-- --

Learning Simple Concepts Under Simple Distributions

Ming Li

Computer Science Department, University of Waterloo
Waterloo, Ontario, Canada N2L 3G1.

Paul M.B. Vitanyi

Centrum voor Wiskunde en Informatica
Kruislaan 413, 1098 SJ Amsterdam
and
Universiteit van Amsterdam
Faculteit Wiskunde en Informatica

Abstract

This is a preliminary draft version. The journal version [SIAM. J. Computing, 20:5(1991), 911-935] is the correct final version. However, the polynomial time computable universal distribution section in there is too sloppy. For a better treatment see ''M. Li and P.M.B. Vitanyi, An Introduction to Kolmogorov Complexity and its Applications, Springer-Verlag, New York, Second Edition, 1997,'' Section 7.6, pp. 506-509.

We aim at developing a learning theory where 'simple' concepts are easily learnable. In Valiant's learning model, many concepts turn out to be too hard (like NP hard) to learn. Relatively few concept classes were shown to be learnable polynomially. In daily life, it seems that things we care to learn are usually learnable. To model the intuitive notion of learning more closely, we do not require that the learning algorithm learns (polynomially) under all distributions, but only under all simple distributions. A distribution is simple if it is dominated by an enumerable distribution. All distributions with computable parameters which are used in statistics are simple. Simple distributions are complete in the sense that a concept class is learnable under all simple distributions iff it is learnable under a fixed 'universal' simple distribution. This holds both for polynomial learning in the discrete case, (under a modified model), and for non-time-restricted learning in the continuous case (under the usual model). We use this completeness result to obtain new learning algorithms and several quite general new learnable classes. These include a discrete class which is known to be not polynomial learnable under Valiant's model, unless P = NP, and a continous class which is not learnable in Valiant's model. Our results allow that for each concept class from a wide range of concept classes, for each underlying distribution from a wide range of distributions, the learning algorithm uses a single fixed procedure to draw examples by a single algorithmic process using a random number generator. The 'universal' simple distribution is not computable. To make our theory feasible, we develop a polynomial time version for it. All results derived for discrete sample spaces hold mutatis mutandis for the polynomial time versions, including versions of completeness, the new learning algorithms and the new learnable classes.

The first author is supported by the NSERC operating grants OGP-0036747 and OGP-046506. He performed part of this work while at the Computer Science Department, York University. The second author was par-

1. Introduction

L.G. Valiant has proposed a learning theory, where one wants to learn a concept with high probability, in polynomial time, and a polynomial number of examples, within a certain error, under all distributions on the examples [V]. A precise definition of this 'pac-learning' is given in Section 1.2. Let us highlight its special features. It contrasts with the common approach in statistical inference, or recursion theoretical learning, where we want to learn a concept precisely in the limit, by insisting only on learning a concept approximately. The feasibility restriction to a polynomial algorithm precludes the precise learning of nontrivial concepts, and therefore we had to relax precision to within a certain error. This corresponds with natural learning, where it is important to learn fast, and it suffices to learn approximately. The additional computational requirements are orthogonal to the usual concerns in inference, and result in a distinct novel theory. But many subsequent investigations have demonstrated negative, hardness, or equivalence results [G2, A2, KLPV, PW, KV, PW1, PV]. There are at least two problems with Valiant's proposal in [V]:

(1) Under all distributions, many concept classes, including some seemingly simple ones, are not known to be polynomially learnable or known not to be polynomially learnable if $P \neq NP$, although some such concept classes are polynomially learnable under some fixed distribution.

(2) In certain situations, it may be undesirable, too slow, or impossible, to sample according to underlying distributions. We aim at developing a theory in which a learning program, for *any* concept class it can learn, under *any* distribution from a large class of distributions, draws the examples using a single fixed table, representing a particular distribution, and a random number generator. This, in contrast with having to go out in the real world and draw its examples from the actual underlying distribution.

Item (1) is at odds with the notion that machine learning should be practically useful. One may interpret it as evidence that Valiant's initially proposed requirements for learning are too strong. In practice, it seems exagarated to require that the algorithm learn under *all* distributions. Accordingly, several authors have proposed to study Valiant learning under particular distributions [KLPV,N,BI]. Then some previously (polynomially) unlearnable classes become learnable. For instance, the class of μDNF formulae is polynomially learnable under the uniform distribution. However, the assumption of *any special* distribution is obviously too restrictive and not practically interesting. There arises the problem of finding a class of distributions which is small enough to improve learning ability, but still large enough to be meaningful.

1.1. A New Approach

We propose to study Valiant-style learning under *all simple distributions*, which properly include *all* computable distributions. This allows us to systematically develop a theory of learning for *simple concepts* that intuitively should be polynomially learnable. To stress this point: maybe it is too much to ask to be able to learn all finite automata fast, but surely we ought to be able to learn a *sufficiently simple* finite automaton fast. Previous approaches looked at syntactically described classes of concepts. We introduce the idea of restricting a syntactically described class of concepts to the concepts that are simple in the sense of having low Kolmogorov complexity. This

tially supported by the NSERC International Scientific Exchange Award ISE0046203. An extended abstract of this paper appears in: *Proceedings 30th Annual IEEE Symposium on Foundations of Computer Science*, 1989, pp. 34-39.

will cover most intuitive notions of simplicity. Our other restriction, from distribution-independent learning to simple-distribution-independent learning is also not much of a restriction. *All* distributions we have a name for, like the uniform distribution, normal distribution, geometric distribution, Poisson distribution, are recursive or enumerable - if we use finite precision parameters.

In many situations sampling according to the real distribution, as prescribed in Valiant's model, is problematic. In practice the examples are more conveniently provided algorithmically, rather than by drawing them from the underlying distribution.

Benevolent teachers provide good examples first in order to train a pupil fast. To learn addition, the teacher starts with '$1+1$' rather than with '$592+4124$'. Providing the simpler examples first intuitively helps to improve the speed of learning. The results in this paper supply evidence for this thesis.

Consider a situation where a robot wants to learn but there is nobody around to provide it with examples according to the real distribution. Because it does not know the real distribution, the robot just has to generate its own examples according to its own (computable) distribution and do experiments to classify these examples (See [RS]). For example, in case of learning a finite state black box (with resetting mechanism and observable accepting/ rejecting behavior).

Putting a man on the moon, we cannot learn according to the real distribution. This is too expensive. Learning to drive without teacher in Boston is too dangerous. We learn according to easily describable emergencies.

A solution is, that in *each case* the examples are algorithmically generated by the same program, whatever concept class or underlying distribution one is dealing with. This magical solution turns out to be realizable for a quite general range of concept classes and distributions.

1.2. Basics of Learning

We review some standard definitions and facts from pac-learning theory.

(1) Let X be a set, the *sample space*. A *concept* is a subset of X. A *concept class* is a set $C \subseteq 2^X$ of concepts. An *example of a concept* $c \in C$ is a pair (x,b) where $b=1$ if $x \in c$ and $b=0$ otherwise. A *sample* is a set of examples. We enumerate a concept class C as $c_1, c_2, \ldots$ by enumerating finite binary strings $s(c_1), s(c_2), \ldots$ representing the concepts. For instance, $s(c)$ is the binary encoding of a finite automaton, and c is the language accepted by that automaton.

(2) Let $c \in C$ be the target concept and P be a distribution on X. Given *accuracy parameter* ε, and *confidence parameter* δ, a *learning algorithm* A draws a sample S of size $m_A(\varepsilon,\delta)$ according to P, and produces a *hypothesis* $h = h_A(S) \in C$.

(3) We say C is *learnable by C* if there is a learning algorithm A, such that for some ε, δ, for every distribution P and every target concept $c \in C$, as in definition (2),

$$\mathbf{P}\{P(h \Delta c) > \varepsilon\} \leq \delta,$$

where Δ denotes the symmetric difference between two sets, and $\mathbf{P}\{X\}$ is the probability of X being true. In this case we say that C is (ε,δ)–*learnable*, or *pac-learnable (probably approximately correct)*.

(4) C is *polynomially learnable* if A (ε, δ)-learns C, and runs in polynomial time (and asks for a polynomial number of examples) in $1/\delta$, $1/\varepsilon$, and the length of the representation $s(c)$ of the concept c to be learned.

-- --

(5) We also consider the case where the learning algorithm A returns $h \in C'$ satisfying definition (3), rather than $h \in C$. In this case, we say C is *learnable by C'*, or simply, C is *learnable*.

Remark. A different model as used by [V,KLPV] assumes separate distributions over positive and negative examples. These models are basically equivalent. Also see [HLW] for an online model.

We need the following very useful theorem proved by Blumer, Ehrenfeucht, Haussler, and Warmuth [BEHW]. See also [KL] for the case when the concept is only consistent with a fraction of the examples.

Occam's Razor Theorem. *Let C and C' be concept classes. Let $c \in C$ be the target concept, and let n be the length of its binary representation $s(c)$. Let A be an algorithm which (ε, δ) learns C. Let $\alpha \geq 1$ and $0 \leq \beta < 1$. Assume that A, using a sample S of m (positive and negative) examples drawn randomly from a distribution over the sample space, output a hypothesis $h \in C'$, which is consistent with at least $(1 - \varepsilon/2)m$ examples in S, and its representation $s(h)$ has binary length less or equal to $n^\alpha m^\beta$. If*

$$m = O(\max(\frac{1}{\varepsilon}\log\frac{1}{\delta}, (\frac{n^\alpha}{\varepsilon})^{\frac{1}{1-\beta}})),$$

then A learns C polynomially. If $\beta = 0$, and $n > \log\frac{1}{\delta}$, then we use $m = O(\frac{n^\alpha}{\varepsilon})$. In many cases, we have $C = C'$.

1.3. Outline of the Paper

We treat both discrete and continuous concept learning.

While each discrete concept class is learnable in unrestricted time, this is not the case for concept classes over continuous sample spaces. In the paper, for expository reasons, we first treat the discrete case, and then the continuous case. To outline the results, the reverse order seems more convenient.

In Section 4, we derive a completeness result for continuous concept learning. There exists a 'universal' measure such that a concept is learnable under this *single* measure iff it is learnable under *all* 'simple' measures. This result holds both if we sample according to the actual simple measure itself, or according to the substitute 'universal' measure. Subsequently, we use the result to show there is a concept class which is learnable under all simple measures, but which is not learnable under all measures.

