, even if j is exponentially larger than i). This sequence is "stronger" than those considered previously [6, 7] in the sense that for those sequences, the only known way to compute in the forward direction (without knowing the seed that allows one to compute in the other direction as well) is to compute the strings sequentially in order.

The sequence of strings is useful for separating the learning models for roughly the following reason. Attached to each σ_i is a classification that can only be computed given σ_j for $j < i$. So, an adversary can present the strings in the reverse order $\sigma_r, \sigma_{r-1}, \ldots$ and cause the learner to make a mistake nearly half the time. However, for any distribution over the strings, if a learner collects m samples, we expect that about $\frac{m}{m+1}$ of the distribution falls on the "easy side" (the set of σ_j for $j > i$) of the string σ_i of least index seen. The "random-access forward" property of the sequence then allows the distribution-free learner to use σ_i as a predictor for any σ_j for $j > i$. Notice that without the random-access property, then even on a uniform distribution the learner might still fail because the examples seen would likely all be exponentially far away from each other.

An earlier version of this paper appears in [5].

2. Notation, definitions, and background . An *example* x is an element of $\{0,1\}^n$ and a concept is a set of examples (subset of $\{0,1\}^n$). We will also identify

concepts with their indicator functions, defining $c(x) = 1$ if $x \in c$ and $c(x) = 0$ otherwise. For a given target concept c, a *labeled example* of c is a pair $\langle x, c(x)\rangle$ where x is an example and $c(x)$ is its *classification*. An example x is a *positive example* if $c(x) = 1$; otherwise it is a negative example. A *concept class* is a collection of concepts, together with a (sometimes implicit) representation for each concept.[1] For instance, the class of DNF formulas consists of concepts given in a disjunctive-normal-form representation. If c is a concept in a given concept class C, we use $|c|$ to denote the size of the (smallest) representation of c under C. In both the distribution-free and mistake-bound models, the object of a learning algorithm for a concept class C is to approximately infer an unknown target concept $c \in C$ from examples. The models differ in how examples are chosen and how successful learning is defined.

In the distribution-free model, there is some distribution $\mathcal{D}$ over the set of labeled examples of the target concept. The learning algorithm is allowed to sample from $\mathcal{D}$ and based on this information and knowledge of the class C to which the target concept belongs, must produce a hypothesis h that approximates the target concept. In particular, if c is the target concept, we say a hypothesis h has *error* ε if on pair $\langle x, c(x)\rangle$ chosen from $\mathcal{D}$, the probability that $h(x)$ does not equal $c(x)$ is ε. We will allow the hypothesis produced by the algorithm to not necessarily belong to concept class C (this version is sometimes termed "polynomial predictability" [12, 13]). More formally, an algorithm A *learns in the distribution-free model* a concept class C over $\{0,1\}^n$ if for some polynomial P, for all target concepts $c \in C$, distributions $\mathcal{D}$, and error parameters ε and δ: in time at most $P(n, \frac{1}{\varepsilon}, \frac{1}{\delta}, |c|)$ the algorithm with probability at least $1 - \delta$ finds a polynomial-time evaluatable hypothesis h (not necessarily from C) with error at most ε. See Haussler et al. [12] for various equivalent formulations of the distribution-free model.

In the mistake-bound model, instead of a distribution, we imagine there is some adversary presenting the examples in any order it wishes. Learning is done on-line in a sequence of *stages*. In each stage the adversary presents an unlabeled example to the learner, the learner suggests a labeling, and then the learner is told whether or not that labeling was correct. The object of the learning algorithm is to make at most a polynomial number of mistakes. Algorithm A *learns* a concept class C *in the mistake-bound model* if for some polynomial P, for all target concepts $c \in C$ and all orderings of the examples, the algorithm makes at most $P(n, |c|)$ mistakes using polynomial time in each stage [21].

A *membership query* is a query in which the learner selects an unlabeled example of its own choosing and asks for (and is told) that example's classification. We may incorporate membership queries into the mistake-bound model by allowing the learner at each stage to choose either to make a membership query or else to receive an example from the adversary. We will say that Algorithm A *learns* a concept class C *in the mistake-bound model with membership queries* if for some polynomial P, for all target concepts $c \in C$, the algorithm makes at most $P(n, |c|)$ mistakes and membership queries using polynomial time in each stage.

Note that in all models, we require a learning algorithm to run in polynomial time. In some of the literature, this is called "polynomial learnability."

We now review some useful notation and definitions from the cryptographic literature. If S is a set we will use the notation $s \in_R S$ to mean that s is chosen

[1] To be rigorous, we should define a concept class C to be a family $\{C_n\}$ (one for each n) where C_n is defined over $\{0,1\}^n$. In order to avoid overly cumbersome notation, we shall assume this type of indexing is done implicitly.

uniformly at random from S. For convenience, if A is a probabilistic polynomial time (PPT) algorithm and g is some function, we will use $\mathcal{P}_d(A, g(s))$ to mean $\mathbf{Pr}\left[A(g(s)) = 1 \mid s \in_R \{0,1\}^d\right]$. If $g(s) = s$ we will just write $\mathcal{P}_d(A, s)$. In order to simplify the statements of some of the theorems, in this paper we will use the term "algorithm" to include circuit families (non-uniform algorithms).

