Towards Representation Independence in PAC Learning

Manfred K. Warmuth*
Department of Computer and
Information Sciences
University of California
Santa Cruz, California 95064

Abstract

In the recent development of various models of learning inspired by the PAC learning model (introduced by Valiant) there has been a trend towards models which are as representation independent as possible. We review this development and discuss the advantages of representation independence. Motivated by the research in learning, we propose a framework for studying the combinatorial properties of representations.

1 Introduction

In Complexity Theory one always tries to find definitions that are independent of the particular representational details. We overview recent research in Computational Learning Theory in view of the objective of obtaining representation independent notions of learnability. We restrict our discussion to one family of learning models (called PAC learning models in [3]) whose original version was introduced by Valiant [39] (see Section 2 for definitions). In the PAC learning models the learning algorithm is given only a polynomial amount of resources (such as examples and running time) to achieve the learning task.

Usually there are two representations required for specifying a learning problem. The first one is the representation of the hidden concept that is to be learned and the second one the hypothesis the learning algorithm produces. Restricting the hypothesis of the algorithm be of a certain form frequently leads to infeasibility results that are very dependent on the particular restrictions used [32] (reviewed in Section 3). This has motivated the introduction of a definition of learning (called prediction) in which the hypothesis is not required to be of any particular form [20] (Section 4). (See [4,5,6,8,30,34,35] for related models of prediction.) The definition of prediction from [20] naturally leads to a notion of reduction (reviewed in Section 5) between representation classes which preserves predictability with a polynomial amount of resources [33]. Various complete

*Supported in part by ONR grant N00014-86-K-0454.

problem with respect to this definition of reduction have been shown to be hard to predict modulo cryptographic assumptions [27,33] (Section 6). In Section 7 we discuss dropping restrictions on the representation of the hidden concept to be learned.

In cases where no reduction has been found from a prediction problem to another, we would like to be able to prove that no such reduction is possible. In Section 8 we introduce a methodology that leads to such results and to a study of the combinatorial properties of representations in their own right. We conclude with a discussion of the usefulness of this methodology and give a large number of open problems (Section 9).

2 The definition of PAC learning

In this section we describe the PAC learning model which was introduced by Valiant [39]. By now many variants of the original model have appeared in the literature ([20] gives a comparison of the most common variants). We define a particular variant below and motivate our choice.

Definition 2.1 *An* alphabet Σ *is any set. Elements of alphabets are called* symbols. *A* word w *over an alphabet* Σ *is any finite length sequence of symbols of* Σ *(*$|w|$ *denotes its* length*).* Σ^* *denotes all such words and* λ *stands for the empty sequence (word) which has length zero. A* concept *or* language c *over an alphabet* Σ *is any subset of* Σ^*. *A* concept class C *over an alphabet* Σ *is any set of concepts, i.e.* $C \subseteq 2^{\Sigma^*}$.

Throughout this paper we use Σ to denote the alphabet over which concepts are defined.

The goal of the PAC learning model is to characterize those concept classes that are "learnable" with a polynomial amount of resources. (The acronym PAC abbreviates "probably approximately correct" [3].) The purpose of this paper is to study representation independence in the PAC learning model. Observe that concepts correspond to $\{0,1\}$-valued functions. In some recent work [18] Haussler generalizes the PAC model to learning real-valued functions. Many interesting issues arise already in the case of learning concepts and we choose to restrict ourselves to that case.

Intuitively we say that a concept class C is "polynomially learnable" if there is a polynomial time (possibly randomized) algorithm that, given polynomially many randomly generated elements of Σ^*, and told for each word whether or not the word is in some unknown *target concept* $c \in C$, produces a "hypothesis" $h \in C$ that approximates c accurately with high probability.

We will formalize the above intuitive definition. First, we would like the amount of resources (such as the number of examples and running time) used by the learning algorithm to grow polynomially in the "complexity" of the unknown target concept $c \in C$. Concepts are possibly infinite languages and thus a reasonable measure of complexity is the length (number of letters) of the description (representation) of the concept in some given representation language. The question

of whether a concept class is polynomially learnable will depend on what representation language for the concepts we have chosen. A concept class might be polynomially learnable with respect to one representation but not with respect to another. For example, the question "Are regular languages polynomially learnable?" is not well specified. A suitable question would be "Are DFAs polynomially learnable?". Observe that learning NFAs should be harder than learning DFAs since there are regular languages whose NFA representation is exponentially more concise than its DFA representation. This discussion motivates the following definition.

Definition 2.2 *A* representation class *is a four-tuple* (R, Γ, c, Σ) *such that*

Γ *is an alphabet and* R *a language over* Γ, *called the* representation language.

Σ *is an alphabet for the concepts represented by* R *and* c *is a mapping from* R *to concepts over* Σ^*. *For each representation* $r \in R$, *the concept* $c(r) \subseteq \Sigma^*$ *is the concept represented by* r.

The concept class represented by (R, Γ, c, Σ) *is the set* $\{c(r) : r \in R\}$.

For $r \in R$, *the* complexity *of the representation* r *is* $max(\{1, |r|\})$.

Throughout this paper the letter Γ denotes the alphabet used for representations. Both the concept alphabet Σ and the representation alphabet Γ may be infinite. For example, if concepts are subsets of the Euclidean n-dimensional real space $\mathbb{R}^n$, then Σ and Γ might contain the set of all reals $\mathbb{R}$. However, note that in our definition of complexity of representations, each real in a word of Σ^* or Γ^* accounts for one unit. This corresponds to using the unit-cost model of computation [2]. For syntactical reasons we assume that there is a special symbol "\$" such that $\$ \notin \Sigma \cup \Gamma$ always holds.

In most cases, given any set of representation class (R, Γ, c, Σ), the alphabets Γ and Σ and the concept mapping c will be implicitly understood, hence we use R as an abbreviation for the whole class. represented by (R, Γ, c, Σ).

Associated with any representation class (R, Γ, c, Σ) is an *evaluation problem*, which is that of determining, given an arbitrary $r \in R$ and $w \in \Sigma^*$, whether or not $w \in c(r)$. This is defined formally as a language:

Definition 2.3 *The* evaluation problem *for a representation class* (R, Γ, c, Σ) *is the language* $\{r\$w : w \in c(r)\}$, *where* \$ *is the special symbol that is not in* $\Sigma \cup \Gamma$. *We denote the defined language as* $E(R, \Gamma, c, \Sigma)$ *and use* $E(R)$ *for shorthand.*

Throughout this paper we assume that for all representation classes considered the evaluation problem is in $\mathcal{P}$ (polynomial time).

In the following definitions of particular representation classes, we use the word "encodes" to abbreviate "encodes with respect to some fixed, standard encoding scheme." As usual, if integers

are used to define the concepts, then we must explicitly mention whether they are to be encoded in unary or binary.

Conventions: For all Boolean representation classes given below, the concept alphabet is $\{0,1\}$, and for simplicity we assume that each Boolean concept b is a subset of $\{0,1\}^n$ for some positive power n. Each representation r such that $c(r) = b$ is assumed to contain n in binary. The number n indicates the number of variables of the concept. These variables have indices in the range from one to n which are encoded in binary and a bit pattern $\{0,1\}^n$ encodes an assignment to the variables.

For representation classes where each concept is a language over some finite alphabet (such as DFAs, NFAs, CFGs) the concept alphabet Σ_∞ is the infinite "universal set of symbols" which is assumed to contain all finite alphabets (except for the symbol "$"). The representation of a language specifies the finite subalphabet of Σ_∞ used for the particular language.

Below we define the representation classes used in this paper.

- $R_{BF} = \{r : r$ encodes a Boolean formula$\}$. For a formula r of n variables the concept $c(r)$ consists of all assignments $\{0,1\}^n$ that satisfy the formula.
- $R_{DNF} = \{r : r$ encodes a Boolean formula in disjunctive normal form$\}$.
- $R_{CNF} = \{r : r$ encodes a Boolean formula in conjunctive normal form$\}$.
- $R_{k-termDNF} = \{r : r$ encodes a DNF formula with at most k terms$\}$.
- $R_{k-clauseCNF} = \{r : r$ encodes a CNF formula with at most k clauses$\}$.
- $R_{k-DNF} = \{r : r$ encodes a DNF formula where each term has at most k literals$\}$.
- $R_{k-CNF} = \{r : r$ encodes a CNF formula where each clause has at most k literals$\}$.
- $R_{monomials} = R_{1-termDNF} = R_{1-CNF}$ and $R_{clauses} = R_{1-clauseCNF} = R_{1-DNF}$.
- $R_{XORsum} = \{r : r$ encodes the exclusive OR of a set of literals$\}$.
- $R_{DT} = \{r : r$ encodes a Boolean decision tree$\}$. A Boolean decision tree is a binary tree, where each internal node is labeled with a variable and each leaf with either 1 or 0. The two branches at each internal node represent the two possible settings of the variable at that note. For any assignment of the variables there is a unique path to a leaf in the tree. The concept $c(r)$ consists of all assignments leading to leafs labeled with 1.
- $R_{CIRC} = \{r : r$ encodes an acyclic Boolean circuit$\}$, where if r has n inputs, then $c(r)$ is the set of assignment $\{0,1\}^n$ accepted by the circuit encoded by r.
- $R_{DFA} = \{r : r$ encodes a DFA$\}$ is a set of representations (for the concept class of regular languages) with the implicit mapping c such that for any r, $c(r)$ is the concept (language) accepted by the DFA encoded by r.