In Section 2 we derive a completeness result for polynomial learning of discrete concepts. There exists a 'universal' distribution such that a concept is polynomial learnable under this *single* distribution iff it is polynomially learnable under *all* 'simple' distributions, provided we sample according to the 'universal' distribution. We use this completeness result as a novel tool to obtain new non-trivial learning algorithms for several (old and new) classes of problems, in our model. For example, the class of DNF's such that each monomial has Kolmogorov complexity $O(\log n)$, the class of Simple DNF (a superclass of the previous one), the class of simple k-reversible DFA (in the Appendix), and the class of monotone k-term DNF, are polynomially learnable under our assumptions. These classes are not known to be polynomially learnable under Valiant's more general assumptions; monotone k-term DNF are not polynomially learnable in Valiant's model, unless *P=NP*. We have put the treatment of simple k-reversible DFA in the

Appendix, because the other examples already illustrate the point we want to make well enough.

The 'universal' distribution in Section 2 is not computable. In Section 3 we develop the theory of Section 2 for polynomial time computable distributions. It turns out that this class also has a 'universal' distribution, which is exponential time computable. Apart from this, all results derived in Section 2 hold mutatis mutandis in the polynomial time setting, including the new learning algorithms and new learnable problems. We give some ideas about how to use the developed theory.

2. Discrete Sample Spaces

Notation. Let N, Q, and R denote the set of nonnegative integers, nonnegative rational numbers, and nonnegative real numbers, respectively. A superscript '+', like in N^+, restricts the set involved to the positive numbers. If x is a binary sequence, then its *length* $l(x)$ is the number of occurrences of zeros and ones in it; if x is an integer, then $l(x)$ denotes the length of the binary representation of x.

Definition. We consider countably infinite sample spaces, say $S = N \cup \{u\}$, where u is an 'undefined' element not in N. A function P from S into R, such that $\sum_{x \in S} P(x) = 1$ defines a *probability distribution* on S. (This allows us to consider defective probability distributions on the natural numbers, which sum to less than one, by concentrating the surplus probability on u.) The function P itself is properly called the 'probability density function', but we identify it loosely with the 'probability distribution'. A probability distribution P is called *enumerable*, if the set of points

$$\{(x, y) : x \in N,\ y \in Q,\ P(x) > y\},$$

is recursively enumerable. That is, $P(x)$ can be approximated from below by a Turing machine, for all $x \in N$. ($P(u)$ can be approximated from above.) Note that enumerable distributions include the recursive ones.

L.A. Levin has shown that we can effectively enumerate all enumerable probability distributions, $P_1, P_2, \ldots$. In particular, it can be proved that there exists a *universal enumerable probability distribution*, denoted by say $\mathbf{m}$, such that

$$\forall i \in N^+ \ \exists c > 0 \ \forall x \in N \ [c\, \mathbf{m}(x) \geq P_i(x)]. \tag{1}$$

That is, $\mathbf{m}$ dominates each P_i multiplicatively. Let $K(x)$ be the prefix variant of Kolmogorov complexity first proposed by L.A. Levin [L,G1]. This is defined as follows. Consider an enumeration $T_1, T_2, \ldots$ of Turing machines with a separate binary one-way input tape. Let T be such a machine. If T halts with output x, then T has scanned a finite initial segment of the input, say p, and we define $T(p) = x$. The set of such p for which T halts is a prefix code, no such input is a proper prefix of another one. Fix a universal machine in this enumeration, say U, and call it the *reference* prefix Turing machine. Define

$$K(x) = \min\{l(p) : U(p) = x\}.$$

It can be proved that, if $T(q) = x$ for some T in the enumeration and some program q, then $K(x) \leq l(q) + c_T$, where c_T is a constant depending on T but not on x. It can be proved that if $T = T_i$, then $c_T = K(i) + O(1) \leq \log i + 2 \log\log i + O(1)$. It can be proved that

$$\mathbf{m}(x) = 2^{-K(x) + O(1)}, \tag{2}$$

It can be proved that, in equation (1), the constant c can be set to

$$c = 2^{K(P_i)+O(1)} = 2^{K(i)+O(1)} = O(i \log^2 i). \tag{3}$$

This means that we can take c to be exponential in the length of the shortest self-delimiting binary program to compute P_i.

The universal distribution (rather, its continuous version) was originally discovered by R.J. Solomonoff in 1964, with the aim of predicting continuations of each finite prefix of infinite binary sequences [So]. We can view the discrete probability density $\mathbf{m}$ as the *a priori* probability of finite objects in absence of any knowledge about them. Solomonoff's approach is as follows.

Consider the enumeration $T_1, T_2, \ldots$ of prefix Turing machines again. Assume the input is provided by tosses of a fair coin. The probability that T halts with output x is $P_T(x) = \sum_{T(p)=x} 2^{-l(p)}$, where $l(p)$ denotes the length of p. Then $\sum_{x \in N} P_T(x) \leq 1$, the deficit from one being the probability that T doesn't halt. Concentrate this surplus probability on $P_T(u)$, such that $\sum_{x \in S} P_T(x) = 1$. It can be shown that P is an enumerable probability distribution iff $P = \Theta(P_T)$ for some T. In particular, $P_U(x) = \Theta(\mathbf{m}(x))$ for a universal machine U. From this, properties (1), (2), and (3) can be derived.

Levin has shown that Solomonoff's definition, and the two definitions (1) and (2) given above, are equivalent up to a multiplicative constant. Thus, three very different formalizations turn out to define the same notion of universal probability. It is customary in mathematics to view such a circumstance as evidence that we are dealing with a fundamental notion. See [ZL] for the analogous concepts in continuous sample spaces; also see [G2], and [LV1] or [LV2] for elaboration of the cited facts and proofs.

This universal distribution has many important properties. Under $\mathbf{m}$, easily describable objects have high probability, and complex or random objects have low probability. Other things being equal, it embodies Occam's Razor, which says we should prefer simple explanations over complicated ones. To give an example, with $x = 2^n$ we have $K(x) \leq \log n + 2 \log\log n + O(1)$ and $\mathbf{m}(x) = \Omega(1/n \log^2 n)$. If we generate the binary representation of y by n tosses of a fair coin, apart from the leading '1', then for the overwhelming majority of outcomes we shall have $K(y) > n$ and $\mathbf{m}(y) = O(2^{-n})$.

By Markov's inequality, for any two probability distributions P and Q, for all k, we have $Q(x) < k \cdot P(x)$ with P-probability at least $1 - 1/k$. By equations (1) and (3) therefore, for each enumerable probability distribution $P(x)$ we have

$$\sum \{P(x) : k\, \mathbf{m}(x) \geq P(x) \geq \mathbf{m}(x)/k\} \geq 1 - 1/k, \tag{4}$$

for all $k \geq K(P)$. In this sense, with high P-probability $P(x)$ is close to $\mathbf{m}(x)$, for each enumerable P. The distribution $\mathbf{m}$ is the only enumerable one which has that property. In absence of any a priori knowledge of the actual distribution therefore, apart from that it is enumerable, studying the behavior under $\mathbf{m}$ is considerably more meaningful than studying the behavior under any other particular enumerable distribution.

Definition. A distribution $P(x)$ is *simple* if it is multiplicatively dominated by some enumerable distribution $Q(x)$, as follows. There is a constant $c \leq 2^{K(Q)+O(1)}$, such that for all $x \in N$,

$$cQ(x) \geq P(x). \tag{5}$$

The first question is how large the class of simple distributions is. It certainly includes all enumerable distributions and hence all distributions with bounded precision parameters in our statistics books. We next show that containment is proper and not vacuous.

Lemma 1. *There is a non-enumerable distribution that is simple.*

Proof. Let A be a subset of the sample space. Consider the distribution

$$P(x) = \begin{cases} c/x^2 & \text{if } x \in A \\ 0 & \textit{otherwise} \end{cases}$$

The constant c is determined such that $\sum_x P(x)=1$. Now if we choose A to be a non-r.e. set, then $P(x)$ is not enumerable. But $P(x)$ is multiplicatively dominated by the recursive distribution $Q(x) = \frac{6}{(\pi x)^2}$. By trivial modification of above, we can also guarantee that $2Q(x) \geq P(x)$ for all x. □

Lemma 2. *There is a distribution which is not simple.*

Proof. We define a probability distribution $f(x)$ which exceeds $\sqrt{x}\,\mathbf{m}(x)$ for infinitely many x. Then by (1) and (5), $f(x)$ is not simple. Let u be the least monotonic upper bound on $\mathbf{m}$, $u(x) = \sup\ \{\mathbf{m}(y) : y \geq x\}$. Then $u(x) = \Omega(1/\log^2 x)$ (consider the sequence of values $x = 2^n$). The desired function $f(x)$ is defined by $f(x) = u(x)$ for infinitely many x such that $\mathbf{m}(x) \leq 1/x$, otherwise $f(x) = 0$. We have to set the x's which yield nonzero values for $f(x)$ such that $\sum f(x) \leq 1$. □

Motivation. If one can dominate the actual distribution by an enumerable distribution, then the theory we develop in this paper can be used to learn. This is the case with all distributions known in statistics, as long as the parameters are computable. What happens when the parameters are real numbers? Let us consider a Bernoulli process $(p, 1-p)$. The probability of k successes out of n outcomes is

$$B(n, k) = \binom{n}{k} p^k (1-p)^{n-k}.$$

If we truncate p to $p-\varepsilon$ in the approximation of a real number p, then the approximating probability becomes

$$B'(n, k) = (1-\frac{\varepsilon}{p})^k (1+\frac{\varepsilon}{1-p})^{n-k} B(n, k).$$

Then, for fixed ε, we find that, for $k=0$,

$$\lim_{n \to \infty} \frac{B'(n, 0)}{B(n, 0)} = \infty.$$

Suppose, however, that we have an algorithm which gives us precision of $A(n, k)$ bits of p, estimating p by $p(n, k)$ such that

$$p - p(n, k) \leq 2^{-A(n, k)}.$$

Denote the resulting approximation of $B(n, k)$, substituting p by $p(n, k)$, by $\hat{B}(n, k)$. Clearly, for each constant $0 < \delta < 1$, there is such a function A, which is computable, such that

$$\delta < \frac{\hat{B}(n, k)}{B(n, k)} < \frac{1}{\delta}.$$

In practice, the algorithm giving $A(n, k)$ bits of a real parameter p may consist in estimating p from a large number of outcomes. We cannot guaranty that such a process will always give the required precision. By the Law of Large Numbers, however, it gives the required precision with any required high probability, using sufficiently many outcomes. As it happens, this suffices for the learning application. Similar approximations can be devised for many other distributions with real parameters.