Informally, a Cryptographically Strong pseudorandom Bit generator, or CSB generator to use the notation of GGM, is a deterministic polynomial-time algorithm that given a randomly chosen input produces a longer pseudorandom output. More formally we have the following.

DEFINITION 1. *A deterministic polynomial-time program G is a* CSB generator with stretch t *if on input $s \in \{0,1\}^k$ it produces a tk-bit output, and for all probabilistic polynomial-time algorithms A, for all polynomials Q, for all sufficiently large k:*

$$|\mathcal{P}_k(A, G(s)) - \mathcal{P}_{tk}(A, s)| < \frac{1}{Q(k)}.$$

That is, no polynomial-time algorithm can distinguish with a $1/poly(k)$ probability between a string chosen randomly from $\{0,1\}^{tk}$ and the output of G on a string chosen randomly from $\{0,1\}^k$. For this paper, we will just need a CSB generator G with stretch 2.

For a CSB generator G that on input $s \in \{0,1\}^k$ produces a $2k$-bit output $G(s) = b_1, \ldots, b_{2k}$, let us define the following notation.

(i) Let $G_0(s)$ be the leftmost k bits $b_1, \ldots, b_k$ of $G(s)$, and let $G_1(s)$ be the rightmost k bits $b_{k+1}, \ldots, b_{2k}$.

(ii) For a d-bit string $i = i_1 \ldots i_d \in \{0,1\}^d$, let

$$G_{i_1 \cdots i_d}(s) = G_{i_d}(G_{i_{d-1}}(\ldots G_{i_2}(G_{i_1}(s)) \ldots)).$$

Note that this is well-defined because $G_0(s)$ and $G_1(s)$ are both k-bit strings.

(iii) Let $G'_0(s) = G_1(s)$ and let $G'_1(s) = \lambda$ (i.e., the empty string). Think of G' as a rightward-shift of G.

If x and y are strings, we will use $x \circ y$ to denote the concatenation xy. Also, if an example created by a concatenation of strings has length less than n, then we will implicitly pad with 0's, say on the right. Finally, let LSB$[x]$ denote the rightmost bit of string x.

We now review the GGM random function construction. Let G be a CSB generator that on input $s \in \{0,1\}^k$ produces a $2k$-bit output. Goldreich, Goldwasser, and Micali define the function $f_s : \{0,1\}^k \to \{0,1\}^k$ on input $i = i_1 \ldots i_k$ to be:

$$f_s(i) = G_{i_1 \cdots i_k}(s) = G_{i_k}(G_{i_{k-1}}(\ldots G_{i_1}(s) \ldots)).$$

The function f_s can be viewed as a complete binary tree of depth k. The root is the seed s, and on input i, the output $f_s(i)$ is the ith leaf in the tree if i is viewed as a binary number between 0 and 2^k. Though this fact will not be needed for our purposes, it is proved that the collection $F_k = \{f_s\}_{s \in \{0,1\}^k}$ is a "poly-random" collection of functions over $\{0,1\}^k$. Essentially, this means that no polynomial-time algorithm can distinguish between a function chosen randomly from F_k and a function chosen randomly from the class of all functions from $\{0,1\}^k$ to $\{0,1\}^k$.

We will use the basic form of the GGM construction as a starting point to create a concept class that separates the distribution-free and mistake-bound learning models.

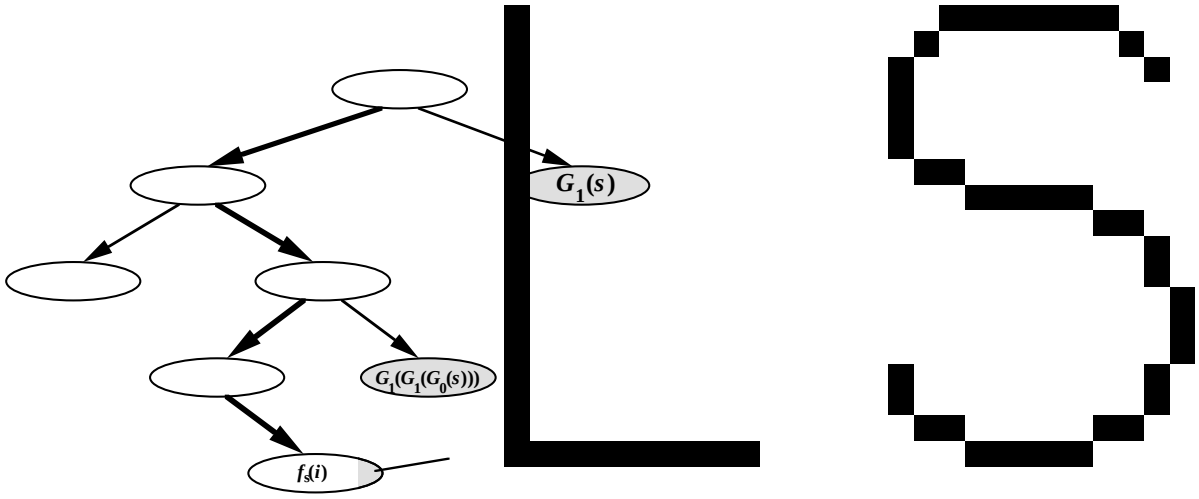

FIG. 1. *Darkened regions are information given with* $\langle x_s^i, c_s(x_s^i)\rangle$ *for* $i = 0101$.