- $R_{NFA} = \{r : r \text{ encodes an NFA}\}$.
- $R_{CFG} = \{r : r \text{ encodes a CFG in Chomsky normal form}\}$.
- $R_{SINGLE} = \{r : r \text{ encodes one bit pattern}\}$. Let u be the bit pattern represented by r. Then $c(r) = \{0,1\}^{|u|} - \{u\}$.
- $R_{PAIRS} = \{r : r \text{ encodes two distinct bit patterns of the same length}\}$. Let u and v be the bit patterns represented by r. Then $c(r) = \{0,1\}^{|u|} - \{u, v\}$.

  We conclude this list of representation classes with a class for which the concept alphabet and the representation alphabet consist of the set of reals $\mathbb{R}$. The defined concepts are iso-thetic boxes in $\mathbb{R}^n$, i.e. each facet of such a box is parallel to a hyper-plane spanned by a subset of $n-1$ of the n coordinate axes.
- $R_{\Box} = \{r : r \text{ is a sequence (word) of an even number of reals }\}$. If $r = x_1 x_2, \cdots, x_n$, then the concept $c(r)$ is the following cross product of closed intervals of $\mathbb{R}$: $[x_1, x_2] \times [x_3, x_4] \times \cdots \times [x_{n-1}, x_n]$. Note that $c(r) \subseteq \mathbb{R}^{n/2}$.

A learning algorithm for a representation class is given words labeled according to an unknown target representation in the class, which are called *examples*:

Definition 2.4 *Let (R, Γ, c, Σ) be a representation class. For any representation $r \in R$, and for any word $w \in \Sigma^*$, let $label_{c(r)}(w) =$ "+" if $w \in c(r)$ and "−" if $w \notin c(r)$. An* example of *the concept $c(r)$ (respectively the representation r) is a pair $\langle w, label_{c(r)}(w)\rangle$. A sequence of examples is* consistent with *$c(r)$ (respectively r) if all examples in the sequence are examples of $c(r)$. An* unlabeled *example is just a word w.*

We have already mentioned that the resources of the learning algorithm is allowed to grow with the complexity of the representation from which it receives examples. Formally this is done by giving the learning algorithm a parameter s, which is an upper bound on the length of the target representation r according to which the examples are labeled. The number of examples is allowed to grow polynomially in this parameter s.

The example words given to the learning algorithm are drawn at random. However, when learning a representation class whose concepts may be infinite sets of words of unbounded length, then it is also reasonable to allow the number of examples required by the learning algorithm to grow polynomially with the length of the longest example word seen (here let the empty word have length one). Note that if the concept alphabet contains the set of reals $\mathbb{R}$, then a point in $\mathbb{R}^n$ may be encoded by a word of n symbols of $\mathbb{R}$ and thus not every infinite concept contains words of unbounded length. However, the regular language a^*b^* over the alphabet $\{a, b\}$ is a set of words of unbounded length.

Since the example words are drawn at random it is difficult to specify the dependence on the length of the longest example. One might let the number of examples grow polynomially in the expectation and the variance of the longest length, where these two parameter are explicit inputs to the learning algorithm. We choose a simpler convention which is motivated in more detail in [33]. We assume that the learning algorithm only receives example words up to a given maximum length n (the probability of all longer words is zero), and the algorithm is given n as a parameter.

Definition 2.5 *For any language L and $n \geq 1$, let $L^{[n]} = \{w \in L : |w| \leq n\}$.*

The examples words are drawn according to an arbitrary but fixed distribution on $\Sigma^{[n]}$ and the number of examples of the learning algorithm is allowed to grow polynomially in the input parameters n and s (the upper bound on the complexity of the representation described above). In the high level definition of a learning algorithm given at the beginning of this section we required that the hypothesis produced by the learning algorithm must be "accurate with high probability". This is made precise using the following definitions.

Definition 2.6 *Let (R, Γ, c, Σ) be a representation class. For two representation $r, r' \in R$, $r \triangle r'$ denotes the set $(c(r) - c(r')) \cup (c(r') - c(r)))$, i.e. the symmetric difference of the concepts defined by r and r'. Let n be a maximum word length and P be a probability distribution on $\Sigma^{[n]}$. Then the* error *of a hypothesis representation r' with respect to a target representation r is defined to be the probability $P(r \triangle r')$.*

Intuitively, the error of the hypothesis is the probability that it labels a random word differently than the target representation. The learning algorithm is given two additional parameters ϵ and δ, both in the interval $(0, 1)$, and is required to produce a hypothesis whose error is at most ϵ with probability at least $1 - \delta$. The number of examples is allowed to grow polynomially in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

Below we give a formal summary of the model discussed.

Definition 2.7 *A* learning algorithm *A is an algorithm that takes as input four parameters $s, n \in \mathbb{N}$, $\epsilon, \delta \in (0, 1)$ and a collection of examples of $\Sigma_1^{[n]} \times \{+, -\}$, for some concept alphabet Σ_1. A outputs a representation of a hypothesis and it is a* polynomial time learning algorithm *if there exists a polynomial t such that the run time of A is at most $t(s, n, \frac{1}{\epsilon}, \frac{1}{\delta}, l)$, where l is the total length of the input of A.*

Definition 2.8 *The representation class $(R_1, \Gamma_1, c_1, \Sigma_1)$ is* polynomially learnable in terms of *the representation class $(R_2, \Gamma_2, c_2, \Sigma_2)$ iff there exists a polynomial time learning algorithm A and a polynomial p such that for all input parameters $s, n \in \mathbb{N}$ and $\epsilon, \delta \in (0, 1)$, for all $r_1 \in R_1^{[s]}$, and for all probability distributions[1] on $\Sigma_1^{[n]}$, if A is given at least $p(s, n, \frac{1}{\epsilon}, \frac{1}{\delta})$ randomly generated examples*

[1]The allowed distributions must fulfill some relatively benign measure-theoretic assumptions discussed in the Appendix of [9].

of an unknown target concept*)* $c_1(r_1)$ *(for* $r_1 \in R_1$*), then A produces a hypothesis in* R_2 *which with probability at least* $1-\delta$ *has error at most* ϵ*. (The class* $(R_1, \Gamma_1, c_1, \Sigma_1)$ *is called the* target class *and* $(R_2, \Gamma_2, c_2, \Sigma_2)$ *the* hypothesis class.*)*

In the original definition of polynomial learnability the target class and the hypothesis class of the algorithms were required to be identical [39]. A number of polynomial learning algorithms have been found which show that a class R is learnable in terms of itself [1,9,12,23,26,37,39]. However, for many useful, known learning algorithms, the target class and the hypothesis class are not the same and the learning algorithm uses that fact [9,19,30,32]. Recall that in this paper we only consider representation classes whose evaluation problem is in $\mathcal{P}$.

3 Hardness results for learning with particular hypotheses classes

In this section we review the background leading to infeasibility results of the following type: A representation class R is not polynomially learnable in terms of the hypothesis class R_1 modulo the very likely complexity theoretic assumption that[2] $\mathcal{RP} \neq \mathcal{NP}$, but at the same time R is polynomially learnable in terms of a different hypothesis class R_2. Thus learnability very much depends on how the hypotheses are represented.

Definition 3.1 [20] *A* random polynomial time hypothesis finder (r-poly hy-fi) for R_1 in terms of R_2 *is a randomized polynomial time algorithm A that for all input parameters* $s, n \in \mathbb{N}$*, all representations* $r_1 \in R_1^{[s]}$*, and all sets of examples of* $c_1(r) \cap \Sigma_1^{[n]}$ *produces with probability at least* γ *(for some fixed* γ*) a hypothesis representation in* R_2 *that is consistent with the examples.*

Theorem 3.2 [20,32] *If* R_1 *is polynomially learnable in terms of* R_2*, then there exists an r-poly hi-fi for* R_1 *in terms of* R_2*.*

Proof sketch: Let A be a polynomial learning algorithm for R in terms of R'. We construct an r-poly hi-fi B from the learning algorithm A. B receives the input parameters s, n, and an example sequence U. B poses a particular learning problem to A which uses the same parameters s and n. All examples not in U have probability zero and the examples in U all have equal probability. The parameter δ (which corresponds to γ) is set to $\frac{1}{2}$ and ϵ is set ot $\frac{1}{q+1}$, where q is the number of distinct examples in U. Note that ϵ is smaller than the probability of any example in U. It is easy to see that when A is applied to this learning problem then it must produce a consistent hypothesis (since ϵ is small) with probability at least $\frac{1}{2}$. □

[2] $\mathcal{RP}$ consists of all languages accepted by probabilistic polynomial time Turing machines with one-sided error bounded away from 0, as defined in [15].

An example (from [32]) of how this lemma can be applied is the following: for each constant $k \geq 2$, it is NP-complete to decide for a given sequence of examples whether there is a consistent k-term DNF formula. Thus if the class $R_{k-termDNF}$ was learnable by the same class of representations then by the above lemma there would exists a randomized polynomial algorithm for an NP-complete problem and thus $\mathcal{RP} = \mathcal{NP}$. However, $R_{k-termDNF}$ can be learned in terms of R_{k-CNF} [32].