2.1. Simple Distribution-Independent Learning

In this section all concept classes we deal with are over discrete sample spaces. In the learning phase we draw the sample according to **m** instead of according to the underlying probability distribution. However, **m** is defined for all x in N, while the underlying probability distribution often assigns nonzero probability to only a (finite) subset $D \subseteq N$ (like the set of Boolean formulas over n variables). Denote a conditional probability distribution $P(x \mid x \in D)$ by $P(x \mid D)$. If the sample space is N, then we have

$$P(x \mid D) = P(x) \frac{\sum \{P(y) : y \in N\}}{\sum \{P(y) : y \in D\}}, \tag{6}$$

for $x \in D$, and $P(x \mid D) = 0$ otherwise.

We extend the notion of prefix complexity to recursively enumerable sets and enumerable functions. For each set $D \subseteq N$ define:

$$\mathbf{m}(D) = \sum \{\mathbf{m}(y) : y \in D\}$$

$$K(D) = -\log \mathbf{m}(D) + O(1).$$

If D is a recursively enumerable set, enumerated by a program p of the reference prefix machine U, then $K(D) \le K(p) + O(1)$. Namely, the ith element of D in enumeration order can be described by $0^{l(i)}\, 1\, i\, q$, where $U(q) = p$, and therefore

$$\begin{aligned} \mathbf{m}(D) &\ge \sum \{2^{-2l(i)-1-K(p)+O(1)} : 1 \le i \le |D|\} \\ &\ge 2^{-K(p)+O(1)}. \end{aligned}$$

If f is an enumerable function, enumerated by a program p of the reference prefix machine U, then $K(f) \le K(p) + O(1)$.

Definition. Let $P(\cdot \mid D)$ be a simple conditional probability distribution. We say that the learning algorithm *samples according to* **m**, if in the learning phase of concept class distributed according to $P(\cdot \mid D)$, the algorithm draws random samples from $\mathbf{m}(\cdot \mid D)$.

All properties of $\mathbf{m}(\cdot)$ versus simple $P(\cdot)$ hold by similar derivation for the conditionals $\mathbf{m}(\cdot \mid D)$ versus simple $P(\cdot \mid D)$. We can formalize the sampling notion in different ways. The following two implementations are equivalent. Assume that D is a recursive set.

(1) The learning algorithm is equipped with an **m** oracle that supplies samples according to **m**. Intuitively, this is natural in a teacher-student situation, or auto-learning by non-random experiments. In such circumstances samples with low Kolmogorov complexity are drawn with high probability. To obtain $\mathbf{m}(\cdot \mid D)$, we simply draw examples from $\mathbf{m}(\cdot)$ and discard the ones not in D. If we need to draw m examples according to $\mathbf{m}(\cdot \mid D)$, then it suffices to draw $\Omega(2^{K(D)} \cdot m)$ examples under $\mathbf{m}(\cdot)$. Namely, for each $x \in D$,

$$\begin{aligned} \mathbf{m}(x \mid D) &\le \mathbf{m}(x) \frac{\sum \{\mathbf{m}(y) : y \in N\}}{\sum \{\mathbf{m}(y) : y \in D\}} \\ &\le \frac{\mathbf{m}(x)}{\sum \{\mathbf{m}(y) : y \in D\}} \end{aligned}$$

$$=O(2^{K(D)}\cdot \mathbf{m}(x)).$$

(2) The algorithm has access to an $\mathbf{m}$ table in the form of a division of the real open interval (0, 1) into nonintersecting halfopen subintervals I_x such that $\bigcup I_x = (0, 1)$. For each x, the length of interval I_x is $\mathbf{m}(x)/\sum_y \mathbf{m}(y)$. Define the *cylinder* Γ_r as the set of all infinite binary strings starting with r. To draw a random example from $\mathbf{m}$, the algorithm uses a sequence $r_1 r_2 \cdots$ of outcomes of fair coin flips until the cylinder Γ_r, $r = r_1 r_2 \cdots r_k$, is contained in some interval I_x. It is easy to see that this procedure of selecting x, using a table $\mathbf{m}$ and fair coin flips, is equivalent to drawing a x randomly according to distribution $\mathbf{m}$. To obtain $\mathbf{m}(\cdot \mid D)$, see under Item (1).

The table $\mathbf{m}$ is noncomputable. In Section 3 we develop time limited analogues of simple distributions and a corresponding universal distribution. In many learning algorithms we consider only examples of fixed length n, which allows us to precompute a time limited version of $\mathbf{m}$. Such a table of the time limited version of $\mathbf{m}$ needs to be precomputed only *once*, and being available, can be used repeatedly by *any* learning process for learning *any* concept class.

Let us discuss the description length of enumerable distributions. By *program length* of each distribution, we mean the length of the Turing machine which computes it plus the length of the description of parameters such as the mean and variance in the normal distribution. Let us look at an example. Let the sample space be N. The uniform distribution L is defined by $L(x) = 6\cdot 2^{-l(x)}/(\pi\cdot l(x))^2$. Hence the uniform probability on n-length strings is the conditional probability $L(x \mid D) = 2^{-n}$ with $D = \{x : l(x) = n\}$. Then, $K(L(\cdot \mid D)) \le K(L) + 2\log n + O(1)$.

Our model of learning has the following *completeness property*. Without loss of generality, we assume that the sample space D is a subset of N. The following theorem says roughly that a concept class is polynomially learnable onder $\mathbf{m}$ iff it is polynomially learnable under each simple distribution. Its statement is unduly complicated because of some technicalities.

Theorem 1. *Let C be a concept class, $D \subseteq N$ the associated sample space, $m = \min\{l(s(c)) : c \in C\}$, and $d > 0$ a constant. C is polynomially learnable under the universal distribution* $\mathbf{m}$ *iff it is polynomially learnable under each simple distribution $P(\cdot \mid D)$ $(= P(\cdot))$, provided there is an enumerable distribution Q dominating P satisfying $K(Q) \le d \log m + O(1)$, and either (i) or (ii) is the case.*

(i) *In the sampling phase, the examples are drawn according to the conditional distribution* $\mathbf{m}(\cdot \mid D)$.

(ii) $K(D) \le d \log m + O(1)$, *and in the sampling phase, the polynomially many examples are drawn according to unconditional distribution* $\mathbf{m}(\cdot)$ *where the degree of the polynomial depends on d. (Typically, there is an n, such that D consists of all n-length vectors over a finite alphabet, so that $K(D) = \log n + 2 \log\log n + O(1)$, and $d = 2$ suffices.)*

Proof. Note, that in the statement of the Theorem, the polynomial associated with the learning algorithm depends on $\mathbf{m}$ and the concept class C, but not on the underlying distribution P. The 'if' case is vacuous since $\mathbf{m}$ is a simple distribution. We prove the 'only if' case.

(i) Conditional Version. We prove that if C with sample space D is polynomially learnable under $\mathbf{m}$, then it is learnable under $P(\cdot \mid D)$ while sampling according to $\mathbf{m}(\cdot \mid D)$. Let $c \in C$ be the concept to be learned, and let D be the set of all examples associated with C. Let n denote the length of the representation $s(c)$ of c. Since P is simple, there is an enumerable Q, and and a constant $d_1 < 2^{K(Q)+O(1)}$, such that, for all $x \in N$, we have $d_1 Q(x) > P(x)$. Using equation (6), there is a constant $d_2 > 0$,

$$d_2 = d_1 \frac{Q(D)}{Q(N)}$$

$$d_2 Q(x \mid D) \geq P(x \mid D).$$

Since Q is in turn dominated by $\mathbf{m}$, by (1), (3), and (5), there is a constant $d_3 = 2^{K(Q)+O(1)}$, possibly dependent on D, such that for all x, we have $d_3\,\mathbf{m}(x) \geq Q(x)$. Using (6) again, there is a constant $d_4 > 0$,

$$d_4 = d_3 \frac{\mathbf{m}(D) \cdot Q(N)}{\mathbf{m}(N) \cdot Q(D)}$$

$$d_4 \mathbf{m}(x \mid D) \geq Q(x \mid D).$$

By (6),

$$d_5 = \frac{\mathbf{m}(N)}{\mathbf{m}(D)}$$

$$\mathbf{m}(x \mid D) = d_5 \mathbf{m}(x).$$

Hence, for all $x \in D$,

$$d_2 d_4 d_5 \mathbf{m}(x) \geq P(x \mid D)$$

$$d_2 d_4 d_5 \leq 2^{2K(Q)+O(1)}.$$

Assume C is polynomially learnable under $\mathbf{m}$ using learning algorithm A. Let n be the length $l(s(c))$ of the representation $s(c)$ of the concept $c \in C$ to be learned. Then one can run algorithm A with error parameter ε / n^{2d+1} in polynomial time. Let *err* be the set of strings that are misclassified by the learned concept. So with probability at least $1 - \delta$

$$\sum_{x \in err} \mathbf{m}(x) \leq \frac{\varepsilon}{n}.$$

Then, since $m \leq n$,

$$\sum_{x \in err} P(x \mid D) \leq 2^{2K(Q)+O(1)} \sum_{x \in err} \mathbf{m}(x \mid D) \leq \varepsilon,$$

for large enough n.

(ii) Unconditional Version. We prove that if C with sample space D, $K(D) \leq d \log m + O(1)$, is polynomially learnable under $\mathbf{m}$, then it is learnable under $P(\cdot \mid D)$ while sampling according to $\mathbf{m}(\cdot)$. Let Q be as in (i). Since Q is dominated by $\mathbf{m}$, by equations (1), (3), and (5), there is a constant $d_6 > 0$, such that

$$d_6 = 2^{K(Q(\cdot \mid D))+O(1)}$$

$$d_6 \mathbf{m}(x) \geq Q(x \mid D).$$

Hence, for all $x \in N$,

$$d_2 d_6 \mathbf{m}(x) \geq P(x \mid D).$$

$$d_2 d_6 \leq d_1 2^{K(Q(\cdot \mid D))+O(1)}.$$

Since $K(Q)$ is a constant independent of n,

$$K(Q(\cdot \mid D)) \le K(Q) + K(D) \le 2d \log m + O(1).$$

Assume C is polynomially learnable under **m**, using learning algorithm A. Let n be the length $l(s(c))$ of the representation $s(c)$ of the concept $c \in C$ to be learned. Run algorithm A with error parameter ε / n^{3d+1} in polynomial time, such that, with probability at least $1 - \delta$,

$$\sum_{x \in err} P(x \mid D) \le n^{3d+1} \sum_{x \in err} \mathbf{m}(x) \le \varepsilon,$$

for n large enough. (We oversample polynomially under **m** in order to approximate $\mathbf{m}(\cdot \mid D)$.) □

In the next sections we show how to exploit this completeness theorem to obtain new learning algorithms. After all, if we know the sample space has a simple distribution, then we can learn using any learning algorithm for the specific distribution **m**. The latter distribution has the remarkably convenient property that in a polynomial sample *all* examples of logarithmic complexity occur with probability near one.

