Learning Functions from Examples

B. K. Natarajan

CMU-RI-TR-87-19

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

August 1987

Table of Contents

1. Introduction 1
2. Problem Solving: An Example 2
3. Preliminaries 3
4. Uniformly Convergent Learning 3
5. Functions over Continuous Spaces 8
6. Two Familiar Function Families 9
7. Conclusion 13
8. Acknowledgements 13
9. References 13

Abstract

This paper concerns algorithms that "learn" functions from examples. Functions on strings of a finite alphabet are considered and the notion of dimensionality defined for families of such functions. Using this notion, a theorem is proved identifying the most general conditions under which a family of functions can be efficiently learned from examples. Turning to some familiar families: we present strong evidence against the existence of efficient algorithms for learning the regular functions and the polynomial time computable functions, even if the size of the encoding of the function to be learned is given. Our arguments hinge on a new complexity measure – the constraint complexity.

1. Introduction

This paper concerns algorithms that "learn" functions from examples. In the main, it is a sequel to the material in [Natarajan 1987] and contains the results presented in [Natarajan 1987b]. The problem has been of interest over the years to workers in artificial intelligence, pattern recognition and numerical analysis. Specifically, we are interested in computing uniformly good approximations to an unknown function, based on its behaviour on a few sample points. This problem is known as interpolation in numerical analysis, pattern matching in pattern recognition and concept learning (amongst others) in artificial intelligence. As our motivation for this study was drawn from artificial intelligence, we will use the term "learning" instead of the other two.

We begin with an example to motivate our work. Consider the problem of learning integral calculus. Given a table of integrals, one has all the information theoretically required to become an expert. Yet, worked examples and practice problems seem to be necessary before one acquires any facility over the domain. We formalize this problem and show that unless $P = NP$, examples play an important role in such learning. Our formalism covers many other domains such as learning to solve puzzles, play games etc. We then argue that it is convenient to view our formalism as an algorithm that learns functions from examples.

The problem of inferring Turing machines from sample computation traces has been studied before [Biermann 1974], but issues of feasibility or correctness have not been addressed. More recently, a general framework for uniformly convergent learning of simple concepts was proposed [Valiant 1984]. Based on this framework, some general results on learning geometric concepts and boolean functions followed [Blumer et al. 1986, Natarajan 1987]. Within the same framework, we consider length preserving functions on strings of a finite alphabet. We define the notion of dimensionality for families of such functions and give a general theorem that states that a family of such functions can be efficiently learned if and only if it is of polynomial dimension. This is an important contribution of the paper. Our approach is similar to the one in [Natarajan 1987] and aims at ease of understanding and intuitive appeal. Turning to functions on continuous spaces, we extend the results on learning boolean-valued functions [Blumer et al. 1986] to general functions.

We then consider two familiar function families: the regular functions and the polynomial-time computable functions. Since these families are not of polynomial dimension, we consider parametrized subsets of these families, the parameter being the bound on the size of the encodings of the functions. We measure the encoding size as the number of states in the deterministic finite automaton computing the function for regular functions, and as the size of the program in some admissible programming system for the polynomial-time computable functions. We then look for learning algorithms that run in time poynomial in the size bound. (Summarizing the above, when attempting to learn an unknown function, is it sufficient to know that the function is regular (or polynomial-time computable) and that it has a short encoding, in order to learn it efficiently?)

For the regular functions, we show that such an algorithm does not exist, unless NP = RP. Our

argument is based on an earlier result on the complexity of ordering the regular sets [Gold 1978, Angluin 1978].

For the polynomial-time computable functions, we argue that it is unlikely that such an algorithm exists. Our argument is not reducible to the condition "unless NP=RP", but is almost as strong, and proceeds as follows. To start with, we introduce the interesting notion of the Constraint Complexity of a set of examples – a measure of the information carried by the set. This is the second important contribution of the paper. As a backdrop, we prove many interesting results with this tool, including a short and intuitive proof of the dimensionality theorem mentioned earlier. We then argue that since the traditional notion of Kolmogorov complexity is a special case of our notion and there are no known algorithms for efficiently computing the polynomial-time bounded Kolmogorov complexity, it is unlikely that we can construct one for our measure. From this we deduce that an efficient learning algorithm for the polynomial-time functions is rather unlikely.

