Special issue for Conf. on Struct. in Compl. Th. 1988

Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
Vol. 41, No. 3, December 1990
Printed in Belgium

10

Prediction-Preserving Reducibility

LEONARD PITT*

Department of Computer Science,
University of Illinois, Urbana, Illinois 61801

AND

MANFRED K. WARMUTH†

Department of Computer and Information Sciences,
University of California, Santa Cruz, California 95064

Received November 22, 1988; revised November 1, 1989

We investigate a model of polynomial-time concept prediction which is a relaxation of the distribution-independent model of concept learning due to Valiant. *Prediction-preserving reductions* are defined and are shown to be effective tools for comparing the relative difficulty of solving various prediction problems. A number of prediction-preserving reductions are given. For example, if deterministic finite automata are polynomially predictable, then so are all Boolean formulas. We develop a complexity theory for prediction problems that parallels standard complexity theory. It is shown that certain problems of concept prediction are "prediction-complete" for a complexity class—a polynomial time algorithm for the prediction problem would imply that all languages in the complexity class are polynomially predictable. For example, polynomial-time prediction of deterministic finite automata implies the polynomial predictability of all languages in the class LOG (deterministic logspace). Similar natural prediction-complete problems are given for the standard complexity classes NC^1, NLOG, LOGCFL, and P. Showing that a prediction problem is prediction-complete for any of these classes provides varying degrees of evidence that no efficient prediction algorithm exists for the problem. Based on very weak cryptographic assumptions, we establish hardness results for prediction of Boolean circuits and other prediction problems that are prediction-complete for P. The recent related resuts of Kearns and Valiant are discussed, which show that Boolean formulas and DFAs are not polynomially predictable based on the assumed intractability of computing specific cryptographic functions.

1. INTRODUCTION

In this paper we are concerned with the learning of *concepts* from examples. Imagine a domain X of possible real world observations. Intuitively, a concept c is

* Supported in part by NSF Grant IRI-8809570 and by the Department of Computer Science, University of Illinois at Urbana-Champaign.

† Supported in part by ONR Grant N00014-86-K-0454. Part of this work was done while visiting the Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138, with partial support from ONR Grant N00014-85-K-0554.

0022-0000/90 $3.00

simply a partition of the observations into *positive examples* and *negative examples*. A learning algorithm is given randomly generated positive and negative examples of some unknown *target* concept c to be learned and must produce a concept c' (a *hypothesis*), such that it is unlikely that c and c' will disagree on the classification of a new randomly generated example.

This extensional definition of a concept is not particularly useful, and in practice, it is desirable to assume that each concept c has associated with it a description r in some given *representation* language. Given any class of possible representations R, there is an associated *concept class* C, consisting of concepts described by elements of R. For example, Boolean formulas and DFAs (deterministic finite automata) are classes of representations that induce the concept classes of Boolean functions and regular languages, respectively. Henceforth we interchangeably use R to refer to a class of representations, as well as the concept class that it induces, and similarly, $r \in R$ will denote either the representation or the concept represented. As we discuss below, the representational power associated with a given concept description language R is a significant factor in determining whether R may be efficiently learned.

Distribution Independent Learning

The definition of distribution independent learnability (called *pac-learnability* [6]) of Valiant [53] addresses these representational issues, among others. A concept class R over domain X is pac-learnable if there exists a polynomial-time algorithm A such that for any $r \in R$, if A is given randomly generated elements of X (chosen according to an arbitrary, unknown distribution D on X) and told which are positive and which are negative examples of (the concept described by) r, then in polynomial time, A will output, with high probability (at least $1-\delta$) a concept (description) $r' \in R$ for which the concept r' approximates the target concept r in the following sense: the probability that an example generated according to D will be classified differently by r and r' is at most ε.

The values δ and ε are given as parameters to the algorithm A, reflecting the desired degrees of *confidence* $(1-\delta)$ in the performance of A, and the *accuracy* $(1-\varepsilon)$ of the hypothesis r' produced by A. The run time of A, and hence the number of examples seen, may grow at most polynomially in $1/\varepsilon$, $1/\delta$, as well as in other natural parameters reflecting the complexity of the learning task (for example, allowing the time to depend on the length of the description r of the target concept is typical). The algorithm is required to work regardless of the distribution D on the domain X.

A main goal in computational learning theory is to determine, for various definitions of successful learning, and in particular for pac-learnability, those concept classes that may be (efficiently) learned from examples. While a number of learnability results have been given (the papers [7, 11, 28, 37, 38] provide partial surveys), the learnability or nonlearnability of many natural classes remains undetermined. For example, the learnability of concept classes defined by Boolean formulas, restricted types of Boolean formulas (e.g., formulas in disjunctive normal

form (DNF)), DFAs,[1] NFAs, PDAs, CFGs, and Turing machines, remains unresolved.