Obviously, Theorem 1 also holds if we replace **m** by any distribution Q that dominates P. But any such Q which is not equivalent to **m** and yet dominates all simple distributions, is not simple itself.

Since **m** assigns higher probabilities to simpler strings, one could suspect that after polynomially many examples, *all* simple strings are sampled and the strings that are left unsampled have only very low (inverse polynomial) probability. However, the next theorem shows that this is not the case.

Theorem 2. *Let S be a set of n^c samples drawn according to* **m** *. Then*

$$\sum_{x \notin S} m(x) = \Omega\left(\frac{1}{(\log n)^2}\right). \tag{7}$$

Proof. Consider the first n^{c+2} strings. These strings have Kolmogorov complexity at most $(c+2)\log n + 2\log\log n + O(1)$ each. The total probability for these strings, excluding S, is at least

$$\frac{1}{2}(n^{c+2} - n^c)\Omega\left(\frac{1}{n^{c+2}(\log n)^2}\right) = \Omega\left(\frac{1}{(\log n)^2}\right)$$

By using more efficient prefix coding, equation (7) can be improved to

$$\sum_{x \notin S} \mathbf{m}(x) = \Omega\left(\frac{1}{\log n \log\log n \log\log\log n \cdots}\right).$$

□

Remark. This says that we cannot just do polynomial sampling and hope to do trivial learning by listing the examples in a table. Namely, the error probability required in Theorem 1 is $\varepsilon / n^{O(1)}$, which cannot be satisfied by (7) if we set $\varepsilon = 1/n$. So nontrivial learning is required to satisfy the requirements with small ε.

Remark. Since $\mathbf{m}(x)$ is not recursively computable, one may be inclined to suggest that "sampling according to **m**" has only theoretical interest. In a separate paper, we will investigate the *polynomial time bounded approximation* of **m** and show that it has similar domination properties with respect to the polynomial time simple distributions, which still includes all textbook distributions we know off. In fact, in all of the discussion below the Kolmogorov complexity K and the universal distribution **m** can be replaced by their polynomial time bounded version. For a polynomial time bounded version of **m**, it will be possible to precompute its table once, to be used for sampling in the learning phase.

2.2. Log n - DNF

We have established that learning under the universal distribution is important, since if one can polynomially pac-learn under the universal distribution, then one can polynomially pac-learn, using the same algorithms, under any simple distribution by using the universal distribution and sampling according to it. Are there classes of concepts which are not (known to be) learnable under all distributions (in the sense of Valiant) but which are learnable while sampling according to **m**? We first consider a class for which it is not known whether it is Valiant learnable.

DNF stands for 'disjunctive normal form'. A DNF is any sum $m_1+m_2+...+m_r$ of monomials, where each monomial m_i is the product of some literals chosen from a universe $x_1, \ldots, x_n$ or their negations $\overline{x_1}, \ldots, \overline{x_n}$. A k-DNF is a DNF where each monomial consists of at most k literals. It is known that k-DNF is learnable in Valiant's sense [V]. One is inclined to think that also $(n-k)$-DNF is learnable, or the sum of monomial terms such that every 3rd variable is true, or every 7th element is true, ... like $\sum_i x_1 x_i ... x_{\lfloor n/i \rfloor i}$, where i ranges from 1 to $f(n)$ for some sublinear function f. Or more generally, expressed in a DNF form:

$$\sum_{\phi \in \Phi} x_{\phi(1)} x_{\phi(2)} \cdots x_{\phi(n)}, \tag{8}$$

with Φ a set of total recursive functions such that $|\Phi|$ is polynomial in n, $K(\phi) = O(\log n)$ and $\phi(i) \le n$, for $\phi \in \Phi$ and $i = 1,...,n$. This class should also be learnable. It is not known whether such formulae are Valiant learnable. We show they are learnable in our sense. For example, the class contains DNF formula like, for any $1 \le i \le n$,

$$f = x_1 ... x_i + x_{i+1} ... x_{2i} + x_{n-i} ... x_n .$$

Let us write $\log n$-DNF to denote DNF formulae over n variables, where each monomial term is of Kolmogorov complexity $O(\log n)$, and the length of the formula does not exceed a polynomial in n. This is a superset of k-DNF (it contains all formulae of the form (8)).

Theorem 3. *The class* $\log n$*-DNF is polynomially learnable under* **m**.

Proof. Let $f(x_1, \cdots, x_n)$ be a $\log n$-DNF where each term has Kolmogorov complexity at most $c \log n$. If m is a monomial term in f, we write $m \in f$. Sample $n^{c'}$ examples, where we choose c' large enough to satisfy the argument below.

Claim 1. With probability greater than $1 - n^c / e^n$, all examples of the following form will be drawn:

> For each monomial term m of f, the example vector that satisfies m and has zero values for all variables not in m, denoted by 0_m; the example vector that satisfies m and has one values for all variables not in m, denoted by 1_m.

Proof. Each monomial m above has Kolmogorov complexity at most $c \log n$, and therefore example 0_m has Kolmogorov complexity at most $c \log n + O(1)$. Therefore, $\mathbf{m}(0_m) \ge 2^{-c \log n + O(1)} \ge n^{-c-1}$, for large enough n. This is the probability that 0_m will be sampled. Let E be the event that 0_m does not occur in $n^{c'}$ examples. Then

$$Pr(E) \le (1 - n^{-c-1})^{n^{c'}} \le \frac{1}{2}(1/e)^n,$$

for c' large enough. The same estimate holds for the probability that the example 1_m is not sampled in $n^{c'}$ examples. There are only n^c possible monomials m such that $K(m) \le c \log n$. Hence, the probability such that all vectors 0_m and 1_m associated with such monomials m are sampled is at least $1 - n^c / e^n$. □

Now we approximate f by the following learning procedure.

Learning Algorithm

(0) Sample $n^{c'}$ examples according to **m**. Let *Pos* (*Neg*) be the set of positive (negative) examples sampled.

(1) For each pair of examples in *Pos*, construct a monomial which contains x_i if both vectors have '1' in position i, contains $\bar{x}_i$ if both vectors have '0' in position i, and does not contain variable x_i otherwise.

(2) Among monomials constructed in Step (1) delete the ones that imply examples in *Neg*. The remainder forms a set S.

(3) Let $A_m = \{x : m(x) = 1\}$, that is, A_m is the set of positive examples implied by monomial m. Use a greedy set cover algorithm to find a small set, C, of monomials $m \in S$, such that $\bigcup_{m \in C} A_m$ covers all positive examples in *Pos*.

We have to prove the correctness of the algorithm.

Claim 2. With probability greater than $1 - n^c / e^n$, $\{m : m \in f\} \subseteq S$.

Proof. By Claim 1, with probability at least $1 - n^c / e^n$, all vectors 1_m and 0_m such that monomial m has Kolmogorov complexity at most $c \log n$ are drawn. From 1_m and 0_m, m is formed in step (1) of the algorithm. Thus with probability at least $1 - n^c / e^n$, all monomials of f belong to S. □

Of course, many other monomials may also be in S. Finding all of the original monomials of f precisely is NP-hard. For the purpose of learning it is sufficient to approximate f. We use the following result due to Johnson [J] and Lovasz [Lo],

Claim 3. Given sets $A_1, \cdots, A_n$, such that $\cup_{i=1}^n A_i = A = \{1, \cdots, q\}$. If there exist k sets $A_{i_1}, \cdots, A_{i_k}$ such that $A = \cup_{j=1}^k A_{i_j}$, then it is possible to find in polynomial time $l = O(k \log q)$ sets $A_{i_1}, \cdots, A_{i_l}$ such that $A = \cup_{j=1}^l A_{i_j}$. □

Let f have k monomials. These k monomials cover the positive examples in the sense that $Pos \subseteq \cup_{m \in f} A_m$. By Claim 3, we can use about $O(kn)$ monomials to approximate f and cover the examples in *Pos* in polynomial time. Then Occam's Razor theorem implies that our algorithm polynomially learns $\log n$-DNF. □

Remark. The reader may wonder if the following scheme would work to learn $\log n$-DNF: Code each monomial as binary vectors efficiently. Then since we can sample all binary vectors of Kolmogorov complexity $c \log n$, decoding these into monomials gives us all monomials of Kolmogorov complexity $c \log n$. Then we run the set cover algorithm to choose a small set of monomials and this will achieve learning. But this scheme will not work since the sampling is done among 2^n Boolean vectors. However there are 3^n monomials of n variables. It can be easily proved that there is no effective encoding scheme which selectively codes only 2^n monomials including all $c\log n$ Kolmogorov complexity monomials. (Because, if there is such an effective procedure, then this procedure can be used to show that certain strings have Kolmogorov complexity larger than, say, $c\log n$. We know this is not possible [LV1].) An interesting open question remains: is $\log n$-decision list polynomially learnable under $\mathbf{m}(x)$? A $\log n$-decision list is a decision list of Rivest [R] with each term having Kolmogorov complexity $O(\log n)$.

2.3. Simple DNF

We learn a more general class of DNF formulae in this section. This time, we allow each term to have very high Kolmogorov complexity. Let us define a DNF formula f to be *simple* if, for each term m of f, there is vector v_m that satisfies m but satisfies no other monomials of f and $K(v_m)=O(\log n)$. Simple DNF's can contain many high Kolmogorov complexity terms. An easy example is to take a random binary sequence y with $K(y) \geq n-c$. Then the number of ones in y is about $n/2$. Construct a term m containing x_i if $y_i=1$, and neither x_i or $\overline{x}_i$ otherwise. Then $K(m) \geq n-c$, but the vector 1_m of all ones satisfying m has $K(1_m)=O(\log n)$. The class of simple DNF's is pretty general, and properly includes the class $\log n$-DNF. The learning algorithm for the class of simple DNF formulae is as follows, assuming $K(v_m) \leq c\log n$ for all v_m.

Learning Algorithm.

(0) Sample n^{c+2} examples from distribution $\mathbf{m}$.