Similarly, it can be shown that the Boolean threshold functions[3] cannot be learned by the same class unless $\mathcal{RP} = \mathcal{NP}$ [32]. Again it is easy to learn Boolean threshold functions in terms of halfspaces using linear programming [9]. (The algorithm Winnow of [30] can also be used to learn Boolean threshold functions in terms of halfspaces.)

Other infeasibility results for learning with particular hypothesis classes which are based on the assumption that $\mathcal{RP} \neq \mathcal{NP}$ are given in [1,7,19,29].

4 Learning independent of the representation of the hypothesis

Motivated by the infeasibility results discussed in the previous section, a notion of learnability (called predictability) is defined in [20] that is independent of the representation of the hypothesis.

Definition 4.1 *A representation class R_1 is* polynomially predictable *if there exists a representation class R_2 (whose evaluation problem is in P) and R_1 is polynomially learnable in terms of R_2.*

From now on we use the following definition of polynomially predictable (which is shown to be equivalent in [20]). In this definition, the prediction algorithm is not required to output a hypothesis.

Definition 4.2 *A* prediction algorithm *A is an algorithm that takes as input three parameters $s, n \in \mathbb{N}$, epsilon $\in (0,1)$, a collection of elements of $\Sigma^{[n]} \times \{+,-\}$ (for some concept alphabet Σ), and an element $w \in \Sigma^{[n]}$. The output of A is either "+" or "−", indicating its prediction for w. A is a* polynomial time prediction algorithm *if there exists a polynomial t such that the run time of A is at most $t(s, n, \frac{1}{\epsilon}, l)$, where l is the total length of the input of A.*

Definition 4.3 *The representation class (R, Γ, c, Σ) is* polynomially predictable *iff there exists a polynomial time prediction algorithm A and polynomial p such that for all input parameters s, n, and $\epsilon > 0$, for all $r \in R^{[s]}$, and for all probability distributions on $\Sigma^{[n]}$, if A is given at least $p(s, n, \frac{1}{\epsilon})$ randomly generated examples of (the unknown* target concept*) $c(r)$, and a randomly generated unlabeled word $w \in \Sigma^{[n]}$, then the probability that A incorrectly predicts $label_{c(r)}(w)$ is at most ϵ.*

[3] 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.

Note that in all definitions of learnability and predictability given in this paper the parameters s, n, $\frac{1}{\epsilon}$, and in the case of learnability the parameter $\frac{1}{\delta}$ as well, are explicitly given as inputs to the algorithms. However, the notions of polynomial learnability and predictability are not weakened if these parameters are not inputs of the algorithm but the number of examples and the running time is still allowed to depend polynomially on the parameters. (See [20,33] for a more formal treatment.)

The above notion of prediction has various advantages over learnability. First there is a natural notion of reductions among representation problems that preserve polynomial predictability [33]. (See next section.) The reductions lead to complete problems for sets of representation classes and for some of these complete problems infeasibility results have been shown [27,33]. (These results are reviewed in Section 6.) Recall that if a representation class is not polynomially predictable then this implies that it is not polynomially learnable *in terms of any representation class* whose evaluation problem is in $\mathcal{P}$. Thus infeasibility results for polynomial predictability are very strong and useful.

5 Prediction preserving reductions

Motivated by the reductions given in [30], Pitt and Warmuth [33] introduced a general notion of reduction among representation classes that preserves polynomial predictability. Thus if there is a reduction from R_1 to R_2, then the existence of a polynomial prediction algorithm A_2 for R_2 implies the existence of a polynomial prediction algorithm A_1 for R_1. The algorithm A_1 is constructible from A_2 and from the reduction R_1 to R_2.

We would like to motivate the definition of reduction by appealing to our intuition of how one can predict R_1 given a prediction algorithm for R_2. When A_1 is given examples of R_1 then it should be able to transform these examples to examples of R_2. Thus our definition of a reduction provides a function f (called the *word transformation*) which is employed as follows: when A_1 receives the example $(x, label)$ then it forwards $(f(x), label)$ to A_2. After receiving a number of labeled examples, A_1 is given an unlabeled word w. Again A_1 passes the corresponding word $f(w)$ to A_2. Hopefully A_1 will be able to simply use the prediction of A_2 on $f(w)$. For this to work there must exist a second transformation g (called the *representation transformation*) which maps any representation $r_1 \in R_1$ to a related representation $g(r_1) \in R_2$ with the property that for all words $w_1 \in \Sigma_1$

$$w_1 \in c_1(r_1) \text{ iff } f(w_1) \in c_2(g(r_1)),$$

where c_i is the implicit concept mapping of R_i. We outlined how a prediction algorithm A_2 for R_2 might be used to obtain a prediction algorithm A_1 for R_1. However, the constructed algorithm A_1 is a *polynomial* prediction algorithm only if certain requirements are met for the

two transformations f and g. The word transformation clearly must be computable in polynomial time. For the representation transformation much weaker conditions apply. We only need that g is length preserving within a polynomial. The representation transformation g does not have to be computable at all.

Below we give the formal definition of reduction. Since the running time of a prediction algorithm is allowed to grow polynomially in s and n, we allow f to grow polynomially in these parameters as well. There are many reductions [33] where the word transformation makes use of the parameters s and n. Similarly we let g also depend on the additional parameter n. (It is not necessary to provide g with the parameter s, which is a bound on $|r_1|$, since r_1 is already a parameter to g.) Note that we did not let f and g grow polynomially in $\frac{1}{\epsilon}$. The reason for this is that so far we have found no reductions that would make use of such a dependence.

Definition 5.1 *A function h all of whose inputs and outputs are either words or natural numbers (encoded in unary notation) is called* polynomially length preserving *if there is a fixed polynomial q such that all inputs to the function h of length at most l produce outputs of length at most $q(l)$.*

Definition 5.2 *Let $(R_1, \Gamma_1, c_1, \Sigma_1)$ and $(R_2, \Gamma_2, c_2, \Sigma_2)$ be a pair of prediction classes. Then with respect to this pair, a* word transformation *is a function $f : \Sigma_1^* \times \mathbb{N} \times \mathbb{N} \to \Sigma_2^*$ and a* representation transformation *a function $g : R_1 \times \mathbb{N} \to R_2$.*

Definition 5.3 *For two representation classes $(R_1, \Gamma_1, c_1, \Sigma_1)$ and $(R_2, \Gamma_2, c_2, \Sigma_2)$ the first one* reduces to *the second one (denoted by $(R_1, \Gamma_1, c_1, \Sigma_1) \trianglelefteq (R_2, \Gamma_2, c_2, \Sigma_2)$) iff there is a polynomially length preserving word transformation f and a polynomially length preserving representation transformation g such that for all $s, n \in \mathbb{N}$, $r_1 \in R_1^{[s]}$, and $w_1 \in \Sigma_1^{[n]}$,*

(1) *$w_1 \in c_1(r_1)$ iff $f(w_1, s, n) \in c_2(g(r_1, n))$;*

(2) *f is computable in time $t(|w_1|, s, n)$, for some fixed polynomial t.*

In [33] it was shown that the above notion of reducibility fulfills the basic requirements (below two lemmas) that make it useful.

Lemma 5.4 [33] *For all representation classes R_1 and R_2, if $R_1 \trianglelefteq R_2$ and R_2 is predictable, then R_1 is predictable.*

Lemma 5.5 [33] *The relation $\trianglelefteq$ is transitive, i.e., if $R_1 \trianglelefteq R_2 \trianglelefteq R_3$, then $R_1 \trianglelefteq R_3$.*

Definition 5.6 *Let $R_1 \not\trianglelefteq R_2$ denote the fact that $R_1 \trianglelefteq R_2$ does not hold.*

5.1 Example reductions

Every k-term DNF formula over n variables can be rewritten as a k-CNF formula over the same n variables of size $O(n^k)$. For example, $xy + rs = (x + r)(x + s)(y + r)(y + s)$ (here $k = 2$). Thus for each k, $R_{k-termDNF} \trianglelefteq R_{k-CNF}$ [32]: the word transformation f is the projection function of its first argument (i.e., f is the identity on w_1), and the representation transformation g maps a k-term DNF expression to the equivalent k-CNF expression.

We next give a reduction from [25,30]: for any fixed k, $R_{k-CNF} \trianglelefteq R_{monomials}$. (Thus by transitivity, for any fixed k, $R_{k-DNF} \trianglelefteq R_{monomials}$.) Observe that in a k-CNF formula there are at most $u = O(n^k)$ clauses, since each clause has at most k literals. Let f map each n-bit assignment into a u-bit assignment, the ith bit of which is 1 iff the ith k-literal clause is 1. Given this transformation of the assignment, the mapping g simply expresses each k-CNF with v clauses as a monomial of size v over the enlarged variable set of size u. Since k is a constant, f is computable in time polynomial in n, and the size of the image of g is bounded by a polynomial in n.