2. Problem Solving: An Example

Many problems such as learning integral calculus, learning to solve puzzles, games etc can be expressed as follows.

A problem domain D is the triplet $\{L,M,N\}$ where

(a) L, the problem set, is any set of strings.

(b) M is a finite and fixed set of operators $m_1,...m_k$ where each m_i is a function from L to L.

(c) N is the goal predicate, a boolean-valued function on L. A problem p in L is solved if $N(p) = 1$.

If D were the domain of integral calculus, L would be all integrals, M a table of integrals, and N the rule "problem is solved iff it does not contain integral signs".

A solution of any problem p is $\sigma(p)$, where σ is any sequence of operators from M such that $N(\sigma(p)) = 1$. A problem solver for a domain is an algorithm that takes as input a problem and produces as output a solution of the problem.

Our interest is to construct a meta-algorithm for any given set of domains H, that would take as input a domain D from H and, after some pre-computation, behave like a problem-solver for D. We now show that if $P \neq NP$, even the simplest of domains will not possess an efficient meta-algorithm, unless the meta-algorithm is allowed to see solved examples for its input domains.

Example: Consider the set of domains H defined as follows: Any $D = \{L,M,N\}$ in H is such that

(a) the problem set $L = \{x\#y | x, y \in \{0,1\}^*$ and # is a special symbol$\}$.

(b) operator set $M = \{m_1, m_2, m_3\}$, where

for $x, y \in \{0,1\}^*, a \in \{0,1\}$

$m_1(x\#ay) = x0\#y$,

$m_2(x\#ay) = x1\#y$,

$m_3(x\#y) = \#xy$.

(c) N is a boolean function constructed as follows. Let N' be a boolean function of n variables and let $p \in L$.

$N(p)$ = if $|p| = n+1$, strip off the # and evaluate N' on the resulting boolean vector.

= 0 otherwise.

In essence, each domain in H is characterized by the boolean function that is its goal predicate. Let $D = \{L,M,N\}$ in H and let N be a function of n variables. Now, if a is a satisfying assignment of N, then a is a solution to every problem of length $n+1$ in L. If N is not satisfiable, then no problem in L has a solution. Hence, every domain in H trivially has a problem-solver, but a meta-algorithm on H is going to have to decide on the satisfiability of boolean formulae. Clearly, an intractable problem. On the other hand, if the meta-algorithm is allowed to see solved examples for the input domain, then it can trivially decide whether or not the goal predicate has a satisfying assignment, and then act accordingly. •

If the meta-algorithm is allowed to see a few examples, say pairs of the form (*problem*, *solution*), and then be required to compute the function that maps each problem to its solution, the entire process can be viewed as learning a good approximation to a function from examples of its behaviour. This is exactly the problem we study below.

3. Preliminaries

Without loss of generality, let Σ be the binary alphabet and Σ^* the set of all binary strings. We consider functions from Σ^* to Σ^*. An example of a function f is a pair $(x, f(x))$. A *learning algorithm* is an algorithm that attempts to infer a function from examples for it. The learning algorithm has at its disposal a routine EXAMPLE, that at each call produces an example for the function to be learned. The probability that a particular example (x,y) will be produced by a call of EXAMPLE is $P(x)$, as given by the probability distribution P. Also, the probability that the learned function will be queried on a particular string x is $P(x)$. The distribution P can be arbitrary and unknown.

We define a *family* of functions F to be any set of length preserving functions from Σ^* to Σ^*. The n^{th}-subfamily F_n of a family F, is the family of functions induced by F on Σ^n. Specifically, if $F = f_1, f_2, \ldots f_i, \ldots$, then $F_n = g_1, g_2, \ldots g_i, \ldots$ where g_i is defined as follows.

$g_i(x) = f_i(x)$ if $|x| = n$

undefined otherwise

4. Uniformly Convergent Learning

4.1 Learnability

Following [Valiant 1984], we say that a family of functions is learnable if there exists a uniformly convergent learning algorithm for it. Specifically, a family of functions F is *learnable* if there exists a learning algorithm that

(a) takes as input integers n and h.