In the search for learnable classes, a variant of pac-learning has been introduced: allow the algorithm A to output a description of the target concept, chosen from a *different* description language. Thus, following [45], we say that (the class of representations) R is *pac-learnable in terms of* (the class of representations) R', if the definition of pac-learnability above holds, but the concept description output by A is an element of R'. The most general relaxation possible is to that of (polynomial-time) *prediction*, where A is not required to output a description at all, but must arrive at a state in which it can predict (classify) future examples accurately (i.e., with error at most ε) with respect to the target concept. This notion of polynomial predictability, formally defined in Section 2, was introduced in [30, 31]. The polynomial predictability of R is equivalent to the existence of *any* class R' and algorithm A such that A pac-learns R in terms of R' [30].

A useful tool for studying pac-learnability (and predictability) is the Vapnik–Chervonenkis (VC) dimension [11, 32, 54] of a concept class. The VC dimension is a combinatorial parameter of a concept class with the following property. Let R be a concept class, and let R_s consist of all elements of R of length s. If the VC dimension of R_s grows more than polynomially in s, then R is not polynomially predictable (and hence not pac-learnable) [11, 19, 31]. In contrast, if the VC dimension of R_s grows polynomially in s (which is the case for all classes considered in this paper), a number of techniques have been given for finding learning and prediction algorithms with good performance, which cannot necessarily be implemented efficiently. Such techniques include: finding "small" consistent hypotheses [11, 12]; finding a prediction algorithm with small *permutation index* [31]; or finding a prediction algorithm with a polynomial worst case mistake bound [43]. Unfortunately, for the problems we consider in this paper, efficient implementations of these techniques have not been found.

Previous (Partially) Negative Results

In the absence of positive learnability or predictability results, we hope to show nonlearnability or nonpredictability. For a class R, let the *consistency problem* for R be the problem of determining, given a collection of positive and negative examples, whether there exists an $r \in R$ that is *consistent* with the examples, i.e., classifies the examples correctly. As shown in [45] (see also [11, 30]), if the consistency problem for R is NP-hard, then assuming that RP (random polynomial time) is not equal to NP, R is not pac-learnable.

For example, it has been shown that for each constant $k \geq 2$, the class of k-term DNF formulas is not pac-learnable (assuming $\mathsf{RP} \neq \mathsf{NP}$) [45]. This is true because for each $k \geq 2$, it is NP-hard to determine whether there exists a k-term DNF expression consistent with given examples. While this shows that k-term DNF is

[1] Angluin gives a learning algorithm for DFAs that relies on the ability to make queries as to the membership of examples chosen by the algorithm [3].

not learnable, it *does not* show that k-term DNF is not learnable in terms of some other representation. In fact, as was pointed out in [45], k-term DNF is pac-learnable in terms of k-CNF (Boolean formulas in conjunctive normal form with at most k literals in each clause). Similarly, it may be possible that k-term DNF is pac-learnable in terms of $f(k)$-term DNF for some polynomial f, and it would follow that DNF is pac-learnable [12]. Consequently, this type of nonlearnability result relies on the syntactic constraints imposed by the requirement that the hypothesis of the algorithm must be expressed in some particular representation (e.g., k-term DNF, for some particular k). If these constraints are relaxed (e.g., to k-CNF), then there may be pac-learning algorithms (e.g., k-term DNF is pac-learnable in terms of k-CNF).

Similarly, it has been shown [45] that Boolean threshold functions[2] are not pac-learnable unless RP = NP. However, it is easy to learn Boolean threshold functions in terms of half spaces by using linear programming [11], or by using the algorithm Winnow of [43]. Haussler [26] gives similar nonlearnability results that rely on representational constraints.

The research reported in this paper was motivated by attempting to determine the complexity of learning or predicting concept classes defined by DFAs and other types of automata. (See [44] for a survey on DFA learning.) By work of Blumer, Ehrenfeucht, Haussler, and Warmuth [12], and Board and Pitt [13], the pac-learnability of DFAs is equivalent to the existence of a randomized *Occam algorithm* for the *minimum consistent DFA problem*. Such an Occam algorithm takes as input any collection of strings (labeled "accept" or "reject") and produces (with high probability) a DFA that is consistent with the labeled strings and whose size is bounded by the product of (1) a polynomial in the size of the smallest consistent DFA and (2) a fractional power of the number of strings in the sample. Thus the pac-learnability of DFAs hinges on showing that the minimum consistent DFA problem can be very weakly approximated in random polynomial time.