One of the first representation classes shown to be polynomially predictable is $R_{monomials}$ [39] The above reductions can be used to show that $R_{k-termDNF}$ and R_{k-CNF} are polynomially predictable as well [25,30,32,39]. As discussed in Section 3, finding a consistent 2-term DNF expression for a given set of examples (each consisting of a truth assignment and a label) is NP-hard [32]. It is easy to find a consistent 2-CNF or a consistent monomial for a given set of examples, or to determine that there is no consistent one. So since $R_{2-termDNF} \trianglelefteq R_{2-CNF} \trianglelefteq R_{monomials}$, can't we use the theses reductions to solve an NP-complete problem? When given a set of examples that is consistent with a 2-term-DNF, apply the word transformation f of the reduction $R_{2-termDNF} \trianglelefteq R_{2-CNF}$ to the assignment and leave the labels unchanged. Now Requirement (1) implies that there is a R_{2-CNF} consistent with the new set of examples. Not surprisingly, the opposite direction is false. If there is a R_{2-CNF} consistent with the new set of examples then this does not imply the there is a consistent $R_{2-termDNF}$ for the original set of examples.

One of the two main open problems in computational learning theory are the questions whether arbitrary DNF (R_{DNF}) and arbitrary decision trees (R_{DT}) are polynomially predictable. (In [12] decision trees of fixed rank are shown to be learnable by the same class.)

It is easy to construct a logically equivalent DNF formula for a given decision tree: take the conjunction of all terms corresponding to the leaves labeled with 1. Thus $R_{DT} \trianglelefteq R_{DNF}$ (f is again the identity on w_1).

An obvious question is whether $R_{DNF} \trianglelefteq R_{DT}$. It can be shown [21] that the smallest logically equivalent decision tree for the formula $x_1x_{\frac{n}{2}+1} + x_2x_{\frac{n}{2}+2} + \cdots + x_{\frac{n}{2}}x_n$, for n even, must have at least 2^n negative leaves. This shows that if f is restricted to be the identity on w_1, then $R_{DNF} \not\trianglelefteq R_{DT}$. However, if the word transformation is allowed to be an arbitrary polynomial function, then there still might be a reduction from R_{DNF} to R_{DT}. In Section 8 we give techniques for proving results

of the form $R_1 \not\trianglelefteq R_2$.

As another example, observe that the language consisting of satisfying assignments (encoded as bit patterns of length n) of any DNF formula over n variables with s terms is accepted by an NFA of size $O(sn)$. (The NFA guesses which of s terms is satisfied, and branches to a chain of $O(n)$ states to verify that the n bit input satisfies the appropriate literals.) Thus predicting DNF trivially reduces to predicting NFAs, where again, f is the identity on w_1, and g maps the DNF expression to the corresponding NFA.

We now give a case where there is no reduction between between two representation classes when f is restricted to be the identity on w, but there is one when this restriction is dropped. Observe that the satisfying assignments of the DNF formula $x_1x_{\frac{n}{2}+1} + x_2x_{\frac{n}{2}+2} + \cdots + x_{\frac{n}{2}}x_n$, for n even, is a language whose smallest DFA has size at least $2^{\frac{n}{2}}$. Thus $R_{DNF} \trianglelefteq R_{DFA}$ does not hold if f is the identity on w. This is contrasted by the following reduction from R_{DNF} to R_{DFA}. We include the proof here since it is instructive.

Theorem 5.7 [33] $R_{DNF} \trianglelefteq R_{DFA}$.

Proof: Let $r_1 \in R_{DNF}$ encode a DNF expression of n variables. Then each example assignment is a word of length n. The parameter s is a bound on the length of the target DNF expression r_1. In particular, s is an upper bound on the number of terms of r_1. For all assignments w, $f(w, s, n) = (w_1)^s$, i.e. f simply replicates w_1 exactly s times. For a given DNF expression r_1 with at most s terms it is easy to design a DFA A with $O(sn)$ states such that r_1 is true on w_1 iff A accepts $(w_1)^s$. In particular, for each of at most s terms, the DFA uses a chain of $O(n)$ states (and input symbols) to check if the term is satisfied. If not, then it moves on to the next copy of w_1 in the input, and the next set of states to test whether the next term is satisfied. Thus g simply needs to map r_1 and n into a representation r_2 of such an automaton A. For any reasonable encoding for DFAs we have that $|r_2|$ is polynomial in sn. □

Reductions of the form $(R, \Gamma, c, \Sigma) \trianglelefteq (R', \Gamma, c, \Sigma)$ where $R' \subseteq R$ are particularly useful for determining the "hard core" of the representation language R. Below we give two of such reductions. The goal is to make R' as small and restricted as possible.

Definition 5.8 [33] $R_{BFtree} = \{r : r$ *encodes a permutation* π *of* $n = 2^k$ *elements, for some* $k\}$. *For any* k, *let* $T^{(k)}$ *be the complete ordered Boolean tree of height* k, *where the gates at even height are* $\vee$*-gates and the gates at odd height are* $\wedge$*-gates. Then* $c(r)$ *consists of exactly those bit patterns* w *of length* 2^k, *such that if the leaves of* $T^{(k)}$ *are labeled (in order) with the inputs* $\pi(w)$, *then* $T^{(k)}$ *evaluates to* 1.

Note that the representations of R_{BFtree} are very restricted Boolean formula and thus in some sense R_{BFtree} is a subset of R_{BF}.

Theorem 5.9 [33] $R_{BF} \trianglelefteq R_{BFtree}$.

By fixing the labels of the gates other than alternating level by level as in R_{BFtree} one can get a normal form for R_{DNF}.

Definition 5.10 $R_{DNFtree} = \{r : r$ *encodes a permutation* π *of* $n = 2^k$ *elements, for some* $k\}$. *For any* k, *let* $T^{(k)}$ *be the complete ordered Boolean tree of depth* k, *where the gates at depth less than* $k/2$ *are labeled AND, and the gates at depth at least* $k/2$ *are labeled OR. Then* $c(r)$ *consists of exactly those strings* w *of length* 2^k, *such that if the leaves of* $T^{(k)}$ *are labeled (in order) with the inputs* $\pi(w)$, *then* $T^{(k)}$ *evaluates to* 1.

Theorem 5.11 $R_{DNF} \trianglelefteq R_{DNFtree}$

Proof: Let D be an arbitrary DNF formula with n variables and at most k terms. We first show that there exists a complete ordered binary tree T that computes D and has the property that all gates of depth less that half of the whole depth of the tree compute $\vee$ and the remaining gates compute $\wedge$. Let $z = 1 + \lceil \log(2n + k) \rceil$ and $y = 2^z$. Our basic building blocks will be complete binary trees of depth z (with y leaves). Call an $\vee$-tree ($\wedge$-tree) a tree with the above specifications that has only $\vee$-gates ($\wedge$-gates) at the internal nodes. The final tree T consists of an $\vee$-tree where each of the y leaves of the tree are merged with the root of a different $\wedge$-tree. Let $T_{\wedge,i}$ be the $\wedge$-tree of the i-th leaf ($1 \leq i \leq y$). Let $k' \leq k$ be number of terms in D. We label the leaves with inputs such that the first k' $\wedge$-trees compute the k' terms of D and the remaining trees compute the constant zero. Note that this would show that T computes D.

Let D_j be the j-th ($1 \leq j \leq k'$) term of D. Label the first $|D_j|$ leaves of $T_{\wedge,j}$ with the literals of D_j and the remaining ones with one. For the remaining trees $T_{\wedge,l}$ such that $k' + 1 \leq l \leq y$, label one of its leaves with zero and the other leaves arbitrary. Observe that each $T_{\wedge,l}$ computes zero. This completes the description of T which computes D. Note that the number of nodes in T is polynomial in k and n.

We now use the above construction to reduce R_{DNF} to R_{BFtree}. For simplicity we assume that all inputs are of length $m = n$. Let $k = s$ and define y and z as above. The word transformation f maps an n-bit assignment w_1 into the assignment $(w_1\bar{w}_1)^s 1^{sy} 0^{y^2-2sn-sy}$. The transformation g maps the representation r_1 ($|r_1| \leq s$) of a DNF formula D into a representation r_2 of a complete binary B tree of depth $2z$ that computes D and whose leaves are labeled with a permutation of the following input sequence: $I = (\langle x_1, ..., x_n, \neg x_1, ..., \neg x_n \rangle)^s (\langle 1 \rangle)^{sy} (\langle 0 \rangle)^{y^2-2sn-sy}$. We use the above construction to show that such a tree exists.

Observe that since $|r_1| \leq s$ the formula D represented by r_1 has at most s terms. So we can apply the above construction with $k = s$ to produce the required tree B from D. The requirement that the leaves of B must be labeled with a permutation of I can be easily fulfilled

because of the following. There are s occurrences of each literal in I and thus there are enough literals for the leaves of the $\wedge$-trees that compute a term. Similarly the sy ones in I suffice to fill the remaining leaves of these $\wedge$-trees with ones. Also if needed there is at least one *zero* for each of the y $\wedge$-trees since the number of zeros in I is at least as large as y, the total number of $\wedge$-trees. □

6 Complete problems and infeasibility results for prediction

Definition 6.1 *If R is a representation class, and $\mathcal{R}$ is a set of representation classes, then R is* prediction-hard *for $\mathcal{R}$ iff for all representation classes $R' \in \mathcal{R}$, $R' \trianglelefteq R$. If $R \in \mathcal{R}$ also, then R is* prediction-complete *for $\mathcal{R}$.*

Thus if a representation class R is prediction-hard for a class $\mathcal{R}$, then the polynomial predictability of R implies the polynomial predictability of every representation class in $\mathcal{R}$.