3. A concept class to separate the learning models. We now create a concept class over $\{0,1\}^n$ that is learnable in the distribution-free model but not in the mistake-bound model if one-way functions exist. We first give the formal definition of the class, and follow this by a more informal explanation and example. Given a CSB generator G, we will fix k, the size of seed s, to equal $\lfloor\sqrt{n}\rfloor$.

DEFINITION 2. *Let* $\mathcal{C}_G = \{c_s\}_{s\in\{0,1\}^k}$ *where the concepts* c_s *are defined as follows:*

$$c_s = \{x_s^i : i \in \{0,1\}^k \text{ and } \mathrm{LSB}[G_{i_1\cdots i_k}(s)] = 1\},$$

where for each $i = i_1 \cdots i_k$,

$$x_s^i = i \circ G'_{i_1}(s) \circ G'_{i_2}(G_{i_1}(s)) \circ G'_{i_3}(G_{i_1 i_2}(s)) \circ \ldots \circ G'_{i_k}(G_{i_1\cdots i_{k-1}}(s)).$$

Now the informal description. There is one concept c_s for each $s \in \{0,1\}^k$. For each i such that $\mathrm{LSB}[f_s(i)] = 1$, there is an associated positive example x_s^i for c_s, so there are about $2^{\lfloor\sqrt{n}\rfloor-1}$ positive examples for c_s. We can think of the labeled example $\langle x_s^i, c_s(x_s^i)\rangle$ as the pair $(i, \mathrm{LSB}[f_s(i)])$ together with the following extra information: for each bit i_j equal to zero (running left to right along i), we include the right half of $G(G_{i_1\cdots i_{j-1}}(s))$. For example, if $k = 4$ and $i = 0101$, then we would have (see Figure 1):

$$\begin{aligned} x_s^i &= 0101 \circ G_1(s) \circ \lambda \circ G_1(G_1(G_0(s))) \circ \lambda \\ &= 0101 \circ G_1(s) \circ G_{011}(s), \\ c_s(x_s^i) &= \mathrm{LSB}[G_{0101}(s)]. \end{aligned}$$

We will think of the index i as both a bit string and a binary number between 0 and 2^k.

Notice that negative examples of c_s come in two forms: some are examples x_s^i for which $\mathrm{LSB}[f_s(i)] = 0$ and some are just bit strings from $\{0,1\}^n$ that are not of

the form x_s^i for any i. We will call the examples x_s^i the "good examples" of c_s; so the positive examples of c_s are the good examples x_s^i such that $\mathrm{LSB}[f_s(i)] = 1$. The examples x_s^i will be shown to have the property of the strings σ_i mentioned in the introduction.

For convenience, let us define the following additional notation.

(i) Let z_s^i be the (correctly) labeled good example $\langle x_s^i, c_s(x_s^i)\rangle$.

(ii) Let $\overline{z}_s^i$ be the incorrectly labeled good example $\langle x_s^i, 1 - c_s(x_s^i)\rangle$.

(iii) For $i_1 \ldots i_d \in \{0,1\}^d$, let:

$$G^{i_1\cdots i_d}(s) \;=\; G'_{i_1}(s) \circ G'_{i_2}(G_{i_1}(s)) \circ \ldots \circ G'_{i_d}(G_{i_1\cdots i_{d-1}}(s)).$$

So, example x_s^i is equal to $i \circ G^{i_1\cdots i_k}(s)$.

We now demonstrate an algorithm that on input x_s^i and $j > i$ computes z_s^j in time $O(nT_k)$ where T_k is the time to compute G on a k-bit input. The essential idea of the algorithm is that contained in example x_s^i are ancestors in the full binary tree of each piece of information contained in z_s^j. The reason we want to compute the entire z_s^j and not just the classification $c_s(x_s^j)$ is that the learning algorithm, given an example x that purports to be some x_s^j (because its first k bits are j), first will compute x_s^j to verify that x really is the "good example," and then predict positive exactly when both $x = x_s^j$ and the classification of x_s^j is 1.

Algorithm Compute-Forward

Given: x_s^i and $j > i$.

1. Write $i = i_1 \cdots i_k$ and $j = j_1 \cdots j_k$. Let r be the least index such that $i_r \neq j_r$. Since $j > i$, we have $i_r = 0$ and $j_r = 1$.
2. Extract from x_s^i the portions:

$$u = G'_{i_1}(s) \circ G'_{i_2}(G_{i_1}(s)) \circ \ldots \circ G'_{i_{r-1}}(G_{i_1\cdots i_{r-2}}(s)) \;=\; G^{i_1\cdots i_{r-1}}(s).$$
$$v = G'_{i_r}(G_{i_1\cdots i_{r-1}}(s)) \;=\; G_{j_1\cdots j_r}(s).$$

   (Notice that $G'_{j_r}(G_{j_1\cdots j_{r-1}}(s)) = \lambda$.)
3. If $r = k$, output: $\langle j \circ u \circ \lambda, \mathrm{LSB}[v]\rangle$.
   Otherwise, output: $\langle j \circ u \circ \lambda \circ G^{j_{r+1}\cdots j_k}(v), \mathrm{LSB}[G_{j_{r+1}\cdots j_k}(v)]\rangle$.