(b) makes polynomially many calls of EXAMPLE, both in the adjustable error parameter h and in the problem size n. EXAMPLE produces examples of some function in F_n.

(c) For all functions f in F_n and all probability distributions P on Σ^n, with probability $(1-1/h)$ the algorithm outputs a function g in F_n such that

$$\sum_{x \in S} P(x) \leq 1/h$$

where $S = \{x \mid |x| = n \text{ and } f(x) \neq g(x)\}$

Furthermore, if the learning algorithm runs in time polynomial in n and h, we say that the family is *polynomial-time learnable.*

We need the following definitions as well.

A function f is *consistent* with a set of examples S if $(x,y) \in S$ implies $f(x) = y$.

An *ordering* O_n of a sub-family F_n is an inclusive, onto mapping from sets of examples to F_n. Specifically,

(a) $O_n: 2^{\Sigma^n \times \Sigma^n} \rightarrow F_n$.

(b) inclusive: For any $S \subseteq \Sigma^n \times \Sigma^n$, if there exists $f \in F_n$ consistent with S, then $O_n(S)$ is defined and is consistent with S.

(c) onto: For all f in F_n, there exists $S \subseteq \Sigma^n \times \Sigma^n$ such that $O_n(S) = f$.

An ordering O of a family F is a sequence of sub-orderings $O_1, O_2, \ldots, O_n, \ldots$ such that O_n is an ordering of F_n, the n^{th}sub-family of F. An ordering O is a *polynomial-time ordering* if there exists a polynomial $T(n)$ such that each sub-ordering O_i of O runs in time $T(n)$ on inputs of length n.

The *width* of an ordering O of a sub-family F_n is the least integer w such that for all f in F_n there exists a set S of w or fewer examples for which $O(S) = f$.

The *dimension* of a sub-family F_n is the least integer d for which there exists an ordering of F_n of width d. A family F is of dimension $D(n)$ if there exists an ordering O of F such that for all n, the n^{th}sub-ordering O_n of O orders F_n in width $D(n)$ or less. If $D(n)$ is a polynomial in n, F is said to be of *polynomial dimension* and O of *polynomial width.*

A set S of examples is *shattered* by a family F if for any $S_1 \subseteq S$ there exists $f \in F$ such that f is consistent with S_1 but not consistent with any non-trivial subset of $S-S_1$.

Remark If $|F_n| \geq 2^k$ for some k, then the dimension of $F_n \geq k/(2n)$.

We are now ready for our first result.

Lemma 1: Let F_n be a subfamily of dimension d. Then there exists a set S of d examples that is shattered by F_n.

Proof: Let O be an ordering for F_n. We first modify O to obtain O^1 as follows.

function $O^1(G$:set of examples)
  Let $C_1, C_2,...,C_i,...$ be sets of examples in
  increasing size and in some canonical order.
  for $C_1, C_2,...$**do**
    if $O(C_i)$ is consistent with G
      then return $O(C_i)$.
  od
end

It is easy to see O^1 is an ordering for F_n as well. Pick a function f in F_n such that

$\forall S: O^1(S) = f$ implies $|S| \geq d$.

Let S be a set such that $f = O(S)$. Now $|S| \geq d$. Suppose there exists a set $S_1 \subset S$ such that any g in F_n consistent with S_1 is also consistent with some non-trivial subset of $S-S_1$. Then, $O^1(S_1) = O^1(S_2)$ for some $S_1 \subset S_2 \subset S$. Modify O^1 to O^2 as follows.

$O^2(G) = O^1(S)$ if $G = S_2$
$O^1(G)$ otherwise.

Now O^2 is also an ordering of F_n except that there is now a set S_2, $|S_2| < |S|$ such that $O^2(S_2) = f$. We can repeat this process for other functions in F_n, eventually reducing the width of the ordering. Since the width cannot be reduced below d, there must be some set of size d or greater that is shattered by F_n. Which implies that there is a set of size d shattered by F_n. Hence the lemma. •

Corollary $|F_n| \geq 2^k$ for some k implies that $\exists S, |S| \geq k/(2n)$ that is shattered by F_n. •

We are now ready for our main theorem.

