The Computational Complexity of Machine Learning

The Computational Complexity of Machine Learning

Michael J. Kearns

The MIT Press
Cambridge, Massachusetts
London, England

Dedicated to my parents

Alice Chen Kearns and David Richard Kearns

For their love and courage

Contents

Preface and Acknowledgements

This book is a revision of my doctoral dissertation, which was completed in May 1989 at Harvard University. While the changes to the theorems and proofs are primarily clarifications of or corrections to my original thesis, I have added a significant amount of expository and explanatory material, in an effort to make the work at least partially accessible to an audience wider than the "mainstream" theoretical computer science community. Thus, there are more examples and more informal intuition behind the formal mathematical results. My hope is that those lacking the background for the formal proofs will nevertheless be able to read selectively, and gain some useful understanding of the goals, successes and shortcomings of computational learning theory.

Computational learning theory can be broadly and imprecisely defined as the mathematical study of *efficient* learning by machines or computational systems. The demand for efficiency is one of the primary characteristics distinguishing computational learning theory from the older but still active areas of inductive inference and statistical pattern recognition. Thus, computational learning theory encompasses a wide variety of interesting learning environments and formal models, too numerous to detail in any single volume. Our goal here is to simply convey the flavor of the recent research by first summarizing work in various learning models and then carefully scrutinizing a single model that is reasonably natural and realistic, and has enjoyed great popularity in its infancy.

This book is a detailed investigation of the computational complexity of machine learning from examples in the *distribution-free* model introduced by L.G. Valiant [93] (also known as the *probably approximately correct* model of learning). In the distribution-free model, a learning algorithm receives positive

and negative examples of an unknown target set (or *concept*) that is chosen from some known class of sets (or *concept class*). These examples are generated randomly according to a fixed but unknown probability distribution representing Nature, and the goal of the learning algorithm is to infer an hypothesis concept that closely approximates the target concept with respect to the unknown distribution. This book is concerned with proving theorems about learning in this formal mathematical model.

As we have mentioned, we are primarily interested in the phenomenon of *efficient* learning in the distribution-free model, in the standard polynomial-time sense. Our results include general tools for determining the polynomial-time learnability of a concept class, an extensive study of efficient learning when errors are present in the examples, and lower bounds on the number of examples required for learning in our model. A centerpiece of the book is a series of results demonstrating the computational difficulty of learning a number of well-studied concept classes. These results are obtained by reducing some apparently hard number-theoretic problems from public-key cryptography to the learning problems. The hard-to-learn concept classes include the sets represented by Boolean formulae, deterministic finite automata and a simplified form of neural networks. We also give algorithms for learning powerful concept classes under the uniform distribution, and give equivalences between natural models of efficient learnability.

The book also includes detailed definitions and motivation for our model, a chapter discussing past research in this model and related models, and a short list of important open problems and areas for further research.

Acknowledgements. I am deeply grateful for the guidance and support of my advisor, Prof. L.G. Valiant of Harvard University. Throughout my stay at Harvard, Les' insightful comments and timely advice made my graduate career a fascinating and enriching experience. I thank Les for his support, for sharing his endless supply of ideas, and for his friendship. I could not have had a better advisor.

Many thanks to my family — my father David, my mother Alice and my sister Jennifer — for all of the love and support you have given. I am proud of you as my family, and proud to be friends with each of you as individuals. I especially thank you for your continued courage during these difficult times.