Algorithm Compute-Forward produces z_s^j as output for the following reason. By the definition of r, we know $G^{i_1\cdots i_{r-1}}(s) = G^{j_1\cdots j_{r-1}}(s)$, so $u \circ \lambda = G^{j_1\cdots j_{r-1}}(s) \circ G'_{j_r}(G_{j_1\cdots j_{r-1}}(s)) = G^{j_1\cdots j_r}(s)$. Since $v = G_{j_1\cdots j_r}(s)$, we have:

$$u \circ \lambda \circ G^{j_{r+1}\cdots j_k}(v) \;=\; G^{j_1\cdots j_r}(s) \circ G^{j_{r+1}\cdots j_k}(G_{j_1\cdots j_r}(s)) \;=\; G^{j_1\cdots j_k}(s),$$

and $\mathrm{LSB}[G_{j_{r+1}\cdots j_k}(v)] = \mathrm{LSB}[G_{j_1\cdots j_k}(s)] = c_s(x_s^j)$. In addition, algorithm Compute-Forward uses at worst $O(k^2)$ computations of G.

THEOREM 3.1. *Concept class $\mathcal{C}_G$ is learnable in the distribution-free model.*

Proof. Algorithm **Learn-$\mathcal{C}_G$** works as follows. Collect $m \geq \frac{1}{\varepsilon}\ln\frac{1}{\delta}$ labeled examples. If all examples are negative, hypothesize the empty concept. Otherwise, we know each positive example is a "good example", so let i be the index (first k bits) of the positive example x_s^i of least index seen so far. Produce the hypothesis:

> "On example x, let j be the first k bits of x. If $j < i$ predict 0. If $j > i$, use algorithm Compute-Forward to produce labeled example $\langle x_s^j, c_s(x_s^j)\rangle$. If $x = x_s^j$ then predict $c_s(x_s^j)$; otherwise predict 0."

One simple way to see that Learn-$\mathcal{C}_G$ learns in the distribution-free model is just to notice that it is an "Occam Algorithm" in the sense of Blumer et al. [8] since for any size sample, the hypothesis produced has size $O(n)$ and is consistent with all the data seen. Alternatively, one can use a direct argument of the type used for learning a subinterval $[0, a]$ of the interval $[0, 1]$. Let l be the least index such that the set of examples $S_l = \{z_s^j : j \leq l \text{ and } c_s(x_s^j) = 1\}$ has probability at least ε. Algorithm Learn-$\mathcal{C}_G$ fails to learn with error less than ε exactly when it sees no example from S_l. So, the probability Learn-$\mathcal{C}_G$ fails is at most $(1-\varepsilon)^m$ which is less than δ for $m \geq \frac{1}{\varepsilon} \ln \frac{1}{\delta}$. □

THEOREM 3.2. *Concept class $\mathcal{C}_G$ cannot be learned in the mistake-bound model if G is a CSB generator.*

In order to prove Theorem 3.2, we show that the sequence of examples x_s^i is difficult to compute in the reverse direction, as described in the following lemma and corollary.

LEMMA 3.3. *For any PPT algorithm A and polynomial Q, for sufficiently large k, for all $i \in \{0,1\}^k$,*

$$|\mathcal{P}_k(A, z_s^i) - \mathcal{P}_k(A, \overline{z}_s^i)| < \frac{1}{Q(k)}.$$

That is, for any algorithm A and index i, for random seeds s, algorithm A cannot distinguish between a correctly labeled x_s^i and an incorrectly labeled one. Using algorithm Compute-Forward, on input x_s^i one can easily compute any other labeled example z_s^j for $j > i$. So, Lemma 3.3 could equivalently be written as:

COROLLARY 3.4. *For any PPT algorithm A and polynomial Q, for sufficiently large k, for all $i \in \{0,1\}^k$,*

$$|\mathcal{P}_k(A^{\mathcal{O}}, z_s^i) - \mathcal{P}_k(A^{\mathcal{O}}, \overline{z}_s^i)| < \frac{1}{Q(k)},$$

where $\mathcal{O}$ is an oracle that on any input $j > i$ outputs labeled example z_s^j.

Lemma 3.3 implies, as shown below, that any learning algorithm will be fooled in the mistake-bound model by an adversary that presents the examples x_s^i in reverse order. In fact, Lemma 3.3 is stronger than we need because it implies that any algorithm will fail not just on *some* $c_s \in \mathcal{C}_G$ but in fact on almost all of the concepts c_s.