(1) For each positive example e, construct the corresponding monomial, m_e, of size n, which is satisfied only by e.

(2) For each monomial m_e constructed in step (1), mark variable x_i in m_e if there is a negative example that differs with e by only one bit at location i. In m_e delete the unmarked variables. Remove those monomials that are satisfied by some negative examples.

(3) Use the set cover algorithm to choose a small set, S, of monomials that *cover* all the positive examples, *i.e.*, so that each positive example is implied by some monomial in S.

Theorem 4. *The class of simple DNF is polynomially learnable under* $\mathbf{m}$.

Proof. For each monomial m in f, there is a vector v_m that satisfies only m and no other monomials in f. Hence if in v_m we flip a bit corresponding to a variable in m, it becomes a negative example of Kolmogorov complexity $(c+1)\log n$. Therefore it will be sampled with high probability according to Claim 1 in the previous section. From this v_m and corresponding negative examples (which will all be sampled with high probability), one forms precisely m. Notice that variable x_i will be marked iff it appears in m. There will also be many other monomials so constructed not belonging to f. But since all monomials of f are in the set, we can cover all positive examples using about $l(f)n$ monomials in polynomial time. Hence using Occam's Razor theorem, we learn correctly with high probability. □

2.4. Monotone k - Term DNF

The previous subsections provided several classes that are polynomially learnable under the universal distribution, and hence in our sense under all simple distributions, and which are not known to be polynomially learnable in the general Valiant model. The purpose of this subsection is to demonstrate a class that was shown to be not polynomially learnable in Valiant's sense, unless P = NP, but which is polynomially learnable under **m** (and hence under all simple distributions in our model).

A Boolean formula is *monotone* if no variable in it is negated. A k-term DNF is a DNF consisting of at most k monomials. In [PV] it was shown that learning a monotone k-term DNF by k-term (or $2k$-term) DNF is NP-complete (See also [KLPV]).

Theorem 5. *The class of monotone k-term DNF is polynomially learnable by monotone k-term DNF, under* **m**.

Proof. Assume we are learning a monotone k-term DNF $f(x_1, \cdots, x_n) = m_1 + \cdots + m_k$, where m_i's are the k monotone monomials (terms) of f.

Learning Algorithm.

0. Draw $n^{k'}$ examples according to **m**, for $k' > k+1$. Set DNF $g := \emptyset$. (g is the DNF we will eventually output as approximation of f.)
1. Pick a positive example $a = (a_1, \cdots, a_n)$. Form a monotone term m such that m includes x_i if $a_i = 1$.
2. *for* each positive example $a = (a_1, ..., a_n)$ *do*: if $a_i = 0$ and deleting x_i from m violates no negative examples, delete x_i from m.
3. Remove from the sample all positive examples which are implied by m. Set $g \leftarrow g + m$. If there are still positive examples left, then go to step 1, else halt and return g.

We show that the algorithm is correct. Let us write $m_i \subseteq m$ for two monotone monomials if all the variables that appear in m_i also appear in m. At step 1, the monomial m obviously implies no negative examples, since for some monomial m_i of f we must have $m_i \subseteq m$. Step 2 of the algorithm keeps deleting variables from m. If at any time for no monomial $m_i \in f$ holds $m_i \subseteq m$, then there exists a negative example that contains at most k zeros such that it satisfies m but no m_i of f. This negative example is of Kolmogorov complexity at most $k \log n$, hence with high probability (at least $1/e^n$) it is contained in the sample. Hence at step 2, with high probability, there will be an m_i such that $m_i \subseteq m$. Hence we eventually find a correct m_i (precisely) with high probability. Then at step 3, we remove the positive examples implied by this m_i and continue on to find another term of f. The algorithm will eventually output $g = f$ with high probability (at least $1 - n^c / e^n$ for some constant c). □

Remark. Notice that this is not an approximation algorithm like the ones in the previous sections. This algorithm outputs the precise monotone formula with high probability.

3. Discrete Sample Spaces and Polynomial Time Computable Distributions

The drawback of the theory developed in Section 2, is that **m** is not computable. In this section we scale the entire theory down to a more feasible domain.

Consider the countably infinite discrete sample space N. It is convenient to formulate this section in terms of distribution functions $P^*: \{1, 2, \cdots\} \rightarrow [0,1]$, where $P^*(x)$ is the probability of all instances not exceeding x. Its density $P(x) = P^*(x) - P^*(x-1)$ is the probability of example x.

A function f is computable in time t, if there exists a Turing machine T which on input x computes output $f(x)$ in at most $f(l(x))$ steps. We construct a time bounded version of equation (2) as follows. First we define a time-bounded version of Kolmogorov complexity, see also [LV1]. Let U be the reference universal prefix machine U (as in Section 2). Let t be a total function. Define the t-time bounded version of $K(x)$ by:

$$K_t(x) = \min\{l(p): U(p)=x, \text{ the computation taking} \leq t(l(x)) \text{ steps}\}.$$

Note that, for all t and x, $K_t(x) \leq l(x)+O(1)$. The limiting value of $K_t(x)$, for $t(\cdot)\to\infty$, is

$$\lim_{t(l(x))\to\infty} K_t(x) = K(x).$$

According to generally accepted notions of feasibility (in the theory of computing), t should be polynomial. Let t denote a *polynomial* in the sequel, up to multiplicative constant factors.

Definition. The t-time-bounded version of $\mathbf{m}$, denoted by $\mathbf{m}_{<t>}$, is defined as follows.

$$\mathbf{m}_{<t>}(x) = 2^{-K_t(y)}$$

$$\mathbf{m}^*_{<t>}(x) = \sum_{y\leq x} \mathbf{m}_{<t>}(y)$$

Note that, for all t and x, we have $\mathbf{m}_{<t>}(x) \leq 2^{-l(x)+O(1)}$. In the limit, for $t(\cdot)\to\infty$, we have, using equation (2),

$$\lim_{t(l(x))\to\infty} \mathbf{m}_{<t>}(x) = \Theta(\mathbf{m}(x)).$$

Let $\mathbf{P}(t)$ denote the class of t-time computable probability distributions P^*. We want to show that $\mathbf{m}_{<nt(n)>}$ multiplicatively dominates all probability density functions P with $P^* \in \mathbf{P}(t)$.

Note. We do *not* know whether $\mathbf{m}^*_{<nt(n)>}$ is polynomial time computable. We can compute the density function $\mathbf{m}_{<n(t(n))>}(x)$ in $t(n)2^n$-time, by computing $K_{nt(n)}$ by running all programs of length less than $l(x)+O(1)$ for $nt(n)$ steps, and determining the length of the shortest program which halts with output x. A variation yields the same trivial upper bound on the computation time for $\mathbf{m}^*_{<nt(n)>}(x)$, by computing the sum

$$\mathbf{m}^*_{<nt(n)>}(x) = \sum_{y\leq x} \mathbf{m}_{<l(y)t(l(y))>}(y).$$

Theorem 6. *The distribution* $\mathbf{m}_{<nt(n)>}$ *is universal for the class* $\mathbf{P}(t)$ *in the sense that it multiplicatively dominates each P with* $P^* \in \mathbf{P}(t)$*: there exists a constant* $c > 0$*, such that, for all x, we have* $c\,\mathbf{m}_{<nt(n)>}(x) \geq P(x)$.

This follows immediately from the following Lemma.

Lemma 3. *If the probability distribution P^* is computable in time t, then there is a constant c, such that for all x:*

$$K_{nt(n)}(x) \leq -\log P(x) + c,$$

where $l(x)=n$.

Proof. We wish to show that $K_{nt(n)}(x) \leq -\log P(x)+c$. Without a polynomial time bound, a proof similar to that of the optimality of the Shannon-Fano code would be sufficient (like that of equation (2), see [LV2]). But we have to deal with the time bound here.

We divide the real interval $[0,1]$ into subintervals such that the code word $p(x)$ for source word x 'occupies' $[P^*(x-1), P^*(x)]$. Notice that $\sum_x P(x) \leq 1$. The *binary interval* determined by the finite binary string r is the half open interval $[0.r,\ 0.r+2^{-l(r)})$ corresponding to the set of reals (cylinder) Γ_r consisting of all reals $0.r\ldots$. If Γ_r is the greatest binary interval contained in $I_x = [P^*(x-1), P^*(x))$, then x is encoded as $p(x) := r$. Since length $l(I_x) = P(x)$, it is easy to show that $l(p(x)) \le -\log P(x) + 2$ bits.

We have to give polynomially encoding and decoding algorithms. The encoding algorithm is trivial: since P^* is computable in time $t(n)$, given source word x, $l(x) = n$, the code word $p(x)$ can be computed from $P^*(x-1)$ and $P^*(x)$ in $O(t(n))$ time. In order to compute p^{-1}, the decoding function, given a code word $p(x)$, we proceed as follows.

Decoding Algorithm.

(1) Set $k := 1$.

(2) Repeatly set $k := 2k$ until $2^{-p(x)}$ lays in or left of interval $[P^*(k-1), P^*(k)]$. Set $u := k$ and $l := k/2$.

(3) (Binary search) Let $m = (u+l)/2$. If $\Gamma_{p(x)}$ is the maximum binary interval in $[P^*(m-1), P^*(m)]$, then return $x = m$. Otherwise set $u := m$ if $2^{-p(x)}$ lays left of $P^*(m-1)$ and set $l := m$ if $2^{-p(x)}$ lays right of $P^*(m)$.

This procedure is similar to a binary search, and it takes at most time

$$\sum_{i=0}^{n} O(t(i)) = O(nt(n)), \quad n = l(x),$$

time to find x.

This completes our encoding/decoding of x using distribution P^*. To reconstruct source word x from its code word $p(x)$, with $l(p(x) \le -\log P(x) + 2$ by the above construction, the description of the Decoding Algorithm is also needed. If q is a binary program to compute P^*, then the latter description takes $l(q) + O(1)$ bits. Since $K_{nt(n)}(x)$ is the shortest program from which x can be reconstructed in $O(nt(n))$ steps, setting $c = l(q) + O(1)$, we have

$$K_{nt(n)}(x) \le -\log P(x) + c.$$

□

Corollary. Note, that $c = K_{nt(n)}(P) + O(1)$ suffices, since the program q computing P^* may be reconstructed in time $O(n(t(n)))$ from a shorter description q'.

Above we noticed that we do not know how to compute $\mathbf{m}_{<t>}$ in polynomial time. But for a subset of the domain we can do better.