Theorem 1: A family of functions F is learnable if and only if it is of polynomial dimension.

Proof: **(If)** Let O be an ordering for F of width $D(n)$, where $D(n)$ is some polynomial in n. The following is a learning algorithm for F.

Algorithm 1

Input: n, h.

begin
  Call EXAMPLE $2hn(D(n)+1)$ times.
  Let S be the set of examples obtained.
  Output $O(S)$.
end

Algorithm 1 is correct as reasoned below. Let f in F_n be the function to be learned, i.e, the function for which EXAMPLE provides examples and let P be the probability distribution on Σ^n. For any g in F_n, define the residue r_g of g as follows.

$$r_g = \sum_{x \in S_g} P(x)$$

where $S_g = \{x | g(x) \neq f(x)\}$.

Let C_f be the set given by

$$C_f = \{g | g \in F_n \text{ and } r_g > 1/h\}$$

i.e., C_f is the set of functions in F_n that differ from the function to be learned with probability exceeding $(1/h)$. The probability that Algorithm 1 outputs a function from C_f should be bounded by $(1/h)$. The probability that m calls of EXAMPLE will produce examples all consistent with some particular function in C_f is bounded by $(1-1/h)^m$. Now,

$$|C_f| \leq |F_n| \leq 2^{2nD(n)}.$$

Hence the probability that m calls of EXAMPLE will produce examples all consistent with any one function in C_f is bounded by $2^{2nD(n)}(1-1/h)^m$. Therefore, if m satisfies

$$2^{2nD(n)}(1-1/h)^m \leq 1/h$$

and Algorithm 1 calls EXAMPLE m times, Algorithm 1 will be within the allowable error with high probability. Simplifying, we get

$$m > h(2nD(n) + log(n)),$$

which is satisfied for $m = 2hn(D(n)+1)$ as in Algorithm 1. Hence, Algorithm 1 learns F and since $D(n)$ is polynomial in n, F is learnable.

(only if) Let F be of super-polynomial dimension $D(n)$ and let A claim to be a learning algorithm for F. Let A call EXAMPLE $(nh)^k$ times on input n, h. Pick n, h such that

$$d = D(n) > (nh)^k/(1-2/h).$$

By Lemma 1, there exists a set S of $D(n)$ examples that is shattered by F_n. Place the uniform probability distribution

$$P(x) = 1/d \text{ if } (x,y) \in S$$
$$= 0 \text{ otherwise}$$

on S and run A on it. Now, on any $m = (nh)^k$ calls of EXAMPLE, A will see at most m elements of S. Let S_1 be the set of examples seen. Let g be the function output by A and let f be the function to be learned. Since S is shattered by F_n, there at least (2^{d-m}) possibilities for f that are consistent with the examples seen by A. On each element of $(S-S_1)$, g will differ with at least half the possibilities for f. Therefore, the total number of differences over all the possibilities for f is at least $(2^{d-m}(d-m)/2)$, and the average is $(d-m)/2$. This average must be attained or exceeded on at least one possibility for f. Hence, there exists a function f for which the function g output by A always differs from f on at least $(d-m)/2$ of the elements of S. The probabilistic weight of this difference is

$$(d-m)/2d > 1/2 - (1-2/h)/2 > 1/h,$$

which is more than the allowable. Hence A does not learn F.

This completes our proof. •

Finally, we present a resource bounded version of Theorem 1. Theorem 2 concerns time complexity, but other resource bounds may be treated similarly.

Theorem 2: A family of functions F is polynomial time learnable if and only if F has a polynomial-time ordering of polynomial width.

Proof: Straightforward extension of Theorem 1. •

Remarks The results presented in [Blumer et al. 1986, Natarajan 1987] concern learning sets from samples of their elements. It is easy to see that sets are encodable as boolean-valued functions and hence can be treated as a special case of our theorem. Conversely, a function from $\{0,1\}^n$ to $\{0,1\}^n$ can be viewed as a combination of n boolean-valued functions on $\{0,1\}^n$, and hence learning functions can be viewed as a special case of learning sets.

In our development, we used a discrete metric to measure the distance between two functions on an input string – two functions agreed on a string or did not. It is worth mention that our arguments carry through for any standard metric.