Proof of Theorem 3.2 (assuming Lemma 3.3). Suppose there exists a learning algorithm L and polynomial P such that for all $c_s \in \mathcal{C}_G$ over $\{0,1\}^n$, algorithm L makes at most $P(n)$ mistakes. Consider an adversary that presents the good examples in reverse order: $x_s^{2^k}, x_s^{2^k-1}, \ldots$. Algorithm L makes a mistake on at most one fourth of the first $4P(n)$ examples presented. If L is deterministic, this implies there exists some index t ($2^k - 4P(n) \leq t \leq 2^k$) such that over all $c_s \in \mathcal{C}_G$, algorithm L makes an error an at most one fourth of the examples x_s^t. If L is randomized, we still have that for some t, the probability over the choice of s and the coin tosses of L that L makes a mistake on x_s^t is at most $1/4$. So, we can use L in an algorithm A that contradicts Corollary 3.4: on input $\langle x_s^t, b\rangle$ we use the oracle (or algorithm Compute-Forward) to simulate the adversary presenting examples to L and then output 1 if L's prediction of $c_s(x_s^t)$ equals b. If b was the correct classification, there is at least a $3/4$ probability that L will output 1, but there is only a $1/4$ probability if b was incorrect. □

We now prove Lemma 3.3, showing that given only example x_s^i it is difficult to calculate the classification $c_s(x_s^i)$. The basic idea of the proof is a version of a

standard cryptographic technique described by Yao [26]. Roughly, if an algorithm can distinguish strings z_s^i from strings $\overline{z}_s^i$, then we will slowly substitute bits in those strings with random bits and watch the performance of the algorithm degrade. At some point before the strings have become completely random the performance must degrade significantly, and we will focus on that location to break the generator.

Proof of Lemma 3.3. Suppose to the contrary there is a PPT algorithm A and polynomial Q such that for infinitely many k, for some $t(k) \in \{0,1\}^k$, we have $|\mathcal{P}_k(A, z_s^{t(k)}) - \mathcal{P}_k(A, \overline{z}_s^{t(k)})| \geq \frac{1}{Q(k)}$. Without loss of generality, let us assume that A has $\{0,1\}$ output. We will use A to create an algorithm B that breaks generator G. In particular, for infinitely many k, we will have $|\mathcal{P}_k(B, G(s)) - \mathcal{P}_{2k}(B, s)| \geq \frac{1}{2kQ(k)}$.[2]

Let $S'_0 = \{0,1\}^k$ and $S'_1 = \{\lambda\}$ (to correspond to the notation for G'_0 and G'_1) and let $t = t(k) = t_1 \ldots t_k$. Let $p_{1,k} = \mathcal{P}_k(A, z_s^t)$ and $\overline{p}_{1,k} = \mathcal{P}_k(A, \overline{z}_s^t)$, and for $1 < d \leq k$, define:

$$p_{d,k} = \mathbf{Pr}[A\langle t \circ r_1 \circ \ldots \circ r_{d-1} \circ G^{t_d \cdots t_k}(s), b\rangle = 1 \mid r_1 \in_R S'_{t_1}, \ldots, r_{d-1} \in_R S'_{t_{d-1}}, s \in_R \{0,1\}^k]$$

$$\overline{p}_{d,k} = \mathbf{Pr}[A\langle t \circ r_1 \circ \ldots \circ r_{d-1} \circ G^{t_d \cdots t_k}(s), \overline{b}\rangle = 1 \mid r_1 \in_R S'_{t_1}, \ldots, r_{d-1} \in_R S'_{t_{d-1}}, s \in_R \{0,1\}^k]$$

where $b = \mathrm{LSB}[G_{t_d \cdots t_k}(s)]$ and $\overline{b} = 1 - \mathrm{LSB}[G_{t_d \cdots t_k}(s)]$.

So, for $d > 1$, $p_{d,k}$ is the probability that A outputs 1 on input z_s^t where the first $d-1$ "pieces" of x_s^t have been replaced by random strings of the appropriate length and the application of generator G begins at depth d in the full binary tree.

We have by assumption that $|p_{1,k} - \overline{p}_{1,k}| \geq 1/Q(k)$ for infinitely many k. We now consider two cases. The first is that $|p_{k,k} - \overline{p}_{k,k}| \geq 1/(kQ(k))$ for infinitely many k. The second is that for infinitely many k we have both $|p_{1,k} - \overline{p}_{1,k}| \geq 1/Q(k)$ and $|p_{k,k} - \overline{p}_{k,k}| < 1/(kQ(k))$. In the first case, the following algorithm B will break generator G (the analysis follows the description of the algorithm):

Algorithm B

1. On input $y \in \{0,1\}^{2k}$, let y_0 be the left half of y and y_1 be the right half of y. Let $y'_0 = y_1$ and $y'_1 = \lambda$.
2. For $i \in \{1, \ldots, k-1\}$ choose $r_i \in_R S'_{t_i}$.
3. Flip a coin:
   (i) If heads, output: $A\langle t \circ r_1 \circ \ldots \circ r_{k-1} \circ y'_{t_k}, \mathrm{LSB}[y_{t_k}]\rangle$.
   (ii) If tails, output: $1 - A\langle t \circ r_1 \circ \ldots \circ r_{k-1} \circ y'_{t_k}, 1 - \mathrm{LSB}[y_{t_k}]\rangle$.