In [33] Pitt and Warmuth propose to classify representation classes according to the complexity of their evaluation problem. Intuitively, the higher the complexity of the language $E(R)$ (Definition 2.3), the harder it is to predict a hidden target representation of R.

Definition 6.2 *For a complexity class $\mathcal{L}$, let $\mathcal{R}_{\mathcal{L}} = \{R : E(R) \in \mathcal{L}\}$.*

Besides $\mathcal{P}, \mathcal{NP}$ and $\mathcal{RP}$ the complexity classes used in this section are:

$\mathcal{LOG}$, the class of languages accepted by a deterministic log-space bounded Turing machine;

$\mathcal{NLOG}$, defined as above except deterministic is replaced by non-deterministic;

$\mathcal{NC}^1$, the class of languages accepted by log-depth circuits of standard fan-in two Boolean gates.

Note that for each circuit in $\mathcal{NC}^1$, there is an equivalent polynomially sized boolean formula [27] and for each boolean formula there is an equivalent polynomially sized circuit in $\mathcal{NC}^1$. Thus R_{BF} is prediction-complete for $\mathcal{NC}^1$.

Other sets of representation classes for which prediction-complete problems have been found [33] include $\mathcal{R}_{\mathcal{LOG}}$, $\mathcal{R}_{\mathcal{NLOG}}$ and $\mathcal{R}_{\mathcal{P}}$.

Theorem 6.3 *[33] R_{DFA}, R_{NFA}, and R_{CIRC} are prediction-complete for $\mathcal{R}_{\mathcal{LOG}}$, $\mathcal{R}_{\mathcal{NLOG}}$ and $\mathcal{R}_{\mathcal{P}}$, respectively, and R_{DFA} is prediction-hard for $\mathcal{R}_{\mathcal{NC}^1}$.*

The following infeasibility results were shown with respect to a weaker model of polynomial predictability which only requires that the algorithm predicts correctly for the unlabeled example with

probability slightly better than $\frac{1}{2}$. Clearly, predictability implies weak predictability. Surprisingly, Schapire shows that the converse also holds [38].

Definition 6.4 *The representation class* (R, Γ, c, Σ) *is* weakly polynomially predictable *iff there exists a polynomial time prediction algorithm A and polynomials p and q such that for all input parameters* $s, n \in \mathbb{N}$, *for all* $r \in R^{[s]}$, *and for all probability distributions on* $\Sigma^{[n]}$, *if A is given at least* $p(s,n)$ *randomly generated examples of (the unknown target concept)* $c(r)$, *and a randomly generated unlabeled word* $w \in \Sigma^{[n]}$, *then the probability that A incorrectly predicts* $label_{c(r)}(w)$ *is at most* $\frac{1}{2} - \frac{1}{q(s,n)}$.

Theorem 6.5 [38] *If a representation class is weakly polynomially predictable then it also is polynomially predictable.*

It can be shown that Lemma 5.5 still holds if predictable is replaced by weakly predictable and thus the above definitions of completeness are the same for both variants of polynomial predictability.

Theorem 6.6 [33] *If there exist a one-way functions that is hard on its iterates [17,28], then no prediction-complete problem for* $\mathcal{R}_{\mathcal{P}}$ *is weakly polynomially predictable.*

Thus in particular R_{CIRC} and all the other complete problems for $\mathcal{R}_{\mathcal{P}}$ given in [33] are not weakly predictable. The proof of the above theorem relies on the fact that the existence of a one-way functions that is *one-way on its iterates* [17,28] is equivalent to the existence of one-way permutations [41]. From one-way permutations [41] a cryptographically secure pseudorandom bit generator can be constructed and using a construction of Goldreich, Goldwasser, and Micali [16] it can be shown that R_{CIRC} and thus any prediction-complete problem for $\mathcal{R}_{\mathcal{P}}$ is not weakly polynomially predictable [33].

At this point a method for proving unpredictability results based on cryptographic assumptions becomes evident. If cryptographic functions that are hard to invert can be shown to have easy evaluation problems in some "lower complexity class" $\mathcal{L}$, then any representation class that is prediction-hard $\mathcal{R}_{\mathcal{L}}$ is not predictable.

Kearns and Valiant [27] have recently taken this approach, and have shown that, based certain cryptographic assumptions (the intractability of inverting the RSA cryptosystem, factoring Blum integers, or deciding quadratic residuosity), there are some presentation classes in $\mathcal{R}_{\mathcal{NC}^1}$ that are not weakly predictable. Consequently, any representation class that is prediction-hard $\mathcal{R}_{\mathcal{NC}^1}$ (see Theorem 6.3) is not weakly predictable, based on the same cryptographic assumptions.

Theorem 6.7 R_{BF} *or* R_{DFA} *are weakly polynomially predictable then there exists a randomized polynomial algorithm for inverting the RSA cryptosystem, for factoring Blum integers, and for deciding quadratic residuosity.*

7 Independence of the target representation

In Section 4 is was shown that there are advantages to letting the representation class of the hypothesis be any arbitrary class in $\mathcal{R}_{\mathcal{P}}$ (i.e. any representation class whose evaluation problem is in $\mathcal{P}$). Why not choose the same approach for the target representation class? A canonical representation class would be R_{CIRC} (Boolean circuits) (or any other representation class derived from a "universal" computational model). However, the corresponding prediction problem R_{CIRC} is already prediction-complete for $\mathcal{R}_{\mathcal{P}}$ (Theorem 6.3) and thus modulo cryptographic assumptions unpredictable (Theorem 6.6).

In practice, one would like to take advantage of the fact that the target representation is required to be of a particular form. This limits the space of possible target concepts of words in $\Sigma^{[n]}$ that have a representation in $R^{[s]}$ and simplifies the "search" of the prediction algorithm for the target concept. For a fixed concept $c \subseteq \Sigma^{[n]}$, restricting the representation language amounts to increasing the smallest size s such that there is a representation for the concept c in $R^{[s]}$. Note that s is a parameter of the prediction algorithm and the number of examples and the running time is allowed to grow polynomially with s. Thus increasing the size of the smallest representation s amounts to giving the prediction algorithm more examples and time for predicting the concept c. It seems that only by restricting the representation language of the targets are we able to construct polynomial prediction algorithms.

A very useful way of producing restricted representation classes is the approach taken in [33]. Representation classes are defined by the complexity of their evaluation problem. Now all the well studied standard complexity classes contained in $\mathcal{P}$ lead to corresponding classes of representations (see Definition 6.2).

8 Techniques for proving non-reductions

In Section 5.1 we have seen several cases of pairs of representation classes R_1 and R_2, for which the following property holds: for any $r_1 \in R_1$, the size of the smallest $r_2 \in R_2$ which represents the same concept as r_1 (i.e. $c_1(r_1) = c_2(r_2)$) grows in the worst case exponentially in the size of r_1. Expressed differently, $R_1 \not\trianglelefteq R_2$, provided that the word transformation is the identity on w_1. We give a number of such non-reducible pairs of representation classes (see [14] for additional examples):

Theorem 8.1 *If the word transformation is restricted to be the identity on w_1, then $R_{DNF} \not\trianglelefteq R_{DT}$, $R_{DNF} \not\trianglelefteq R_{DFA}$, $R_{NFA} \not\trianglelefteq R_{DFA}$, $R_{XOR^{sum}} \not\trianglelefteq R_{DNF}$, $R_{XOR^{sum}} \not\trianglelefteq R_{CNF}$, $R_{DNF} \not\trianglelefteq R_{CNF}$, and $R_{CNF} \not\trianglelefteq R_{DNF}$.*

Proof sketch: Assume for the rest of this proof that f must be the identity on w_1. The fact that $R_{DNF} \not\trianglelefteq R_{DT}$ was shown in [21]. (In Section 5.1 we gave a DNF expression for which there only exists exponentially sized equivalent decision trees.) In Section 5.1 we already showed that $R_{DNF} \not\trianglelefteq R_{DFA}$. The remaining non-reductions are trivial for the case that f is identity on w_1. □

Note that for a modified version of the notion of reduction than the one given in this paper (Definition 5.3), it is easy to show [30,33] that $R_{DNF} \trianglelefteq R_{CNF}$ and $R_{CNF} \trianglelefteq R_{DNF}$ even if f is the identity on w_1. Simply replace Requirement (1) by

(1') $w_1 \in c_1(r_1)$ iff $f(w_1, s, n) \notin c_2(g(r_1, n))$.

In Requirement (1'), the word transformation maps examples to examples with the opposite labels, whereas in the case of Requirement (1), the labels of the examples must be preserved by the word transformation. For the sake of simplicity of the presentation we only used Requirement (1) for the definition of reduction (Definition 5.3) and for the rest of this section.

Even though $R_{DNF} \not\trianglelefteq R_{DFA}$ holds if f is restricted to be the identity on w_1, one can still show that $R_{DNF} \trianglelefteq R_{DFA}$ (see Theorem 5.7) by chosing the polynomial time computable word transformation f judiciously. However, in other cases such as the pair "R_{NFA}, R_{DFA}" this seems to be hard to show.[4]

Our goal is to obtain proofs for the fact that $R_1 \not\trianglelefteq R_2$, that hold even if f is allowed to be an arbitrary polynomially length preserving, polynomially computable word transformation (these are the conditions imposed on f in the definition of $\trianglelefteq$). We achieve this by relaxing the notion of reduction and proving non-reducibility with respect to the less restricted notion.