Since the minimum consistent DFA problem is intimately related to the pac-learnability of DFAs, this problem has received significant study. Gold [21] shows that it is NP-hard to find the smallest DFA consistent with a given sample. Angluin [5] shows that this is true even if all but ε of the words up to a given length are given as examples. Li and Vazirani extend the NP-hardness result of [21] by showing that it is NP-hard to produce a consistent NFA that is at most $\frac{9}{8}$ times larger than the smallest consistent DFA [42]. In [46], we show (assuming $P \neq NP$) that there is no polynomial-time approximation algorithm that is guaranteed to produce a consistent NFA of size bounded above by *any* polynomial in the size of the smallest DFA.

Although these results show that finding small DFAs consistent with a given sample is difficult, even the last result mentioned does not rule out the possibility of the existence of an Occam algorithm for DFAs, since such an algorithm would

[2] Such a function is given by a clause and a threshold k. The function is true for all assignments that make at least k literals in the clause true.

be allowed to produce a consistent DFA whose size depended not only polynomially in the smallest consistent one, but also could be as large as a fractional power of the number of examples in the input sample. The problem of extending the results of [46] to show that no Occam algorithm exists seems very difficult. Furthermore, even if such a result were obtained, showing that DFAs were not pac-learnable, it would still be possible that DFAs were pac-learnable in terms of some other class of representations (not necessarily of the regular sets), and hence DFAs would be polynomially predictable.

Thus, we have not been able to show that a number of concept classes are learnable, and, on the other hand, the partial nonlearnability results for these problems rely on syntactic constraints that reflect the complexity of the consistency problem associated with a given choice of representations.[3] As discussed above, polynomial predictability captures learnability in terms of an arbitrary hypothesis class [30]. Thus, negative results for predictability are more meaningful; they reflect the complexity of noticing patterns in the data, as opposed to simply reflecting the syntactic difficulties of expressing such patterns. Negative results for prediction imply negative results for learning.

A Complexity Theoretic Approach

The theory of complexity classes and reducibilities (e.g., NP-completeness) has been particularly useful in providing evidence for the intractability of computational problems. Here we develop a similar complexity theory for predictability. We give formal definitions for prediction problems and introduce a notion of polynomial-time prediction-preserving reducibility among prediction problems.

Intuitively, a polynomial-time prediction-preserving reduction consists of two mappings: a polynomial-time computable function f that maps unlabeled examples of the first problem to unlabeled examples of the second problem and a function g that maps representations of the first problem to representations of the second problem. An interesting feature of our definition of reduction is that the mapping g need not be computable. We only require that g be length preserving within a polynomial.

Our definition of reduction is similar to the ones given in [37, 43], except that we allow a variety of domains and we determine general sufficient conditions that ensure the preservation of polynomial-time predictability. Littlestone [43] gives reductions between prediction problems in Boolean domains for a different model of predictability discussed in Section 8. Kearns *et al.* [37] give reductions between various pac-learning problems in the Boolean domain, and Haussler [26] gives reductions between pac-learning problems in structural domains.

We review reductions from previous work that have been given in a less formal

[3] In a more demanding, deterministic model of learnability, Angluin has shown that DFAs are not learnable by polynomially many *equivalence queries* [2], but are learnable by polynomially many equivalence and *membership* queries [3]. Neither of these results has any implications in the model considered here, where examples are generated according to an arbitrary probability distribution.

setting, and present as examples a number of new reductions. One such reduction shows that predicting arbitrary Boolean formulas is no easier even if the prediction algorithm is given the description of a tree circuit that computes the formula, with all gates specified, except the mapping of variables to inputs of the circuit is unknown.

With each prediction problem, we associate an *evaluation problem*: given a string and a representation, is the string a positive or a negative example of the concept denoted by the representation? We classify prediction problems by the complexity of their evaluation problems. This gives rise to a notion of *prediction-completeness*; if a prediction problem is prediction-complete for a given complexity class, then its predictability implies the predictability of all prediction problems whose evaluation problem is in that complexity class.

Using these ideas, we give a prediction-preserving reduction from an arbitrary prediction problem whose evaluation problem is in deterministic logspace (LOG) to the prediction problem for DFAs. This shows that the prediction problem for DFAs is prediction-complete for LOG. Since determining whether an assignment satisfies a given Boolean formula is in LOG, it follows that predicting arbitrary Boolean formulas reduces to predicting DFAs.