Lemma 4. *The probabilities* $\mathbf{m}_{<t>}(x) = 2^{-K_t(x)}$ *for the set of all x's with* $K_t(x) = O(\log n)$, *can be computed in time polynomial in* $t(n)$,

Proof. As in Section 2, use the universal prefix Turing machine U with a one-way input tape, a two-way work tape, and an output tape. On the input tape it is provided with sequence of 0's and 1's, generated by random flips of a fair coin. The universal machine finds the first initial segment of the coin-tossing sequence which constitutes a program in a prefix-free code. Such a program must be in form of $(n, t(n), p)$. If this is not the case, U discards the code. Otherwise U proceeds as follows. Simulate p for $t(n)$ steps. If p stops within $t(n)$ steps, then print the output of p, otherwise print *undefined* on the output tape. This way the probability of generating a string x, $l(x) = n$, is at least $2^{-K_t(x) - O(\log n)}$.

If $K_t(x) = O(\log n)$, then we can try all programs p of length $l(p) \le K_t(x)$, and determine the values of $\mathbf{m}_{<t>}(x) = 2^{-K_t(x)}$ for all such x together, in time polynomial in $t(n)$. □

In time polynomial in $t(n)$ we can obviously find all x of length n with $K_t(x) = O(\log n)$, and by Lemma 4 also determine their probabilities $\mathbf{m}_{<t>}(x)$. This way we can precompute $\mathbf{m}_{<t>}$ for the high probability x's in polynomial time. However, for us the following Lemma is more important.

Lemma 5. *To compute a table* $\mathbf{m}_{<t>}(x+1), \ldots, \mathbf{m}_{<t>}(x+2^n)$ *takes time* $O(t(n)\, 2^n)$. *In other words, we can divide* $[0,1)$ *into* 2^n *half open disjoint intervals* I_y *with* $|I_y| = \mathbf{m}_{<t>}(y)/(\mathbf{m}^*_{<t>}(x+2^n) - \mathbf{m}^*_{<t>}(x))$, *such that* $\bigcup_y I_y = [0,1)$, $y = x+1, \ldots, x+2^n$.

Hence, if we want to learn a concept class using examples of fixed size n, sampling according to $\mathbf{m}_{<t>}$, we can *precompute* the interval representation of the table (as in the Corollary) once and for all, and use it to sample according to $\mathbf{m}_{<t>}$ by means of a sequence of fair coin flips analogous to what we explained in Section 2 for $\mathbf{m}$.

The entire Sections 2 and 3 can now be reformulated in terms of t-time limited simple distributions, $\mathbf{m}_{<nt(n)>}$ and $K_{nt(n)}$. For example, a distribution $P(x)$ is *t-simple*, if it is dominated by some distribution $Q \in \mathbf{P}(t)$, as follows. There is a constant $c \le 2^{K_{nt(n)}(Q)+O(1)}$, such that, for all $x \in N$, we have $cQ(x) > P(x)$. We leave it to the reader to prove the analogons of Lemma's 1 through 2, and Theorems 1 through 5.

To use it, proceed as follows. Fix a low t, like $t = O(n^2)$, and precompute once and for all a $\mathbf{m}_{<n^3>}$ table. Let C be a concept class which is polynomially learnable under $\mathbf{m}_{<n^3>}$. For example, C is the class of n^2-simple DNF (analogous to simple DNF in Section 2.3). We can use this table, together with random coin flips as explained in Section 2, to polynomially learn n^2-simple DNF under all n^2-simple distributions.

4. Continuous Sample Spaces

We consider continuous spaces, say the set of all one-way infinite sequences over some basic set of elements B. The sample space is $\Omega = B^\infty$. A *measure* μ on Ω satisfies, with Λ the empty word and $x \in B^*$:

$$\mu(\Lambda) = 1, \tag{9}$$

$$\mu(x) = \sum_{a \in B} \mu(xa). \tag{10}$$

The meaning of $\mu(x)$ is the combined probability (measure) of the set of elements, or *cylinder*, $\Gamma_x \subseteq \Omega$ defined as $\Gamma_x = \{x\omega : \omega \in \Omega\}$. Using standard notions from measure theory, we can form the closure of the set of cylinders under complementation and taking countable union (and therefore countable intersections), each resulting set having an appropriate μ-measure. The resulting sets are called Borel sets, and form a so-called σ-algebra denoted by, say, σ. The pair (σ, μ), is called a probability field. See [Ha].

Example. The discrete probability distributions we considered before, actually probability densities, correspond to measures with $B = N \cup \{u\}$ and the sample space restricted to $\{x : x \in \Omega, l(x) = 1\}$. This way, we only consider the measures of the cylinders Γ_a, where $a \in N \cup \{u\}$.

Example. Another example is the Lebesgue measure, or uniform measure, on interval $(0,1)$. Take $B = \{0, 1\}$, and consider the measure $\lambda(x) = 2^{-l(x)}$. This has a geometric interpretation. Consider real numbers in $(0, 1)$ as being represented by their binary representation. A number like $1/2$ has two representations. Then we choose the representation with infinitely many ones. The uniform measure of the cylinder Γ_x is the length of the real interval $(0.x,\ 0.x + 2^{-l(x)})$.

A measure μ over Ω is *enumerable*, if the set

$$\{(x, y) : x \in (B - \{u\})^*, \ y \in Q, \ y \leq \mu(x)\} \tag{11}$$

is recursively enumerable, where u is a special 'undefined' symbol in B.

It remains to elucidate the role of u. We would have liked to satisfy (9) and (10) with $B = \{0, 1\}$, but (9), (10) and (11) together, with $B - \{u\}$ replaced by B, imply that the measure is recursive. It can be shown that the set of recursive measures cannot be enumerated such that it contains a universal recursive measure. Using enumerable functions it turns out that we cannot satisfy (9) and (10), but only $\mu(\Lambda) \leq 1$ and $\mu(x) \geq \mu(x0) + \mu(x1)$. The surplus probability corresponds with nonterminating computations. By use of the 'undefined' symbol u, we normalize the enumerable defective measures to proper measures, by concentrating the surplus probability on dummy 'undefined' elements in the sample space.

We obtain an enumeration of enumerable measures as follows. Consider an enumeration $T_1, T_2, \ldots$ of Turing machines, with a one-way input tape, a one-way output tape and a two-way worktape. The input elements are taken from a set A, one element in each input tape square. Initially, the output tape contains the symbol u in each square. A machine T in the list computes a function from A^* into B^*, as follows. If after having scanned an initial input segment p, upon shifting its input head to the $(l(p) + 1)$th input tape square, the initial segment of the output tape up to the position of the output tape head contains x, $l(x) < \infty$, then $T(p) = x$. Such Turing machines are called *monotonic machines*. Setting $A = \{0, 1\}$ and $B = \{0, 1, u\}$, we define

$$\mu_T(x) = \sum_{T(p) = x} 2^{-l(p)}.$$

In other words, if the input to T is supplied by tosses of a fair coin, then $\mu_T(x)$ is the probability that the output of T starts with x. Using the definition of μ_T, it is easy to define a recursive function which approximates μ_T from below in the sense of (11). Hence μ_T is an enumerable measure. It can be shown [ZL], that μ is an enumerable measure iff $\mu = \mu_T$ for some monotonic machine T. Hence, we have obtained an enumeration $\mu_1, \mu_2, \ldots$ of enumerable measures.

We are particularly interested in μ_U, where U is a universal monotonic machine U in the list $T_1, T_2, \ldots$. Say, U has the property that $U(0^{l(l(n))} 1 l(n) np) = T_n(p)$ for all p in $\{0, 1\}^*$. This means that if $T = T_n$ in the enumeration, then

$$\mu_U(x) \geq \frac{\mu_T(x)}{2n \log^2 n},$$

for all x in $\{0, 1\}^*$. Fixing U and defining $\mathbf{M} = \mu_U$, we have established the result of Levin: the enumerable measure $\mathbf{M}$ multiplicatively dominates all enumerable measures μ, in the sense that

$$\forall i \in N^+ \ \exists c > 0 \ \forall x \in (B - \{u\})^* \ [c \, \mathbf{M}(x) \geq \mu_i(x)]. \tag{12}$$

We call such a $\mathbf{M}$ a *maximal* enumerable measure or a *universal* enumerable measure.

A measure μ over Ω is *simple* if it is dominated multiplicatively by an enumerable measure σ, (there is a $c > 0$ such that $\sigma(x) > c\,\mu(x)$ for all x). Obviously, each enumerable measure is simple, and each simple measure μ is multiplicatively dominated by $\mathbf{M}$ in the sense of $\mu(x) = O(\mathbf{M}(x))$.*

* Similar to the discrete case one can show that there are simple measures which are not enumerable, and there are measures which are not simple.

4.1. Learning Continuous Concepts

A concept class is a subset $C \subseteq 2^{\Omega}$ of concepts, each of which is a Borel set. If c is a concept to be learned, then $x \in \Omega$ is a positive example if $x \in c$, and it is a negative example if $x \in \Omega - c$. The remaining definitions of learning can now be rephrased in the continuous setting in the obvious way.

While for discrete sample spaces all concept classes are learnable (although not all are polynomially learnable), this is not the case for continuous sample spaces. Here we show that all continuous concept classes are learnable over each simple measure μ iff they are learnable under the universal measure **M**. In contrast with polynomial learning of discrete concepts, we do not need to require (but do allow) that the learning algorithm samples according to the universal measure.

Theorem 7. *A concept class C of concepts in Ω is learnable under* **M** *iff it is learnable under each simple measure.*

Proof. The 'if' part holds vacuously. We only need to prove the 'only if' part. We use some definitions and results from [BI]. According to [BI], C_{ε} is an ε-*cover* of C, with respect to distribution μ, if for every $c \in C$ there is a $\hat{c} \in C_{\varepsilon}$ which is ε-close to c ($\mu(c \Delta \hat{c}) < \varepsilon$). A concept class C is *finitely coverable*, if for every $\varepsilon > 0$ there is a finite ε-cover C_{ε} of C, everything with respect to a given measure μ. A finite ε-cover C_{ε} has finitely many concepts c, and each c is in the closure of the set of cylinders under finite union, complement (and finite intersection).

Lemma 6. *A concept class C is finitely coverable with respect to μ iff C is learnable with respect to μ.*

Proof. We give the main idea of the proof in [BI].