Definition 8.2 *Let R_1 and R_2 be representation classes. Then R_1 is* freely reducible to R_2 *(denoted by $R_1 \sqsubseteq R_2$) iff for all polynomially length preserving word transformations f and polynomially length preserving representation transformations g, there exists $s, n \in \mathbb{N}$, $r_1 \in R_1^{[s]}$, and $w_1 \in \Sigma^{[n]}$, such that Requirement (1) does hold. Let $R_1 \not\sqsubseteq R_2$ denote the fact that $R_1 \sqsubseteq R_2$ does not hold.*

Note that $R_1 \sqsubseteq R_2$ iff $R_1 \trianglelefteq R_2$ but Requirement (2) is dropped (see Definition 5.3), i.e. the word transformation f does not have to be computable in polynomial time.

Clearly, $R_1 \trianglelefteq R_2$ implies $R_1 \sqsubseteq R_2$ and $R_1 \not\sqsubseteq R_2$ implies $R_1 \not\trianglelefteq R_2$.

The reason that we dropped the requirement that f is polynomially computable in the definition of $\not\sqsubseteq$ is that we can develop a purely combinatorial characterization that is necessary and sufficient for the fact that $R_1 \not\sqsubseteq R_2$. Restricting f to be polynomial time computable would give additional leverage for showing that $R_1 \not\trianglelefteq R_2$. However, it seems hard to make use of that leverage.

[4] Note that if $\mathcal{NLOG} = \mathcal{LOG}$ then by Theorem 6.3, $R_{NFA} \trianglelefteq R_{DFA}$. It is unclear whether the opposite implication holds.

We now introduce the tools that lead to a combinatorial characterization of $\not\exists$ and then give a few examples of how to apply the characterization. Further research needs to be done to investigate the most common representation classes used in Artificial Intelligence and Computational Learning Theory with respect to this characterization.

We first introduce an equivalence relation that has the property that all words in the same equivalence class always receive the same label.

Definition 8.3 *For all representation classes (R, Γ, c, Σ), all $s, n \in \mathbb{N}$ and all $v, w \in \Sigma^{[n]}$, the word v* is equivalent to *w (denoted by $u \equiv_{R,s,n} v$), if for all $r \in R^{[s]}$, $label_{c(r)}(v) = label_{c(r)}(w)$. Let $P(\equiv_{R,s,n})$ be any subset of $\Sigma^{[n]}$ which contains one element (*representative*) from each equivalence class of the relation $\equiv_{R,s,n}$. Let $N(\equiv_{R,s,n})$ denote the number of equivalence classes of the relation $\equiv_{R,s,n}$.*

All words of $\Sigma^{[n]}$ that lie in the same equivalence class of the relation $\equiv_{R,s,n}$ are labeled identically for each representation in $R^{[s]}$. Thus $f(.,s,n)$ could simply map all words of $\Sigma^{[n]}$ that lie in the same class to the same image word.

Definition 8.4 *For any set Q, $\langle Q \rangle$ denotes some fixed sequence containing exactly the elements of Q. Let $\langle Q \rangle = (x_1, \cdots, x_q)$ and let h be a function on the elements of Q, then $h(\langle Q \rangle)$ denotes the sequence $(h(x_1), \cdots, h(x_q))$.*

For a given sequence D of words in Σ^* and a representation r, we are interested in the sequence of plus labels and minus labels induced by r on D.

Definition 8.5 *Let R be a representation class and D be a finite sequence of words of Σ^*. For $r \in R$, the sequence $label_{c(r)}(D)$ is called the* dichotomy induced by *r on D. Similarly, we define the* set of all dichotomies induced by *R on the sequence D (denoted by $\Pi_R(D)$) as the set $\{label_{c(r)}(D) : r \in R\}$.*

The set of dichotomies $\Pi_R(D)$ induced by a representation class R on a sequence D has been used in pattern recognition [10,40] and in Computational Learning Theory [9] to measure the "richness" of the concept class associated with a representation class. (A more commonly used definition is $\Pi_R(D) = \{D \cap c(r) : r \in R\}$, where D is a subset of the domain Σ^* [9,10,22,40].) Intuitively, the larger the set of dichotomies, the harder is the learning task. (See [9] of how this intuition is formalized.) It is easy to obtain bounds on the number of dichotomies in $\Pi_R(D)$ using the following parameter of a representation class called the *Vapnik-Chervonenkis (VC) dimension*[5].

[5]Usually the VC dimension is defined for concept classes rather than representation classes. The same parameter is called *capacity* of a concept class C in [40] (named after a similar notion in [10]) and is denoted by $S(C)$ in [11].

Definition 8.6 *A set* $Q \subseteq \Sigma^*$ *is* shattered by R *if* $|\Pi_R(\langle Q \rangle)| = 2^{|Q|}$. *The VC dimension of a representation class* R *is the cardinality of the largest set that is shattered by* R.

Lemma 8.7 [22,40] *Let* R *be a representation class with VC dimension* d *and let* D *be any sequence of* m *words in* Σ^*. *Then* $|\Pi_R(D)| \leq m^d + 1$.

If the concept and the representation alphabets contain the set of reals $\mathbb{R}$ and if the concepts are subsets of some higher dimensional Euclidean domain, then the VC dimension (and more precisely $|\Pi_R(D)|$ is especially useful for estimating the number of examples necessary and sufficient for learning [9,13]. The methodology of the VC dimension has also been generalized to case of learning real-valued function classes [18,31,36,40].

Note that the set of dichotomies $\Pi_{R^{[s]}}(\langle P(\equiv_{R,s,n}) \rangle)$ is independent of the choice of the representative set $P(\equiv_{R,s,n})$. We now give a purely combinatorial characterization of $\sqsubseteq$.

Theorem 8.8 *Let* R_1 *and* R_2 *be representation classes. Then* $R_1 \sqsubseteq R_2$ *iff for all polynomials* p *and* q, *there exist* $s, n \in \mathbb{N}$, *such that for all sequences* $D_{s,n}$ *of* $N(\equiv_{R_1,s,n})$ *words of* $\Sigma_2^{[p(s,n)]}$, *we have that* $\Pi_{R_1^{[s]}}(\langle P(\equiv_{R_1,s,n}) \rangle) \not\subseteq \Pi_{R_2^{[q(s,n)]}}(D_{s,n})$.

Proof: We prove the following equivalent restatement of the theorem: $R_1 \sqsubseteq R_2$ iff there exist polynomials p and q, such that for all $s, n \in \mathbb{N}$, there exists a sequence $D_{s,n}$ of $N(\equiv_{R_1,s,n})$ words of $\Sigma_2^{[p(s,n)]}$, for which $\Pi_{R_1^{[s]}}(\langle P(\equiv_{R_1,s,n}) \rangle) \subseteq \Pi_{R_2^{[q(s,n)]}}(D_{s,n})$ holds.

For the forward direction assume that $R_1 \sqsubseteq R_2$. Then there is a polynomially length preserving word transformation f and polynomially length preserving representation transformation g, such that for all $s, n \in \mathbb{N}, r_1 \in R_1^{[s]}$ and $w \in \Sigma_1^{[n]}$, Requirement (1) holds. Since g is polynomially length preserving there exists a polynomial q, such that $|g(r_1, n)| \leq q(|r_1|, n)$. Requirement (1) implies that the dichotomy induced by r_1 on the sequence $\langle P(\equiv_{R_1,s,n}) \rangle$ is the same as the dichotomy induced by $g(r_1, n) \in R^{[q(s,n)]}$ on the sequence $f(\langle P(\equiv_{R_1,s,n}) \rangle, s, n)$. Let $D_{s,n} = f(\langle P(\equiv_{R_1,s,n}) \rangle, s, n)$. Since f is polynomially length preserving, there exists a polynomial p such that for all $w_1 \in \Sigma_1^{[n]}$, $|f(w_1, s, n)| \leq p(s, n)$. We follow that for all $s, n \in \mathbb{N}$ and the sequence $D_{s,n}$ of $\Sigma_2^{[p(s,n)]}$, $\Pi_{R_1^{[s]}}(\langle P(\equiv_{R_1,s,n}) \rangle) \subseteq \Pi_{R_2^{[q(s,n)]}}(D_{s,n})$. This completes the proof of the forward direction of restatement of the theorem.

For the opposite direction, let p and q be two polynomials, such that for all $s, n \in \mathbb{N}$, there exists a sequence $D_{s,n}$ of $N(\equiv_{R_1,s,n})$ words of $\Sigma_2^{[p(s,n)]}$, for which $\Pi_{R_1^{[s]}}(\langle P(\equiv_{R_1,s,n}) \rangle) \subseteq \Pi_{R_2^{[q(s,n)]}}(D_{s,n})$. From p, q and $D_{s,n}$, we will construct a polynomially length preserving word transformation f and a polynomially length preserving representation transformation g witnessing the fact that $R_1 \sqsubseteq R_2$ holds.

First define $f(., s, n)$ as a mapping from the set of representatives $P(\equiv_{R_1,s,n})$ to the elements of the sequence $D_{s,n}$: the ith element of sequence $\langle P(\equiv_{R_1,s,n}) \rangle$ is mapped to ith element of $D_{s,n}$.