By modifying the proof for DFAs, we show that the prediction problems for NFAs, CFGs, and alternating DFAs are prediction-complete for the complexity classes NLOG, LOGCFL, and P, respectively, and we show that prediction of Boolean formulas is prediction-complete for the complexity class NC^1.

Problems Prediction-Complete for Polynomial Time—Hardness Results

In addition to the alternating DFA prediction problem, we describe a number of other prediction problems that, if predictable, would imply the predictability of all languages accepted in polynomial time. Our list of such hard prediction problems includes

- *Convex polytope intersection.* The concept is an unknown convex polytope; positive examples are cubes that have non-empty intersection with the polytope.
- *Horn clause consistency.* The concept is an unknown conjunction of Horn clauses; positive examples are sets of facts that are consistent with the conjunction.
- *Augmented CFG emptiness.* The concept is an unknown context free grammar; positive examples are sets of productions that when added to the grammar, yield a grammar generating the empty language.

It follows from the work of Goldreich, Goldwasser, and Micali [22] that these prediction problems are not predictable (even in an extremely weak sense) assuming the existence of cryptographically secure pseudorandom bit generators, which is equivalent to the existence of a certain type of one-way function [41].

Subsequent to this research, Kearns and Valiant [39] show that a polynomial-time learning or prediction algorithm for DFAs can be used to invert certain cryptographic functions. This is done by first showing that predicting arbitrary

Boolean formulas is as hard as inverting the given cryptographic functions. Next, apply Corollary 5.7, which shows that prediction of Boolean formulas reduces to prediction of DFAs. Consequently, DFAs are not polynomially predictable based on the same cryptographic assumptions.

The rest of this paper is organized as follows. In Section 2 we formally define polynomial-time predictability. Section 3 introduces the notion of a prediction-preserving reduction, and gives a number of examples. In Section 4 we examine additional properties of prediction-preserving reductions. We associate a language (called the evaluation problem) with each prediction problem, and we define what is meant for a prediction problem to be prediction-complete for a standard complexity class. Section 5 gives some of our main results: we relate automata prediction to various complexity classes, and in particular, we show that DFAs are prediction-complete for LOG. In Section 6 we describe some prediction problems that are as hard to predict as any language in polynomial time and describe in Section 7 why it follows that these problems are not predictable based on certain cryptographic assumptions. In Section 7 we also discuss the results of [39] showing that DFAs are not predictable based on different cryptographic assumptions. Finally, in Section 8 we summarize our results and discuss a number of interesting open problems.

2. Polynomial Predictability

Formal definitions for the complexity classes discussed in this paper may be found in [58], and in the references given below. LOG, NLOG, P, and NP denote the classes of languages accepted in deterministic logspace, nondeterministic logspace, polynomial time, and nondeterministic polynomial time, respectively. LOGCFL denotes the class of languages accepted by polynomial time bounded auxiliary (nondeterministic) PDAs with logarithmic additional work tape [52]. NC^1 denotes the class of U_{E^*}-uniform bounded fan-in circuit families of logarithmic depth (and hence polynomial size) [16, 48]. Ruzzo [48] showed that this is exactly the class of languages accepted by some *alternating* Turing machine in logarithmic time. (See [15, 48, 58] for definitions.)

Throughout the paper, let Σ and Γ be fixed, finite alphabets.

Definition 2.1. A *concept* c is any subset of Σ^*. A *concept class* C is any collection of concepts.

We are interested in characterizing those concept classes that are "polynomially predictable," i.e., concept classes C for which there exists a polynomial-time algorithm that, given polynomially many randomly generated elements of Σ^*, and told for each word whether or not the word is in some unknown concept $c \in C$, can predict with high probability whether a new (unseen) word is in c.

There are many issues involved in defining the predictability of a concept class.

A main one is that it is desirable to allow a prediction algorithm to receive more training examples (and to spend more time) before achieving accurate prediction, depending on the "complexity" of the concept to be predicted. A reasonable measure of this complexity is the length (number of letters) of the description of the concept in some given representation. Thus the predictability will depend on what type of representation of the concept we have chosen. For example, we may choose to represent regular languages by DFAs, NFAs, regular expressions, etc. We would like to ask the question "Are DFAs predictable?" rather than the question "Are regular languages predictable?" This motivates the following definition.

DEFINITION 2.2. A *prediction problem* is a pair (R, c), where $R \subseteq \Gamma^+$, and c is a mapping $c : R \to 2^{\Sigma^*}$. R is a "set of representations," and each $r \in R$ denotes the concept $c(r) \subseteq \Sigma^*$. The concept class represented by (R, c) is $\{c(r) : r \in R\}$.

Generally, given any R, the mapping c will be implicitly understood, hence we use R as an abbreviation for the prediction problem (R, c). We also use R to denote the concept class it represents.