'Only If'. Assume that C is finitely coverable under μ. We show C is learnable under μ. This is done by encoding the finite cover set C' of C into the learning algorithm. We iterate the following procedure. We draw a sample, and choose the concept from C' which minimizes the error in the classification of the elements from the sample. By the standard application of Occam's Razor theorem this algorithm learns if the size of the sample sufficiently exceeds the size of the concept selected. This can always be guaranteed since there is no a priori limit on the sample size, but there is an a priori limit on the size of concepts in C' since C' is finite.

'If'. Assume that A learns C under μ using a sample of size l with error less than ε with probability greater than $1-\delta$. We show that C is finitely coverable under μ. Let $n = n(\mu, C, \varepsilon)$ be the cardinality of the smallest 2ε-cover of C under μ (n is possibly infinite).

Choose a set $C_{2\varepsilon} \subseteq C$ of n pairwise 2ε-far concepts ($\mu(c \,\Delta\, c') \geq 2\varepsilon$ for all unequal c, c' in $C_{2\varepsilon}$). For instance, define a sequence of concept classes $C_0, C_1, \ldots$ by $C_0 = \emptyset$ and $C_{i+1} = C_i \cup \{c\}$ such that c is 2ε-far from all concepts in C_i. Then $C_{2\varepsilon} = C_n$ if n is finite, or $C_{2\varepsilon} = \lim_{i \to \infty} C_i$ if $n = \infty$.

Let c be the concept to be learned. Let $\mathbf{x} = (x_1, \cdots, x_l) \in \Omega^l$ be a sequence of l examples and $\mathbf{L} = (L_1, \cdots, L_l) \in \{0,1\}^l$. Then $(\mathbf{x}, \mathbf{L})$ is a sample of size l. If $L_i = 1$ if $x_i \in c$ and $L_i = 0$ if $x_i \in \Omega - c$, then we denote this $\mathbf{L}$ by $\mathbf{L}_c$. Let $h_A(\mathbf{x}, \mathbf{L})$ be the concept returned by learning algorithm A. For $c \in C$, let

$$\chi(c, h_A(\mathbf{x}, \mathbf{L}), \varepsilon) = \begin{cases} 1 & \text{if } \mu(h_A(\mathbf{x}, \mathbf{L}) \Delta c) < \varepsilon \\ 0 & \textit{otherwise} \end{cases}$$

Consider the sum

$$S = \sum_{c \in C_{2\varepsilon}} \int_{\mathbf{x}} \chi(c, h_A(\mathbf{x}, \mathbf{L}_c), \varepsilon) d\mu. \tag{13}$$

By hypothesis, the probability that $\chi(c, h_A(\mathbf{x}, \mathbf{L_c}), \varepsilon) = 1$ exceeds $1-\delta$, for a randomly drawn sample $(\mathbf{x}, \mathbf{L}_c)$ from μ. From (13) we obtain $S > (1-\delta)n$. On the other hand we have, trivially,

$$S \le \sum_{c \in C_{2\varepsilon}} \int_{\mathbf{x}} \sum_{\mathbf{L} \in \{0,1\}^l} \chi(c,\ h_A(\mathbf{x}, \mathbf{L}), \varepsilon)\ d\mu$$

$$= \int_{\mathbf{x}} \sum_{\mathbf{L} \in \{0,1\}^l} \sum_{c \in C_{2\varepsilon}} \chi(c,\ h_A(\mathbf{x}, \mathbf{L}), \varepsilon)\ d\mu.$$

Since concepts in $C_{2\varepsilon}$ are 2ε-far, for every $(\mathbf{x}, \mathbf{L})$ there exists at most one $c \in C_{2\varepsilon}$ such that $\chi(c,\ h_A(\mathbf{x}, \mathbf{L}), \varepsilon) = 1$. Thus

$$(1-\delta)n \le S \le \int_{\mathbf{x}} \Big(\sum_{\mathbf{L} \in \{0,1\}^l} 1 \Big)\ d\mu = \int_{\mathbf{x}} 2^l\ d\mu = 2^l.$$

Hence $l \ge \log((1-\delta)n)$. When $n = \infty$, this implies that the learning algorithm A has to take an infinite sample. Hence finitely learnable means finitely coverable. □

Using Lemma 6, if C can be learned under $\mathbf{M}$, then C can be finitely covered with respect to $\mathbf{M}$. Let μ be a simple distribution, and $d > 0$ such that $\mathbf{M}(x) \ge d\,\mu(x)$ for all x. Then, any finite $\varepsilon\, d$-cover of C, with respect to $\mathbf{M}$, is also a finite ε-cover with respect to μ. Using Lemma 6 again, it follows that C is learnable with respect to μ. □

Remark. Note that this is a strong statement since we are saying that if one can learn under $\mathbf{M}$, then one can also learn under any simple measure μ, while sampling according to μ. In the polynomial learning of discrete concepts we required sampling according to $\mathbf{m}$ in the learning phase. This improvement of Theorem 7 over Theorem 1 is made possible by relaxing the 'polynomial learning' requirement to 'learning' (without a priori time bound).

Obviously, by the proof, if a concept class C is finitely coverable with respect to $\mathbf{M}$, then it is also finitely coverable and hence learnable under any simple distribution μ.

We may wonder whether Theorem 7 is vacuously true. Namely, do there exist concept classes that are learnable under all simple measures but are not learnable under all measures?

Definition. Given a concept class C and finite $S \subseteq \Omega$. If $\{S \cap c : c \in C\} = 2^S$, then we say S is *shattered* by C. The *Vapnik-Chervonenkis (VC) dimension* of C is the smallest integer d such that no $S \subseteq \Omega$ of cardinality $d+1$ is shattered by C; if no such d exists then the dimension of C is infinite.

It is a fundamental result of [BEHW], that a class C has finite VC-dimension iff it is learnable under arbitrary measures.

Theorem 8. *There is a concept class that is learnable under all simple measures but not learnable under all measures.*

Proof. Each singleton set $\{\omega\}$, $\omega \in \Omega$, is a Borel set obtainable by countable intersection:

$$\{\omega\} = \bigcap_{n \in N} (\Gamma_{\omega_{1:n}}),$$

and therefore is measurable. There are elements $\omega \in \Omega$ with $\mathbf{M}(\{\omega\}) > 0$. Clearly, there are only countably many such elements. Therefore, modified cylinders in Ω' defined by

$$\Gamma_x' = \Gamma_x - \{\omega \in \Gamma_x : \mathbf{M}(\omega) > 0\},$$

are also Borel sets. Let the concept class C be the class of concepts c, where each c is defined by by an index set $I \subseteq B^*$, such that

$$c = \bigcup \{\Gamma_x' : x \in I\},$$

satisfying: if $x, y \in I$, $x \neq y$, then either $\mathbf{M}(\Gamma_x') \geq 2\mathbf{M}(\Gamma_y')$ or vice versa. We show C has infinite VC dimension. For any d consider a set $A \subseteq \Omega'$ of cardinality d. For every subset A' of A we can find a concept $c \in C$ of d cylinders such that $c \cap A = A'$. Therefore, by the result of [BEHW] quoted above, C is not learnable under arbitrary measures.

We now show that C is finitely coverable under $\mathbf{M}$, and hence learnable under $\mathbf{M}$, by Lemma 6. For each $\varepsilon > 0$, there are only finitely many cylinders Γ_x' such that $\mathbf{M}(\Gamma_x') > \varepsilon/2$. We denote this set of cylinders by A_ε. Given ε, we define an ε-cover of C with respect to $\mathbf{M}$ as follows:

$$C_\varepsilon = \{c' : \exists c \in C\, [c' = \bigcup \{\Gamma_x' \in A_\varepsilon : \Gamma_x' \subseteq c\}]\}.$$

It can be easily verified that C_ε is finite and ε-covers C. Therefore C is finitely coverable with respect to $\mathbf{M}$, and C is learnable with respect to $\mathbf{M}$, by Lemma 6. □

5. Concluding Remarks

We have restricted Valiant distribution-independent learning to simple-distribution-independent learning. In the case of discrete sample spaces, we found a ‘hardest’ or ‘universal’ distribution **m** which holds all the secrets about polynomial learning of simple concepts in this world. In a sense we were even more successful for continuous sample spaces. There we found the straight completeness result that a concept is learnable under all simple measures iff it is learnable under the particular simple measure **M**. We demonstrated the use of these completeness results by exhibiting new learning algorithms, new learnable concept classes, and distinctness of our model from Valiant’s.

Our approach does have its disadvantages, the most obvious one being that **m** is not computable, but only enumerable. It turns out that we can scale down the entire theory as developed to more practically interesting classes of computable distributions, for instance, polynomially computable ones. Treating polynomial time computable distributions as the analogon of the simple distributions, we encounter the mathematically inelegant drawback that this class does not contain a universal distribution - there is a universal distribution for this class but not in it. It has been shown, however, that the class of polynomial samplable distributions, as defined in [BCGL], contains a universal distribution. One may also further restrict our assumption to even narrower classes to make the theory practically usable. An entirely similar set of considerations holds for the continuous incarnation **M** of **m**.

It seems likely that many simple concepts previously polynomially unlearnable become polynomially learnable in our model. We have given evidence for this by several examples. Is $\log n$ decision list - in analogy with $\log n$ DNF - polynomially learnable in our model? The connection between our approach of sampling under **m** and learning via queries is obvious, but has not been treated here. Many other classes, such as monotone DNF formulae, are also attractive candidates that may be learnable in our model.

We view this paper as another step towards a viable mathematical theory for machine learning in the tradition Solomonoff - Gold - Valiant, as described in [LV2]. The ultimate aim is a theory supporting machine learning using few examples, rather than polynomially many.

Acknowledgements.

Peter Gács, Gloria Kissin, Ray Solomonoff, and John Tromp commented on the manuscript. John suggested the need for Lemma 2; Peter Gács suggested turning semimeasures like **m** and **M** into measures by concentrating the surplus probability on an undefined symbol. We thank the referees for their valuable comments.

References

[A] D. Angluin. Learning Regular Sets From Queries and Counter-examples, Yale University, Computer Science Department, Techn. Rept. TR-464, 1986.

[A1] D. Angluin, Inference of Reversible Languages, Journal of the ACM, 29(1982), pp. 741-765,

[BCGL] S. Ben-David, B. Chor, O. Goldreich, M. Luby, On the theory of average case complexity, Proceedings of the 21st ACM Symposium on Theory of Computing, 1989, pp. 204-216.

[BI] G. Benedek and A. Itai, Learnability by fixed distributions, Proceedings of the 1st ACM Workshop on Computational Learning Theory, 1988, pp. 80-90.