The following is a resource bounded, weak form of Theorem 1.

Theorem 2: A family of functions F is polynomial time learnable (1) if F has an ordering of polynomial width computable in polynomial time. (2) only if F has an ordering of polynomial width computable in random-polynomial time.

Proof: Straightforward extension of Theorem 1. •

4.2 Properties of the Dimension

For any family of functions F, let $dim(F)$ denote the dimension of F. Let A and B be two families of functions such that $dim(A), dim(B) \geq 1$.

Lemma 2: If $C = A \cap B$, then $dim(C) \leq min(dim(A), dim(B))$.

Proof: Immediate. •

Let A and B be two families from $X_1 \rightarrow Y_1$ and $X_2 \rightarrow Y_2$ respectively. Then $C = A \times B$ is the family of functions from $X_1 \times X_2 \rightarrow Y_1 \times Y_2$ such that each function in C is the product of some two functions in A and B. i.e

$C = \{a \times b \mid a \in A, b \in B\}$
where $a \times b$ is defined as follows:
For all $(x_1, x_2) \in X_1 \times X_2$,
$(a \times b)(x_1, x_2) = (a(x_1), b(x_2))$

Lemma 3: If $C = A \times B$, then $dim(C) \leq dim(A).dim(B)$

Proof: Straightforward. •

Lemma 4: If $C = A \cup B$ then $dim(C) \leq max(dim(A),dim(B))+1$.

Proof: Without loss of generality, let $dim(A) \geq dim(B)$. Combine the minimum width orderings O_A, O_B for A and B to obtain an ordering O_C for C as follows.

function O_c (S: set of examples)

begin
If $|S| \leq dim(A)$
    then return $O_A(S)$
    else return $O_B(S)$
end

Clearly, O_C is an ordering for C of width $dim(A) + 1$. •

Lemma 5: Let $A = \{a_1, a_2, ...a_i,...,\}$ be a family of $\{0,1\}$-valued functions and let $\overline{A}$ be the family $\{\overline{a}_1,\overline{a}_2,...\overline{a}_i,...,\}$ where $\overline{a}_i = 1-a_i$. Then, $dim(A) = dim(\overline{A})$.

Proof: Immediate. •

5. Functions over Continuous Spaces

5.1 Learnability

As our results are derived using information theoretic methods, it is impossible to extend them directly to continuous spaces where each example can be of infinite length. On the other hand, the results in [Blumer et al 1986] for learning boolean-valued functions are obtained using some classical results in probability theory and are valid over continuous spaces. Hence, we will concentrate our efforts on extending their results to arbitrary functions.

As in [Blumer et al. 1986], we define the *Vapnik-Chervonenkis* dimension $d_{vc}(F)$ of a family F as follows.

For any set of examples S, define the set $\Pi_F(S)$ as the set of all subsets of S obtained by intersecting S with the functions in F. i.e

$\Pi_F(S) = \{R | R \subseteq S$, and $\exists f \in F$ such that f agrees with S on R and disagrees with S on $S-R\}$.

If $\Pi_F(S) = 2^S$, we say that F *shatters* S. $d_{vc}(F)$ is the smallest integer d such that no set of cardinality $d+1$ is shattered by F.

Since we no longer need the notion of a sub-family, we modify our definition of learnability accordingly. In particular, a family of functions F is learnable if there exists an algorithm that

(a) takes as input an integer h,

(b) makes polynomially many calls of EXAMPLE, polynomial in the adjustable error parameter h.

(c) as in the earlier definition of learnability.

With these definitions in hand, we can state the following theorem.

Theorem 3: For any finite alphabet Σ, a family of functions from Σ^* to Σ^* is learnable if and only if it is

finite Vapnik-Chervonenkis dimension.

Proof: The proof of this theorem is similar to the proof of the corresponding theorem for boolean valued functions [Blumer et al. 1986]. •

5.2 Properties of the Dimension

Lemmas 2, 3, and 5 stand in their present form for the Vapnik-Chervonenkis dimension as well. Lemma 4 needs to be rewritten as follows.

Lemma 4': If $C = A \cup B$ then $d_{vc}(C) \leq d_{vc}(A)+d_{vc}(B)$.