If the input y to B equals $G(s)$ for random s, then the probability B outputs 1 is: $\frac{1}{2}p_{k,k} + \frac{1}{2}(1 - \overline{p}_{k,k}) = \frac{1}{2} + \frac{1}{2}(p_{k,k} - \overline{p}_{k,k})$. If the input y to B is randomly chosen from $\{0,1\}^{2k}$, however, then we claim the probability B outputs 1 is just $\frac{1}{2}$. The reason is that since y_{t_k} and y'_{t_k} are disjoint pieces of y, B outputs with equal probability either

[2] Algorithm B will be non-uniform, in part because we have no guarantee that the index $t(k)$ is easy to compute given k (to contradict Lemma 3.3 just requires there exist *some* $t(k)$). One could modify (and make less "clean") the statement of Lemma 3.3 so that this problem no longer occurs and still separate the learning models, since our adversary chooses the examples in an easy to compute ordering.

$A\langle t \circ r, b\rangle$ or $1 - A\langle t \circ r, b\rangle$ for independent, random r and b. So, for infinitely many k we have $|\mathcal{P}_k(B, G(s)) - \mathcal{P}_{2k}(B, s)| \geq |\frac{1}{2}(p_{k,k} - \overline{p}_{k,k})| \geq \frac{1}{2kQ(k)}$, breaking G.

The second case, recall, is that for infinitely many k we have $|p_{1,k} - \overline{p}_{1,k}| \geq \frac{1}{Q(k)}$ and $|p_{k,k} - \overline{p}_{k,k}| < \frac{1}{kQ(k)}$. If k is such that these two inequalities hold, then either $|p_{d,k} - p_{d+1,k}| \geq \frac{1}{(2k-1)Q(k)}$ or $|\overline{p}_{d,k} - \overline{p}_{d+1,k}| \geq \frac{1}{(2k-1)Q(k)}$ for some $d \in \{1, 2, \ldots, k-1\}$; this just follows from the fact that $|p_{1,k} - \overline{p}_{1,k}| \leq |p_{1,k} - p_{2,k}| + \ldots + |p_{k-1,k} - p_{k,k}| + |p_{k,k} - \overline{p}_{k,k}| + |\overline{p}_{k,k} - \overline{p}_{k-1,k}| + \ldots + |\overline{p}_{2,k} - \overline{p}_{1,k}|$. So, this second case implies that either:

(i) for infinitely many k there exists $d = d(k) \in \{1, \ldots, k-1\}$ such that $|p_{d,k} - p_{d+1,k}| > \frac{1}{2kQ(k)}$, or

(ii) for infinitely many k there exists $d = d(k) \in \{1, \ldots, k-1\}$ such that $|\overline{p}_{d,k} - \overline{p}_{d+1,k}| > \frac{1}{2kQ(k)}$.

Let us assume that the first situation occurs; the second situation is completely analogous. The algorithm B' to break generator G is then as follows.

Algorithm B′

1. On input $y \in \{0,1\}^{2k}$, let y_0 be the left half of y and y_1 be the right half of y. Let $y'_0 = y_1$ and $y'_1 = \lambda$. Let $d = d(k)$ as above.
2. For $i \in \{1, \ldots, d-1\}$ choose $r_i \in_R S'_{t_i}$.
3. Output:
$$A\langle t \circ r_1 \circ r_2 \circ \ldots \circ r_{d-1} \circ y'_{t_d} \circ G^{t_{d+1}\cdots t_k}(y_{t_d}), \mathrm{LSB}[G_{t_{d+1}\cdots t_k}(y_{t_d})]\rangle.$$

If the input y to B' equals $G(s)$ for random s, then $y'_{t_d} \circ G^{t_{d+1}\cdots t_k}(y_{t_d}) = G^{t_d \cdots t_k}(s)$ and $G_{t_{d+1}\cdots t_k}(y_{t_d}) = G_{t_d \cdots t_k}(s)$. So, the probability B' outputs 1 is $p_{d,k}$. On the other hand, if $y \in_R \{0,1\}^{2k}$, then y_{t_d} and y'_{t_d} are completely independently chosen strings, so the probability B' outputs 1 is $p_{d+1,k}$. By the choice of d, algorithm B' breaks the generator G on infinitely many k, a contradiction. □

So by Theorems 3.1 and 3.2, concept class $\mathcal{C}_G$ is learnable in the distribution-free model but not in the absolute mistake-bound model if G is a CSB generator, and such G exist if one-way functions exist.

4. Allowing membership queries. It turns out that a mistake-bound model learning algorithm can do no better on this concept class even if it is allowed to make membership queries. The reason is that in order for membership queries to help, the algorithm at some point must produce an entire good example x_s^i it has not yet seen to query. (Otherwise the membership query oracle could be replaced by a machine that always answers "negative" to any query of an example not yet seen.) As we show below, this would allow one to break G.