[BF] A. Biermann and J. Feldman. On the synthesis of finite-state machines from samples of their behavior, IEEE Trans. Comput. C-21(1972), pp. 592-597.

[BEHW] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, Classifying Learnable Geometric Concepts With the Vapnik-Chervonenkis Dimension, Proceedings of the 18th ACM Symposium on Theory of Computing, 1986, pp. 273-282.

[BEHW1] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, Occam's Razor, Information Processing Letters, 24(1987), pp. 377-380.

[G1] P. Gács, On the symmetry of algorithmic information, Soviet Math. Dokl., **15**(1974), pp. 1477-1481, (Correction, *Ibid.*, 15(1974), p. 1481)

[G2] P. Gács, Lecture notes on descriptional complexity and randomness, Manuscript, Boston University, Boston, Mass., October 1987 (Unpublished).

[Ha] P.R. Halmos, *Measure Theory*, Springer Verlag, 1974.

[HLW] D. Haussler, N. Littlestone, and M. Warmuth, Expected Mistake Bounds for On-Line Learning Algorithms, Proceedings of the 29th IEEE Symposium on Foundation of Computer Science, 1988, pp. 100-109.

[J] D. Johnson, Approximation algorithms for combinatorial problems, Journal of Computer and System Sciences, 9(1974), pp. 256-276.

[KL] M. Kearns and M. Li, Learning in the Presence of Malicious Errors, Proceedings of the 20th ACM Symposium on Theory of Computing, 1988, pp. 267-280.

[KLPV] M. Kearns, M. Li, L. Pitt, and L.G. Valiant, On the Learnability of Boolean Formulae. Proceedings of the 19th ACM Symposium on Theory of Computing, 1987, pp. 285-295.

[L] L.A. Levin, Laws of information conservation (nongrowth) and aspects of the foundation of probability theory, *Probl. Inform. Transm.*, **10**(1974), pp. 206-210.

[LV1] M. Li and P. Vitanyi, Kolmogorov complexity and its applications, pp. 187-254 in: *Handbook on Theoretical Computer Science, Vol. 1*, Jan van Leeuwen, Managing Editor, North-Holland, 1990.

[LV2] M. Li and P. Vitanyi, Inductive reasoning and Kolmogorov complexity, Proceedings of the

4th IEEE Structure in Complexity Theory conference, 1989, pp. 165-185.

[L] N. Littlestone, Learning Quickly When Irrelevant Attributes Abound: A New Linear Threshold Algorithm, Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science, 1987, pp. 68-77.

[Lo] L. Lovasz, On the ratio of optimal integral and fractional covers, Discrete Math, 13(1975), pp. 383-390.

[N] B.K. Natarajan, On Learning Boolean Functions, Proceedings of the 19th Symposium on Theory of Computing, 1987, pp. 296-304.

[PV] L. Pitt and L.G. Valiant, Computational Limitations on Learning From Examples, Journal of the ACM 35(1989), pp. 965-984.

[R] R. Rivest, Learning Decision-Lists, Machine Learning, 2(1987), pp. 229-246.

[RS] R. Rivest and R. Schapire, Diversity-based inference of finite automata, Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science, 1987, pp. 78-88.

[V] L. G. Valiant, A Theory of the Learnable, Comm. ACM, 27(1984), pp. 1134-1142.

[ZL] A.K. Zvonkin and L.A. Levin, The complexity of finite objects and development of the concepts of information and randomness by means of the theory of algorithms, Russian Math. Surveys, 25:6(1970), pp. 83-124.

Appendix: Simple Reversible Languages

We exhibit one more discrete concept class which is polynomially learnable under all simple distributions, and not known to be polynomially learnable under all distributions. A deterministic finite automaton (DFA) $\mathbf{A}=(Q,q_0,F,A,\delta)$ consists of a set of states Q, a finite nonempty input alphabet A, an initial state q_0 and a set of final states $F \subseteq Q$, and a transition function $\delta: Q \times I \rightarrow Q$. Reversible languages were defined by Angluin in [A1], see also [BF]. $\mathbf{A}$ is *0-reversible* if it has only one final state ($|F|=1$), and its reversal $\mathbf{A}^R$ is deterministic. ($\mathbf{A}^R$ is obtained from $\mathbf{A}$ by reversing each transition in $\mathbf{A}$ and exchange the initial and the final state of $\mathbf{A}$.) An alternate definition would be that $\mathbf{A}$ is *0-reversible* if it is deterministic with one initial state and one final state and for no q_1 and q_2 is $\delta(q_1,a)=\delta(q_2,a)$ for some $a \in A$. A language L is 0-reversible if it is accepted by a 0-reversible DFA.

Many languages/DFA's are 0-reversible and of low Kolmogorov complexity. Examples are, for fixed n, the language L_1= set of strings of length at least n and containing an even number of zeroes, and the language $L_2=\{0^k 1^j : k,j \geq n\}$.

A nondeterministic finite automaton (NFA) is like a DFA with q_0 replaced by $I \subseteq Q$, and $\delta: Q \times A \rightarrow 2^Q$. We generalize 0-reversibility as follows. A *k-reversible* DFA is a DFA $\mathbf{A}$ such that in (the possibly nondeterministic) $\mathbf{A}^R$, if two distinct states q_1,q_2 are initial states or $q_1,q_2 \in \delta(q_3,a)$ for $a \in A$, then no string u of length k satisfies both $\delta(q_1,u) \neq \emptyset$ and $\delta(q_2,u) \neq \emptyset$. This guaranties that any nondeterministic choice in the operation of $\mathbf{A}^R$ can be resolved by looking ahead k symbols past the current one. A language is k-reversible if it is accepted by a k-reversible automaton.

For each fixed n, $L_3=\{0^k 1^m : k \geq n,\ m \geq 1\}$ and $L_4=\{0^m 1^k : k \geq n,\ m \geq 1\}$ are 1-reversible and have $O(\log n)$ Kolmogorov complexity. We say a *path* from the initial state to a final state is *simple* if it has Kolmogorov complexity $O(\log n)$. A k-reversible DFA $\mathbf{A}$ is *simple* if each state of $\mathbf{A}$ lies on a simple path.

Example. We show that the set of k-reversible DFA's of Kolmogorov complexity $O(\log n)$

are simple. To see this, consider **A** such that $K(\mathbf{A}) = c \log l(\mathbf{A})$. We show that every state of **A** is on a simple path of Kolmogorov complexity at most $(c+1)\log l(\mathbf{A})$. Without loss of generality, assume that every state of **A** is reachable from the initial state. If this is not the case, we can just delete those obsolete states from **A**. We have assumed that **A** can be specified in $c\log n$ bits. Fix an enumeration of the paths from the initial state to final states of **A** such that each path contains at least one more new state. There are at most $n = l(\mathbf{A})$ such paths since each path must contain at least one new state which is not contained in the previous paths. Obviously, each such path can be specified using **A**, that is, $c\log n$ bits, and the index of the path in $\log n$ bits. Hence the Kolmogorov complexity of each such path is at most $c \log n + \log n$. Every state is on at least one of these paths by construction.

Notice that if $K(\mathbf{A}) = O(\log n)$, it is still possible that **A** may have very random paths. For example, the automaton which accepts all strings of length n has Kolmogorov complexity $O(\log n)$, but it actually contains a path for every string of length n. In particular, it contains a path of Kolmogorov complexity n. On the other hand, one can construct (left to the reader) a simple 0-reversible DFA which has Kolmogorov complexity much larger than $\log n$ (like $\Omega(\log^2 n)$).

In the general Valiant distribution-independent setting, it is not known whether the class of 0-reversible languages is learnable. Angluin [A1] shows that the set of k-reversible languages can be identified in the *limit* in the Gold paradigm. A DFA is polynomially learnable by membership queries and equivalence queries [A].

Theorem 9. *The class of simple k-reversible automata is polynomially learnable under* **m** .

Proof. We first show how to learn a simple 0-reversible DFA under the universal distribution.

Claim 1. The class of 0-reversible DFA of Kolmogorov complexity $O(\log n)$ is polynomially learnable under **m**.

Proof. The following algorithm and proof use ideas in [A1].

Learning Algorithm.

(1) Randomly sample n^{c+2} positive examples. Construct the trivial tree DFA from these examples. (See Figure 1 for an example.)

(2) Merge all the final states in above tree.

(3) *repeat*

if there are states $p,q \in Q$ such that on input $a \in A$, p,q lead to the same state, then merge p and q

until no more merges.

We prove that this algorithm correctly infers the underlying DFA **A** with high probability. Each positive example represents a path from q_0 to q_f, where some states may be repeated because of loops in **A**.

By previous arguments, we know that with high probability (at least $1-1/e^{O(n)}$) each string corresponding a simple path is sampled.

Claim 1.1. Given all simple paths of **A**, **A** can be inferred from the above algorithm.

Proof. All states of **A** are presented at least once in the tree constructed above. It is up to the algorithm to merge them correctly. Now between any two simple paths, P_1 and P_2, if there is a transition from a state a on P_1 to a state b on P_2, then the path from q_0 to a via P_1 then to b then to q_f via P_2 is also simple, hence also given, hence the above merging process will add a transition from a to b correctly. Since this applies to all transitions, eventually **A** will be correctly inferred. As for the non-simple strings, some of which may also be sampled since they have higher than $1/(\log n)^2$ probability in total, will also fit into the structure. Notice that all above merges do not introduce mistakes since we are dealing with 0-reversible DFA's. □

This also finishes the proof of Claim 1. □

Claim 2. For each k, the class of simple k-reversible DFA is polynomially learnable under **m**.

Proof. A generalization of the algorithm and the proof in Claim 1 shows that simple k-reversible languages are polynomially learnable under **m**, for each fixed k. We refer the readers to [A1] for more details. □

We now show how to learn the class of simple k-reversible languages for all k. The algorithm is given as follows:

for k:=1 to ∞ do

(1) Apply the algorithm for learning k-reversible language to learn a k-reversible DFA;

(2) Draw a polynomial, in the size of above derived DFA, number of examples, according to $\mathbf{m}(x)$, to test the inferred automaton;

(3) *if* the DFA learned is consistent with $(1-\varepsilon/2)$ fraction of the test set

then output this DFA
else continue with the next k value;

At step (3), if the DFA is consistent with $(1-\varepsilon/2)$ fraction of the test set, then applying Occam's Razor theorem, the DFA we have learned approximates the real DFA with high probability. Notice that it is not necessary that the real k-reversible DFA is inferred. The rest of the correctness and complexity analysis are standard and similar to that in [A1], so we again refer the reader to [A1]. □