Proof: Let $d_{vc}(A) = d_A$ and $d_{vc}(B) = d_B$. Let S be any set of examples such that $|S| = s > d_A + d_B$. Since $C = A \cup B$,

$\Pi_C(S) = \Pi_A(S) \cup \Pi_B(S)$

Hence,

$|\Pi_C(S)| \leq |\Pi_A(S)| + |\Pi_B(S)|$

By Lemma 1 of [Vapnik and Chervonenkis 1971],

$$|\Pi_A(S)| \leq \sum_{k=0}^{d_A} \binom{s}{k}$$

and

$$|\Pi_B(S)| \leq \sum_{k=0}^{d_B} \binom{s}{k}$$

Hence

$$|\Pi_C(S)| \leq \sum_{k=0}^{d_A} \binom{s}{k} + \sum_{k=0}^{d_B} \binom{s}{k}$$

$$|\Pi_C(S)| \leq \sum_{k=0}^{d_A} \binom{s}{k} + \sum_{k=s-d_B}^{s} \binom{s}{k}$$

$$< 2^s.$$

Hence C cannot shatter S if $|S| > d_A + d_B$ implying that $d_{vc}(C) \leq d_A + d_B$ as claimed. •

6. Two Familiar Function Families

We now turn our attention to two familiar function families – regular sets and the polynomial-time computable functions. Our interest here is to construct learning algorithms for these families. Since these families are of exponential dimension, we modify our definition of learnability to be meaningful in this context. The motivation behind our definition is as follows. Suppose that we are trying to learn an unknown function from examples and are told only that the function is regular (or computable in polynomial time) and is accepted by an deterministic finite automaton of d states (has an encoding of length d). Is this information sufficient to enable us to efficiently learn the function?

Let F be a family of functions with a measure on the size of the encodings for each function in the family. For any integer d, let $f_1^d, f_2^d, \ldots f_i^d, \ldots$ be the functions in F of size d. Then, for any n, the

nth-subfamily F_n^d of F with respect to d is the set of functions $g_1, g_2, \ldots, g_i, \ldots$ where

$g_i(x) = f_i^d(x)$ if $|x| = n$

= undefined otherwise

The family F is learnable if there exists an algorithm A that

(a) takes as input: problem size n, error parameter h and output size d.

(b) runs in time polynomial in n, h, d. EXAMPLE provides examples for some function in F_n^d.

(c) for all functions f in F_n^d and all probability distributions P on Σ^n, with probability $(1-1/h)$ the algorithm outputs a function g in F_n^d such that

$$\sum_{x \in S} P(x) \leq 1/h$$

where $S = \{x \mid |x| = n \text{ and } f(x) \neq g(x)\}$

We say that A learns F.

From Theorem 2 we know that in order to construct a learning algorithm for F in the above sense, we only need construct an efficient ordering for F, i.e, given a set S of examples for some function in F_n^d, we should be able to efficiently compute a function in F_n^d consistent with S.

6.1 Regular Functions

We extend the notion of regular sets to that of regular functions, by considering Mealy machines [Hopcroft & Ullman 1979] instead of accept/reject finite automata. Specifically, we associate a character of the alphabet with transition of the automaton and this character is output each time that transition is completed. The function value for a string is the output obtained by running the automaton on the string. Our regular functions are from Σ^* to Σ^* and are length preserving.

We now consider the issue of efficiently ordering the regular sets. Define the encoding size of a regular function to be the size of the minimal automaton that computes the function. We need to answer the following question: given a set of examples S and an integer d, find a deterministic finite automaton of size d, consistent with S. This is equivalent to finding the minimal deterministic finite automaton consistent with the given set of examples. Unfortunately, this problem is NP-complete as shown by [Gold 1978; Angluin 1978]. Consequently, we conclude that it is unlikely that the regular functions are learnable as claimed below.

Claim: If the regular functions are learnable as defined above, then P=RP.

Proof: If the regular functions were learnable, then we could order them in random polynomial time. But, as reported in [Gold 1978], ordering the regular functions is an NP-complete problem. Hence the claim. •

6.2 The Polynomial-time Computable Functions

We consider the family of all length preserving, polynomial-time computable functions. To develop some tools for our arguments, we first look at the family of all computable functions.