THEOREM 4.1. *Concept class $\mathcal{C}_G$ cannot be learned in the mistake-bound model with membership queries if G is a CSB generator.*

Proof. Suppose for contradiction there is a learning algorithm L and polynomial P such that for all $c_s \in \mathcal{C}_G$ over $\{0,1\}^n$, algorithm L makes at most $P(n)$ mistakes plus queries. The adversary will present the good examples in the order: $x_s^{2^k}, x_s^{2^k-1}, \ldots$, and let us consider the behavior of L on the first $5P(n)$ examples. Clearly, L makes a mistake on at most $1/5$ of the examples and makes at most $P(n)$ membership queries.

Let us say that a membership query is a "good query" if it is a query of a good example L has not yet seen. Also, let us define the algorithm *dequery*(L) to be the algorithm for the mistake-bound model (no queries) that runs L, and each time L

attempts to make a membership query on an example it has not yet seen, returns to L the answer "no."

Suppose first that the probability that L makes no good queries during the first $5P(n)$ examples is at least $4/5$ (probability taken over the random choice of $c_s \in \mathcal{C}_G$ and the coin tosses of L). This would imply that for some index t, over random s, the algorithm *dequery*(L) has a probability at least $4/5 - 1/5 = 3/5$ of producing the correct classification of example x_s^t. This contradicts Corollary 3.4.

We may now assume that L makes a good query at some point in the first $5P(n)$ examples with probability at least $1/5$ (over random s). Thus, for some $t \geq 2^k - 5P(n)$, the probability algorithm L makes its *first* good query after seeing x_s^t but before seeing x_s^{t-1} is at least $\frac{1}{25P(n)}$. Let $t = t_1 \ldots t_k$ and let $Q(n) = 25[P(n)]^2$.

Note that $G_{t_1 \cdots t_k}(s)$ is efficiently computable from any good example x_s^i for $i < t$. So, we can use L in an algorithm A that on input x_s^t has a probability at least $\frac{1}{Q(n)}$ of producing $G_{t_1 \cdots t_k}(s)$ as output, as follows. Algorithm A uses x_s^t and Compute-Forward to run L on inputs $x_s^{2^k}, x_s^{2^k-1}, \ldots, x_s^t$, returning "no" as an answer to each membership query made. It then guesses which membership query made after x_s^t is the good query (in case L makes several queries after x_s^t; the number of such queries is at most $\frac{1}{P(n)}$). A then uses that query to produce a hypothesis for $G_{t_1 \cdots t_k}(s)$.

In analogy to definitions in the proof of Lemma 3.3, let $q_{d,k}$ be the probability that A outputs $G_{t_d \cdots t_k}(s)$ on input $t \circ r_1 \circ \ldots \circ r_{d-1} \circ G^{t_d \cdots t_k}(s)$, where r_i is a randomly chosen string of length k if $t_i = 0$ and $r_i = \lambda$ if $t_i = 1$. We are given that $q_{1,k} \geq \frac{1}{Q(k)}$. However, we know that $q_{k,k}$ must be small for the following reason. Quantity $q_{k,k}$ is the probability that A outputs $G_{t_k}(s)$ on input $t \circ r \circ G'_{t_k}(s)$, where r is a random string of the appropriate length. If A does so with probability greater than $\frac{1}{kQ(k)}$, then the algorithm B to break G is as follows: on input y (let y_0 be the left half and y_1 be the right half), pick a random r and feed to A input $t \circ r \circ y_1$ if $t_k = 0$ or $t \circ r \circ \lambda$ if $t_k = 1$, and output 1 if A outputs y_{t_k}. If $y = G(s)$ for random s, then B outputs 1 with probability at least $\frac{1}{kQ(k)}$, but if y is a random string then B outputs 1 with probability at most $(1/2)^k$ since y_{t_k} is a random string of length k and independent of the input to A. So, B breaks G.

Thus, there must exist some value d such that $|q_{d,k} - q_{d+1,k}| > \frac{1}{kQ(k)}$. We can now break G with a variant of algorithm B' of the proof of Lemma 3.3. On input $y \in \{0,1\}^{2k}$ the algorithm outputs 1 if $A(t \circ r_1 \circ \ldots \circ r_{d-1} \circ y'_{t_d} \circ G^{t_{d+1} \cdots t_k}(y_{t_d})) = G_{t_{d+1} \cdots t_k}(y_{t_d})$. So, if $y = G(s)$ for random s, it outputs 1 with probability $q_{d,k}$, and if $y \in_R \{0,1\}^{2k}$, then it outputs 1 with probability $q_{d+1,k}$, and thus breaks G. □

5. Conclusion. We have shown how to construct a concept class over $\{0,1\}^n$ that is learnable in the distribution-free model but not in the mistake-bound model if cryptographically secure bit generators (or equivalently one-way functions) exist. In fact, the assumption of one-way functions gives us something stronger. Not only is the concept class $\mathcal{C}_G$ hard to learn in the mistake-bound model in the sense that for any learning algorithm there is *some* $c \in \mathcal{C}_G$ hard to learn, but in fact we have that for any learning algorithm, nearly all $c \in \mathcal{C}_G$ are hard to learn. The fraction of concepts which an algorithm *can* learn is less than any polynomial fraction $\frac{1}{Q(n)}$ (for sufficiently large n). This fact leads one to ask the question: is it possible to separate the two models using a weaker assumption? Some assumption is apparently necessary since if computational considerations are ignored, any concept class learnable in one model is learnable in the other.