Then extend $f(.,s,n)$ to the domain $\Sigma_1^{[n]}$ by mapping all words in each equivalence class of the relation $\equiv_{R_1,s,n}$ to the same word of $D_{s,n}$ as the representative of the class. Since $D_{s,n}$ consists of words of $\Sigma_2^{[p(s,n)]}$, the defined word transformation f is polynomially length preserving.

Let $g(r_1,n)$ consist of a representation $r_2 \in R^{[q(s,n)]}$ such that the dichotomy induced by r_2 on the sequence $D_{s,n}$ is the same as the dichotomy induced by $r_1 \in R_1^{[s]}$ on the sequence $\langle P(\equiv_{R_1,s,n})\rangle$. Note that by the above assumption r_2 exists and observe that the defined representation transformation g is also polynomially length preserving (with the polynomial q). From the definition of f and g it follows that they fulfill Requirement (1) and this completes the proof of the opposite direction of the restatement of the theorem given at the beginning of the proof. □

The following corollaries give sufficient conditions for $R_1 \not\sqsubseteq R_2$.

The first one follows from the above theorem and the observation that $|\Pi_{R_1^{[s]}}(\langle P(\equiv_{R_1,s,n})\rangle)| = |\Pi_{R_1^{[s]}}(\langle \Sigma_1^{[n]}\rangle)|$.

Corollary 8.9 *Let R_1 and R_2 be representation classes. Then $R_1 \not\sqsubseteq R_2$, if for all polynomials p and q, there exist $s,n \in \mathbb{N}$ such that for all sequences D of $N(\equiv_{R_1,s,n})$ words of $\Sigma_2^{[p(s,n)]}$, the inequality $|\Pi_{R_1^{[s]}}(\Sigma_1^{[n]})| > |\Pi_{R_2^{[q(s,n)]}}(D)|$ holds.*

Corollary 8.10 *Let R_1 and R_2 be representation classes. Then $R_1 \not\sqsubseteq R_2$, if for all polynomials p and q, there exist $s,n \in \mathbb{N}$ and $Q_{s,n} \subseteq \Sigma^{[n]}$, such that for all sequences D of $|Q_{s,n}|$ words of $\Sigma_2^{[p(s,n)]}$, we have that $\Pi_{R_1^{[s]}}(\langle Q_{s,n}\rangle) \not\subseteq \Pi_{R_2^{[q(s,n)]}}(D)$.*

Corollary 8.11 *Let R_1 and R_2 be representation classes. Then $R_1 \not\sqsubseteq R_2$, if the VC dimension of R_1 is larger than the VC dimension of R_2.*

Proof: We will apply the previous corollary to show this. Let Q be any maximum cardinality subset of Σ_1^* that is shattered by R_1. Choose s and n such that $Q \subseteq \Sigma^{[n]}$ and $|\Pi_{R_1^{[s]}}(\langle Q\rangle)| = 2^{|Q|}$. Since the VC dimension of R_1 is larger than the VC dimension of R_2, we have that for all sequences D of $|Q|$ words of Σ_2^*, $\Pi_{R_1^{[s]}}(\langle Q\rangle) \not\subseteq \Pi_{R_2^*}(D)$. The proof now follows from the previous corollary: for all polynomials p and q, choose s and n as above and $Q_{s,n} = Q$. □

It is easy to see that the VC dimension of R_{PAIRS} is two and the VC dimension of R_{SINGLE} is one. From the previous corollary we follow that

Theorem 8.12 $R_{PAIRS} \not\sqsubseteq R_{SINGLE}$.

The above theorem implies that $R_{PAIRS} \not\preceq R_{SINGLE}$. It is easy to show that $R_{SINGLE} \preceq R_{PAIRS}$.

The converse of the previous corollary clearly does not hold. We will show that $R_{SINGLE} \not\sqsubseteq R_{\Box}$, even though the VC dimension of R_{SINGLE} is one and the VC dimension of $R_{\Box}$ is infinite (More precisely, the VC dimension of $R_{\Box}$ is $2n$ when restricted to boxes in $\mathbb{R}^n$ [9,23].)

Theorem 8.13 *$R_{SINGLE} \not\sqsubseteq R_{\Box}$ and $R_{\Box} \not\sqsubseteq R_{SINGLE}$.*

Proof: Using the previous corollary, the second half of the theorem follows from the fact that the VC dimension of $R_{\Box}$ is larger than the VC dimension of R_{SINGLE}. The proof of the fact that $R_{SINGLE} \not\sqsubseteq R_{\Box}$ is more involved. Let c be a constant such that all representations of R_{SINGLE} that represent concepts of the form $\{0,1\}^n - w$, where $w \in \{0,1\}$, have length at most cn. Clearly such a constant exists. We will use Corollary 8.10 to proof this theorem. Arbitrarily choose polynomials p and q. Choose n such that $p(cn,n) < 2^{n-1}$. Furthermore, set s to cn and $Q_{s,n}$ to the set of all 2^n bit patterns of length n. From the choice of s we have that $\Pi_{R_{SINGLE}^{[s]}}(\langle Q_{s,n} \rangle)$ contains all dichotomies of length 2^n with exactly one minus. Denote the set $\Pi_{R_{SINGLE}^{[s]}}(\langle Q_{s,n} \rangle)$ as T.

The concept alphabet of $R_{\Box}$ is the set of all reals $\mathbb{R}$. Let D be any sequence of 2^n points of $\mathbb{R}^{[p(cn,n)]}$. To apply Corollary 8.10 we still have to show that $T \not\subseteq \Pi_{R_{\Box}^{[q(s,n)]}}(D)$.

Let $d = p(cn,n)$ and let $[x_1,x_2] \times [x_3,x_4] \times \cdots \times [x_{2d-1},x_{2d}]$. be the smallest box in $\mathbb{R}^d$ that contains all points of D. Assume there is point $y \in D$ that is in the interior of the box, i.e. in $(x_1,x_2) \times (x_3,x_4) \times \cdots \times (x_{2d-1},x_{2d})$. Clearly, there is no concept (box) in $R_{\Box}$ that labels y as minus and the remaining points of D as plus. Thus if there is a point y in the interior, then $T \not\subseteq \Pi_{R_{\Box}^{[q(s,n)]}}(D)$.

We are left with the case that all points of D lie on the faces of the smallest box. Each x_i $(1 \leq i \leq 2d)$ corresponds to one of the $2d$ faces of the box. We say a point of D *owns* a face if it is the only point of D that lies on that face. Since $2d = 2p(cn,n) < 2^n$ and D has length 2^n, there must exist a point $y \in D$ that does not own any of the $2d$ faces of the smallest box. Similarly to the above case, there is no box in $R_{\Box}$ that labels y as minus and the remaining points of D as plus and thus $T \not\subseteq \Pi_{R_{\Box}^{[q(s,n)]}}(D)$ holds. □

Once we have shown for two representation classes that $R_1 \not\sqsubseteq R_2$, then more results of this type may be obtained by applying the following lemma.

Lemma 8.14 *If $R_1 \not\sqsubseteq R_2$, $R_1 \sqsubseteq R_a$ and $R_b \sqsubseteq R_2$, then $R_a \not\sqsubseteq R_b$.*

Corollary 8.15 *For all classes $R_1 \in \{R_{PAIRS}, R_{2-clauseCNF}, R_{DT}, R_{DFA}\}$ and $R_2 \in \{R_{\Box}, R_{monomials}\}$, $R_1 \not\sqsubseteq R_2$.*

Proof sketch: It is easy to show the following reductions: $R_{PAIRS} \trianglelefteq R_{2-clauseCNF}$, $R_{PAIRS} \trianglelefteq R_{DT}$, $R_{PAIRS} \trianglelefteq R_{DFA}$ and $R_{monomials} \trianglelefteq R_{\Box}$. Now the corollary follows from the previous lemma and theorem. □

By the above corollary, $R_{SINGLE} \not\sqsubseteq R_{monomials}$. Recall that in our definition of $R_{monomials}$ we assumed (as done for all Boolean function in this paper) that assigments for n variables are encoded

as bit patters of length n. For a somewhat non-standard way of encoding assignments (which we *only* use in the next theorem) we can show that $R_{SINGLE} \trianglelefteq R_{monomials}$.

Theorem 8.16 *If assignments of n variables are encoded by a list that contains the numbers (all in binary) n and all indices whose values are false, then $R_{SINGLE} \trianglelefteq R_{monomials}$.*

Proof: For any $w_1 \in \{0,1\}^*$, let $N(w_1)$ denote the number encoded by the binary pattern w. Let $f(w_1, s, n)$ be a list containing 2^n in binary and the bit pattern w_1. Note that $f(w_1, s, n)$ encodes an assignment of 2^n variables, all of which are set to true except the one with index $N(w_1)$. For $w_1 \in \{0,1\}^n$, let $r_{w_1} \in R_{SINGLE}$ be a representation of the concept $\{0,1\}^{2^n} - B(w_1)$, where $B(w_1)$ consists of the bit pattern of length 2^n with exactly one 0 in position $N(w_1)$ and all other bits 1. Let $g(r_{w_1}, n)$ be the encoding of the monomial over 2^n variables which consists of only one unnegated literal, the one with index $N(w_1)$. Since all indices and the number of variables 2^n are encoded in binary, f and g are polynomially length preserving. It is easy to see that Requirement (1) holds, which completes the reduction. □

9 Conclusion

Our notion of non-reducibility between representation classes is based on the definition of reduction (the relation $\trianglelefteq$ of Definition 5.3) between representation classes introduced in [33]. This definition of reduction follows the spirit of the standard many-one reducibilities used in Complexity Theory [24]. One may define a version of a Turing reduction [24] between representation classes that preserves polynomial predictability. Useful notions of reduction should have the following property: If R_1 reduces to R_2 and R_2 is polynomially predictable, then R_1 is polynomially predictable as well. It would be interesting to develop notions of non-reducibility which are based on a Turing reduction between representation classes. Is there also a combinatorial characterization of the fact that R_1 is not reducible to R_2 (as in Theorem 8.8)?