Consider the problem of ordering the computable functions. Let the encoding size of a function be the size of the shortest program computing the function in some admissible programming system, say the Turing machine system. We need to be able to compute: given a set of examples S and an integer d, find a program of size d consistent with S – a problem that is equivalent to computing the minimal program consistent with S. This leads us naturally to the notion of the *constraint complexity* $G(S)$ of a set S of examples.

$G(S) = min_d \exists z, |z|=d$ and $\forall (x,y) \in S: M_u(z,x)=y$

where M_u is the universal program. In words, $G(S)$ is the size of the shortest program consistent with S. Contrast this with the definition of the Kolmogorov complexity of a string x, [Hartmanis 1983].

$K(x) = min_d \exists z, |z|=d$ and $M_u(z) = x$.

If S is a set of examples for a function f, $G(S)$ aims at measuring the amount of information about f carried by S. This is brought out in the following propositions.

Proposition 1: For any string x

$G((0^{|x|}, x)) \leq K(x) \leq G((0^{|x|}, x)) + log(|x|)$.

Proposition 2: If S is a set of examples for a program p, then,

$G(S) \leq |p|$

$G(S) \leq K(p) + c$

where c is a small constant.

Proposition 2 tells us that the information carried by a set of examples is bounded by the shortest description for the program generating the examples. Extend the notation $G(S)$ to $G(f)$ where f is a function, as follows: $G(f)$ is the length of the smallest program consistent with any set of examples for f, i.e., $G(f)$ is the length of the shortest program computing f.

Proposition 3: Let f, g be two functions on Σ^n. Then f and g differ on at least $|G(f)-G(g)|/2n$ strings.

Proof: Without loss of generality, let $G(f) < G(g)$ and let f and g differ on fewer than $(G(f)-G(g))/2n$ strings. If p_f is the minimal program for f, construct a program p_g for g by simply tagging on a table of differences to p_f. The length of this tag is at most $2n(G(f)-G(g))/2n = G(f)-G(g)$ and hence p_g is a program for g that is shorter than $G(g)$. A contradiction and hence the proposition. •

To illustrate the power of the notion of constraint complexity, we prove the following version of the *only if* part of Theorem 1.

Theorem 1': Let F be a family of functions and let F_n be of dimsension $D(n)$. Then, no algorithm that calls EXAMPLE $T(n)$ times where $\lim_{n \to \infty} nT(n)/D(n) = 0$, can learn F.

Proof: Let A be a learning algorithm for F calling EXAMPLE $T(n)$ times, where $\lim_{n \to \infty} nT(n)/D(n) = 0$. Pick n such that $2nT(n)+|A| \ll D(n)$. Since F_n is of dimension $D(n)$, $|F_n| > 2^{D(n)}$ and hence there exists a

function f in F_n such that $G(f) \geq D(n) \gg T(n)$. But, for any set S of $T(n)$ examples for f, $G(S) \leq 2nT(n)$ and hence any function g output by A is such that $G(g) \leq G(S) + |A| \leq 2nT(n) + |A| \ll D(n)$. By Proposition 3, g differs from f on too many strings, and hence A cannot learn F for the uniform distribution on Σ^n. •

As the reader might expect, the constraint complexity of sets of examples is badly noncomputable, displaying many of the strong properties of Kolmogorov complexity.

Proposition 4: The set $\{S \mid S$ is a set of examples and $G(S) \geq |S|/2\}$ is immune, i.e., there exists no computable set that enjoys an infinite intersection with the above set.

Proof: Similar to the corresponding result for Kolmogorov complexity. See [Natarajan 1985] for example. •

Returning to the realm of polynomial-time computable functions, we introduce the time-bounded constraint complexity. For a set of examples S and time bound $T(n)$,

$G^{T(n)}(S) = min_d \exists z, |z|=d$ and $\forall (x,y) \in S: M_u^{T(n)}(z,x) = y$,

where $M_u^{T(n)}$ is the $T(n)$ time bounded universal program. Hence, to order the functions computable in $T(n)$ time, we need to able to compute $G^{T(n)}(S)$ for any set S of examples. Unfortunately, the best algorithm known is

Proposition 5: $G^{T(n)}(S)$ is computable in non-deterministic time $|S|T(|S|)$.