The concept class $\mathcal{C}_G$ remains difficult to learn in the mistake-bound model even

if the learner is allowed membership queries. This lies in contrast to the class of DFAs which *are* learnable in the mistake-bound (equivalence query) model with membership queries [2] but not in the distribution-free model [19]. Thus, neither model is strictly easier than the other. Recently, membership queries have been shown not to help in learning (for a reason similar to that described here) for the important class of DNF formulas [4].

The concept class constructed here is very non-natural. To date, every "natural" concept class known to be polynomial-time learnable in the distribution-free model is also known to be polynomial-time learnable in the mistake-bound model. However, for some of these classes the known mistake-bound algorithm is qualitatively more difficult than, and sometimes followed quite a bit later than, the distribution-free algorithm. One such case is the class of decision lists (distribution-free algorithm by Rivest [23], mistake-bound algorithm by Littlestone (personal communication) and Helmbold, Sloan, and Warmuth [14]). With the addition of membership queries to both models, another case is that of read-twice DNF (distribution-free with queries algorithm by Hancock [10], and a more complicated mistake-bound with queries algorithm by Aizenstein and Pitt [1]). An interesting open problem is whether there is any natural class that would be a good candidate for being learnable in the distribution-free model but not the mistake-bound model.

I would like to thank Shafi Goldwasser, Silvio Micali, Ron Rivest, and Phil Rogaway for helpful discussions, and Rob Schapire for suggesting the simple "Occamness" argument for the proof of Theorem 3.1.

REFERENCES

[1] H. Aizenstein and L. Pitt, *Exact learning of read-twice DNF formulas*, Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, pp. 170–179, 1991.

[2] D. Angluin, *Learning regular sets from queries and counterexamples*, Information and Computation, 75 (1987), pp. 87–106.

[3] ———*Queries and concept learning*, Machine Learning, 2 (1988), pp. 319–342.

[4] D. Angluin and M. Kharitonov, *When won't membership queries help*, Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing, pp. 444–454, 1991.

[5] A. Blum, *Separating distribution-free and mistake-bound learning models over the Boolean domain*, Proceedings of the 31st Annual Symposium on Foundations of Computer Science, pp. 211–218, 1990.

[6] L. Blum, M. Blum, and M. Shub, *A simple unpredictable pseudo-random number generator*, SIAM J. Computing, 15 (1986), pp. 364–383.

[7] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Computing, 13 (1984), pp. 850–863.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, *Occam's razor*, Information Processing Letters, 24 (1987), pp. 377–380.

[9] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, Journal of the ACM, 33 (1986), pp. 792–807.

[10] T. Hancock, *Learning 2μ DNF formulas and $k\mu$ decision trees*, Proceedings of the Fourth Annual Workshop on Computational Learning Theory, pp. 199–209, 1991.

[11] J. Hastad, *Pseudo-random generators under uniform assumptions*, Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, 1990.

[12] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth, *Equivalence of models for polynomial learnability*, Proceedings of the 1988 Workshop on Computational Learning Theory, pp. 42–55, 1988.

[13] D. Haussler, N. Littlestone, and M. K. Warmuth, *Predicting $\{0,1\}$-functions on randomly drawn points*, Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science, pp. 100–109, 1988.

[14] D. Helmbold, R. Sloan, and M. K. Warmuth, *Learning nested differences of intersection-closed concept classes*, Proceedings of the Second Annual Workshop on Computational

Learning Theory, pp. 41–56, 1989.

[15] R. Impagliazzo, L. A. Levin, and M. Luby, *Pseudo-random generation from one-way functions*, Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, pp. 12–24, 1989.

[16] M. Kearns, The Computational Complexity of Machine Learning. PhD thesis, Harvard University Center for Research in Computing Technology, Technical Report TR-13-89, 1989.

[17] M. Kearns, M. Li, L. Pitt, and L. Valiant, *On the learnability of boolean formulae*, Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, pp. 285–295, 1987.

[18] ———*Recent results on boolean concept learning*, Proceedings of the Fourth International Workshop on Machine Learning, pp. 337–352, 1987.

[19] M. Kearns and L. G. Valiant, *Cryptographic limitations on learning boolean formulae and finite automata*, Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, pp. 433–444, 1989.

[20] L. A. Levin, *One-way functions and pseudorandom generators*, Proceedings of the Seventeenth ACM Symposium on Theory of Computing, pp. 363–365, 1985.

[21] N. Littlestone, *Learning when irrelevant attributes abound: A new linear-threshold algorithm*, Machine Learning, 2 (1988), pp. 285–318.

[22] W. Maass and G. Turán, *On the complexity of learning from counterexamples*, Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science, pp. 262–267, 1989.

[23] R. L. Rivest, *Learning decision lists*, Machine Learning, 2 (1987), pp. 229–246.

[24] R. E. Schapire, *The strength of weak learnability*, Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science, 1989.

[25] L. G. Valiant, *A theory of the learnable*, Communications of the ACM, 27 (1984), pp. 1134–1142.

[26] A. C. Yao, *Theory and application of trapdoor functions*, Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, pp. 80–91, 1982.