In the following definitions of particular prediction problems, we use the word "encodes" to abbreviate "encodes with respect to some fixed, standard encoding scheme." As usual, if numbers are used to define the concepts, then we must explicitly mention whether they are to be encoded in unary or binary.

For ease of presentation, Σ was chosen as some fixed finite alphabet over which concepts are defined. Consequently, all formal language defined below are languages over Σ. As discussed in Section 8, our results hold without this restriction to a single alphabet. In any case, the restriction to a fixed alphabet is justified for the purposes of polynomial predictability (defined below), because the classes of languages considered are closed under homomorphisms.

- $R_{\mathrm{DFA}} = \{r : r \text{ encodes a DFA}\}$ is a set of representations (for the concept class of regular languages over the fixed alphabet Σ) with the implicit mapping c such that for any r, $c(r)$ is the concept (language) accepted by the DFA encoded by r.
- $R_{\mathrm{NFA}} = \{r : r \text{ encodes an NFA}\}$.
- $R_{\mathrm{PDA}} = \{r : r \text{ encodes a PDA}\}$.
- $R_{\mathrm{CFG}} = \{r : r \text{ encodes a CFG in Chomsky normal form}\}$.

Define a Boolean formula over Boolean variables $\{x_1, \ldots, x_n\}$ as a (not necessarily binary) rooted tree with internal nodes labeled with elements of $\{\wedge, \vee, \neg\}$, leaves labeled with elements of $\{x_1, \ldots, x_n\}$, and such that internal nodes labeled with $\vee$ or $\wedge$ have at least two children, and internal nodes labeled with $\neg$ have exactly one child. The value of a Boolean formula is defined in the usual way as the value of the circuit it represents, with the leaves as inputs, the root as output, and the interior nodes labeled $\wedge$, $\vee$, and $\neg$ interpreted as Boolean gates for logical AND, OR, and NOT, respectively.

• $R_{BF} = \{r : r$ encodes a binary number n and a Boolean formula over variables $\{x_1, ..., x_n\}\}$. In this case, given a formula r of n variables, the concept $c(r)$ is exactly those words of length n that, when interpreted as an assignment to the n variables, satisfy the formula encoded by r.

• $R_{DNF} = \{r : r$ encodes a number n in binary and a Boolean formula in disjunctive normal form over variables $\{x_1, ..., x_n\}\}$.

• $R_{CNF} = \{r : r$ encodes a number n in binary and a Boolean formula in conjunctive normal form over variables $\{x_1, ..., x_n\}\}$.

• $R_{CONVEX} = \{r : r$ encodes a number d in binary, and a system of linear inequalities over d variables (coefficients are integers encoded in binary) that defines a convex polytope$\}$. The concept $c(r)$ consists of all d-dimensional vectors that are solutions to the system of equations represented by r. The components of a vector are encoded in binary.

Before formally defining polynomial predictability, we must consider the amount of resources (time and number of examples) that should be available to the prediction algorithm. It is natural to allow the resources to grow polynomially in the inverse of the parameter ε, an upper bound on the desired predictive error of the prediction algorithm. As discussed above, the length (number of letters[4]) of the representation r of the unknown target concept $c(r)$ is a measure of its complexity; consequently, we will allow the resources to grow polynomially in a parameter s, which is an upper bound on $|r|$.

Finally, the resources of the prediction algorithm should be allowed to grow with the length of the input examples. For example, for prediction of Boolean formulas, the time (and number of examples) allowed will depend on n, the number of variables over which the formula is defined. However, for some concept classes, the words of a concept may not all have the same length (e.g., the concepts (languages) defined by DFAs, CFGs, etc.). An arbitrary probability distribution may supply words of significantly different lengths as examples. Because the words are chosen randomly, there are subtle issues involved in specifying in a natural way how the resources of the algorithm may depend on the word lengths.

We illustrate this difficulty by considering Angluin's approach regarding the same issue in a deterministic setting [2, 3]. In this model, the algorithm is allowed, at any point, time polynomial in the longest word that it has received up to that point. Suppose we adopt this definition in our stochastic setting, and consider a prediction algorithm that is allowed time (and hence a number of examples) at most quadratic in the length of the longest example yet seen. More specifically, a prediction algorithm using a quadratic number of examples may continue to run providing that the number of examples requested is less than the square of the length of the longest example yet seen, but must halt as soon as these values become equal.

We construct a distribution for which such an algorithm may never halt, although it is quadratically resource bounded. Let the distribution be such that for each $i \geq 1$,

[4] Other measures of length are discussed in Section 8.