Proof: Since $G(S) \leq (|S| + c)$ for some constant c, simply guess a string of that length and verify consistency with S. •

If $T(n)$ were a polynomial in n, $G^{T(n)}(S)$ is computable in non-deterministic polynomial time, NP. As argued below, we do not know if we can push it into random polynomial time RP, or deterministic polynomial time P.

Proposition 6: If $T(n)$ is a polynomial, $G^{T(n)}$ is computable in NP, but not known to be in P or RP.

Proof: From Proposition 1 and the fact that it is not known whether polynomial-time Kolmogorov complexity is in RP or P. •

In the light of the above, we cannot give a deterministic polynomial time ordering for the polynomial-time computable functions. In fact, we cannot even offer a randomized polynomial-time algorithm. Consequently, we cannot give a deterministic polynomial-time learning alogorithm for the poynomial-time computable functions. Indeed, it seems unlikely that such an algorithm exists. We can, however, give a non-deterministic polynomial-time algorithm as follows.

Algorithm 2

Input: problem size n, error parameter h,
output size d and time bound n^k.

begin
Call EXAMPLE dh times. Guess a string of length d
and verify that $M_u{}^{n^k}(d,)$ is consistent with the
examples seen.
If so, output the string.
end

---

7. Conclusion

This paper concerns algorithms that learn functions from examples. We considered length preserving functions on strings of a finite alphabet and defined the notion of dimensionality for families of such functions. Using this notion, we proved a general theorem that identifies the conditions under which a family of such functions can be efficiently learned. This theorem was extended to functions on continuous spaces by generalizing the notion of the Vapnik-Chervonenkis dimension introduced in [Blumer et al 1986]. We then considered the families of regular functions and the polynomial time computable functions. We showed that efficient algorithms for learning the regular functions do not exist. We also argued that it is unlikely that efficient algorithms exist for the polynomial-time computable functions. In doing so, we introduced the notion of the constraint complexity of a set of examples, a notion that is not only intuitively pleasing, but a useful tool as well.

8. Acknowledgements

I thank T.M. Mitchell for giving freely of his time.

9. References

Angluin, D, (1978) "On the Complexity of Minimum Inference of Regular Sets", *Information and Control*, 39, pp337-350.

Biermann, A.W., (1974), "On the Inference of Turing Machines from Sample Computations", *Artificial Intelligence*, vol3, pp181-198.

Biermann, A.W., and Feldman, J.A., (1972) "On the Synthesis of Finite State Machines from Samples of their Behavior", *IEEE Transactions on Computers* June, pp592-596.

Blumer A., Ehrenfeucht, A., Haussler D., & Warmuth, M., (1986), "Classifying Learnable Geometric Concepts with the Vapnik-Chervonenkis Dimension", ACM Symposium on Theory of Computing, pp273-282.

Gold, E.M., (1978), "Complexity of Automaton Identification from Given Data", *Information and Control*, 37, pp302-320.

Hartmanis, J., (1983), "Generalized Kolmogorov Complexity", IEEE Symposium on Foundations of Computer Science.

Hopcroft, J.E., and Ullman, J.D., "Introduction to Automata Theory, Languages and Computation", Addison-Wesley, 1979.

Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T., (1980), "Explanation Based Generalization: A Unifying View", *Machine Learning*, Vol 1, No 1, January.

Natarajan, B.K., (1985) "The Homogenous Capture of Random Strings", Cornell U. CS Tech. Report.

Natarajan, B.K., (1987) "On Learning Boolean Functions", ACM Symposium on Theory of Computing, pp296-304.

Natarajan, B.K., (1987b), Machine Learning Lecture, Dept. of Computer Science, Carnegie-Mellon University.

Valiant, L.G., (1984) "A Theory of the Learnable", ACM Symposium on Theory of Computing, pp436-445.

Vapnik, V.N., and Chervonenkis, A.YA., (1971), "On the Uniform Convergence of Relative Frequencies of Events to their Probabilities", *Theory of Probability and its Applications*, vol16, No. 2, pp264-280.