Observe that there are polynomially predictable representation classes such that both $R_1 \not\trianglelefteq R_2$ and $R_2 \not\trianglelefteq R_1$ holds (See Theorem 8.13). However, we think even independent of the objectives of Computational Learning Theory the definition of $\not\trianglelefteq$ and its combinatorial characterization is a useful combinatorial tool for studying representation classes.

There are any number of open problems of the type is $R_1 \not\trianglelefteq R_2$? We first give a few conjectures that involve representation classes used in this paper.

Conjectures $R_{DNF} \not\trianglelefteq R_{DT}$, $R_{NFA} \not\trianglelefteq R_{DFA}$, $R_{XORsum} \not\trianglelefteq R_{DNF}$, $R_{XORsum} \not\trianglelefteq R_{CNF}$, $R_{DNF} \not\trianglelefteq R_{CNF}$, and $R_{CNF} \not\trianglelefteq R_{DNF}$.

Boolean terms and clauses over n variables can be interpreted as special halfspaces in $\mathbb{R}^n$. Furthermore, DNF expressions correspond to unions and CNF expressions to intersections of such

halfspaces. Do the above conjectures where R_{DNF} and R_{CNF} appear on the right-hand-side of $\sqsubseteq$ still hold if R_{DNF} and R_{CNF} are replaced by their geometric generalizations: unions and intersection of arbitrary halfspaces, respectively. (Let $\mathbb{R}$ be the concept and representation alphabet and represent a halfspace by the list of the coefficients of the bordering hyperplane.)

We conclude with a simple conjecture involving halfspaces.

Conjecture Let R_1 be the representation class defining halfspaces of $\mathbb{R}^n$ (where n depends on the concept) and let R_2 be the representation class defining intersections of two such halfspaces. Then $R_2 \sqsubseteq R_1$. More strongly, we conjecture that $R_{2Bool} \sqsubseteq R_1$, where R_{2Bool} is the same as R_2, but now the the intersecting halfspaces are restricted to halfspaces of Boolean hypercubes $\{0,1\}^n$ (where n depends on the concept).

The representation class R_1 is learnable in terms of itself using linear programming [9]. However, we know of no polynomial prediction algorithm for R_2 or for R_{2Bool}.[6]

The above conjecture is particularly interesting in view of the fact that $R_2' \trianglelefteq R_1$, where the concepts of R_2' are symmetric differences of halfspaces instead of intersections of halfspaces [39] as for R_2.

Acknowledgements

The notion of reduction used extensively in this paper was developed together with Lenny Pitt [33]. The basic definitions given here and many example reductions are from or are very similar to the ones given in our joint paper [33]. Thanks to him for giving me feedback on early drafts of this paper and for lively discussions of the material. Thanks also to David Haussler, David Helmbold and Phil Long for stimulating conversations.

References

[1] N. Abe. Polynomial learnability as a formal model of natural language acquisition. 1989. Ph.D. Thesis, in preparation, Department of Computer and Information Science, University of Pennsylvania.

[2] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, London, 1974.

[3] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1987.

[6] In [7] it is shown that R_2 is not learnable in terms of R_2 unless $\mathcal{RP} \neq \mathcal{NP}$. (See Section 3 for a discussion of results of this type.)

[4] D. Angluin and C. H. Smith. Inductive inference: theory and methods. *Computing Surveys*, 15(3):237–269, 1983.

[5] J. M. Barzdin. *Prognostication of automata and functions*, pages 81–84. Elsevier North-Holland, New York, 1972.

[6] J. M. Barzdin and R. V. Freivald. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.

[7] A. Blum and R. Rivest. Training a 3-node neural network is $\mathcal{NP}$–complete. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, Morgan Kaufmann, San Mateo, CA, August 1988.

[8] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Inform. Contr.*, 28:125–155, 1975.

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. *Learnability and the Vapnik-Chervonenkis Dimension.* Technical Report UCSC-CRL-87-20, Department of Computer and Information Sciences, University of California, Santa Cruz, November 1987. To appear, *J. ACM.*

[10] T. Cover. Geometrical and statistical properties of systems of linear inequalities with applications to pattern recognition. *IEEE Trans. Elect. Comp.*, 14:4326–334, 1965.

[11] L. P. Dudley. *A Course on Empirical Processes.* Springer Verlag, New York, 1984. Lecture Notes in Mathematics No. 1097.

[12] A. Ehrenfeucht and D. Haussler. *Learning Decision Trees from Random Examples.* Technical Report UCSC-CRL-87-15, Department of Computer and Information Sciences, University of California, Santa Cruz, October 1987.

[13] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. G. Valiant. A general lower bound on the number of examples needed for learning. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 139–154, Morgan Kaufmann, San Mateo, CA, August 1988.

[14] S. I. Gallant. Representability theory. 1988. Unpublished manuscript.

[15] J. Gill. Probabilistic Turing machines. *SIAM J. Computing*, 6(4):675–695, 1977.

[16] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[17] O. Goldreich, H. Krawczyk, and M. Luby. On the existence of pseudorandom generators. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 12–24, IEEE Computer Society Press, Washington, D.C., October 1988.

[18] D. Haussler. Generalizing the pac model: sample size bounds from metric dimension-based uniform convergence results. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Washington, D.C., October 1989.

[19] D. Haussler. *Learning Conjunctive Concepts in Structural Domains.* Technical Report UCSC-CRL-87-01, Department of Computer and Information Sciences, University of California, Santa Cruz, February 1987. To appear in *Machine Learning.*

[20] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of models for polynomial learnability. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 42–55, Morgan Kaufmann, San Mateo, CA, August 1988.

[21] D. Haussler and G. Pagallo. *Feature discovery in empirical learning.* Technical Report UCSC-CRL-88-08, Department of Computer and Information Sciences, University of California, Santa Cruz, October 1987. Revised May 1, 1989.

[22] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Computational Geometry*, 2:127–151, 1987.

[23] D. Helmbold, R. Sloan, and M. K. Warmuth. Learning nested differences of intersection closed concept classes. In *Proceedings of the 1989 Workshop on Computational Learning Theory*, Morgan Kaufmann, San Mateo, CA, August 1989.

[24] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, London, 1979.

[25] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. On the learnability of boolean formulae. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, Assoc. Comp. Mach., New York, May 1987.

[26] M. Kearns and L. Pitt. A polynomial-time algorithm for learning k-variable pattern languages from examples. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Washington, D.C., October 1989.

[27] M. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 433–444, Assoc. Comp. Mach., New York, May 1989.

[28] L. A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[29] J. Lin and J. Vitter. Complexity issues in learning neural nets. In *Proceedings of the 1989 Workshop on Computational Learning Theory*, Morgan Kaufmann, San Mateo, CA, July 1989.

[30] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1987.

[31] B. K. Natarajan. *Some Results on Learning.* Technical Report CMU-RI-TR-89-6, Robotics Institute, Carnegie-Mellon University, 1989.

[32] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *J. ACM*, 35(4):965–984, 1988.

[33] L. Pitt and M. K. Warmuth. Reductions among prediction problems: On the difficulty of predicting automata. In *Proceedings of the 3rd Annual IEEE Conference on Structure in Complexity Theory*, pages 60–69, IEEE Computer Society Press, Washington, D.C., June 1988.

[34] K. M. Podnieks. Comparing various concepts of function prediction, part 1. *Latv. Gosudarst. Univ Uch. Zapiski*, 210:68–81, 1974. (in Russian).

[35] K. M. Podnieks. Comparing various concepts of function prediction, part 2. *Latv. Gosudarst. Univ Uch. Zapiski*, 233:33–44, 1975. (in Russian).

[36] D. Pollard. *Convergence of Stochastic Processes.* Springer-Verlag, New York, 1974.

[37] R. L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[38] R. Schapire. The strength of weak learnability. In *Proceedings of the 1989 Workshop on Computational Learning Theory*, Morgan Kaufmann, San Mateo, CA, August 1989.

[39] L. Valiant and M. K. Warmuth. Predicting symmetric differences of two halfspaces reduces to predicting halfspaces. 1989. Unpublished manuscript.

[40] L. G. Valiant. A theory of the learnable. *Comm. Assoc. Comp. Mach.*, 27(11):1134–1142, 1984.

[41] V. N. Vapnik. *Estimation of Dependencies Based on Empirical Data.* Springer Verlag, New York, 1982.

[42] A. C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, IEEE Computer Society Press, Washington, D.C., November 1982.