Many of the results presented here were joint research between myself and coauthors. Here I wish to thank each of these colleagues, and cite the papers in which this research appeared in preliminary form. The example of learning provided in Chapter 2 is adapted from "Recent results on Boolean concept learning", by M. Kearns, M. Li, L. Pitt and L.G. Valiant, appearing in the *Proceedings of the Fourth International Workshop on Machine Learning* [61]. Results from Chapters 4, 6 and 8 appeared in "On the learnability of Boolean formulae", by M. Kearns, M. Li, L. Pitt and L.G. Valiant, in the *Proceedings of the 19th A.C.M. Symposium on the Theory of Computing* [60]. The results of Chapter 5 initially appeared in the paper "Learning in the presence of malicious errors", by M. Kearns and M. Li, in the *Proceedings of the 20th A.C.M. Symposium on the Theory of Computing* [59]. Parts of Chapter 6 appeared in "A general lower bound on the number of examples needed for learning", by A. Ehrenfeucht, D. Haussler, M. Kearns and L.G. Valiant, in *Information and Computation* [36]. Results of Chapters 7, 8 and 9 appeared in "Cryptographic limitations on learning Boolean formulae and finite automata", by M. Kearns and L.G. Valiant, in the *Proceedings of the 21st A.C.M. Symposium on the Theory of Computing* [64]. Working with these five colleagues — Andrzej Ehrenfeucht, David Haussler, Ming Li, Lenny Pitt and Les Valiant — made doing research both fun and exciting. I also had the pleasure of collaborating with Nick Littlestone and Manfred Warmuth [51]; thanks again to you all.

Thanks to the many people who were first colleagues and then good friends. Your presence was one of the most rewarding aspects of graduate school. Special thanks to David Haussler and Manfred Warmuth for their friendship and for their hospitality during my stay at the University of California at Santa Cruz during the 1987-88 academic year. Many thanks to Dana Angluin, Sally Floyd, Ming Li, Nick Littlestone, Lenny Pitt, Ron Rivest, Thanasis Tsantilas and Umesh Vazirani. I also had very enjoyable conversations with Avrim Blum, David Johnson, Prabhakar Raghavan, Jim Ruppert, Rob Schapire and Bob Sloan. I'd also like to thank three particularly inspiring teachers I have had, J.W. Addison, Manuel Blum and Silvio Micali.

Thanks to the members of the Middle Common Room at Merton College, Oxford University for their hospitality during my time there in the spring of 1988.

Thanks to A.T. & T. Bell Laboratories for their generous financial support during my graduate career. I am also grateful for the financial support

provided by the following grants: N00014-85-K-0445 and N00014-86-K-0454 from the Office of Naval Research, and DCR-8600379 from the National Science Foundation. Thanks also for the support of a grant from the Siemens Corporation to M.I.T., where I have been while making the revisions to my thesis.

Thanks to the Theory of Computation Group at M.I.T.'s Laboratory for Computer Science for a great year!

Finally, thanks to the close friends who shared many great times with me during graduate school and helped during the hard parts.

Michael J. Kearns
Cambridge, Massachusetts
May 1990

The Computational Complexity of Machine Learning

1

Introduction

Recently in computer science there has been a great deal of interest in the area of machine learning. In its experimental incarnation, this field is contained within the broader confines of artificial intelligence, and its attraction for researchers stems from many sources. Foremost among these is the hope that an understanding of a computer's capabilities for learning will shed light on similar phenomena in human beings. Additionally, there are obvious social and scientific benefits to having reliable programs that are able to infer general and accurate rules from some combination of sample data, intelligent questioning, and background knowledge.

From the viewpoint of empirical research, one of the main difficulties in comparing various algorithms which learn from examples is the lack of a formally specified model by which the algorithms may be evaluated. Typically, different learning algorithms and theories are given together with examples of their performance, but without a precise definition of "learnability" it is difficult to characterize the scope of applicability of an algorithm or analyze the success of different approaches and techniques.

Partly in light of these empirical difficulties, and partly out of interest in the phenomenon of learning in its own right, the goal of the research presented here is to provide some mathematical foundations for a science of efficient machine learning. More precisely, we wish to define a formal mathematical model of machine learning that is realistic in some (but inevitably not all) important ways, and to analyze rigorously the consequences of our definitions. We expect these consequences to take the form of learning algorithms along with proofs of

their correctness and performance, lower bounds and hardness results that delineate the fundamental computational and information-theoretic limitations on learning, and general principles and phenomena that underly the chosen model.

The notion of a mathematical study of machine learning is by no means new to computer science. For instance, research in the areas known as inductive inference and statistical pattern recognition often addresses problems of inferring a good rule from given data. Surveys and highlights of these rich and varied fields are given by Angluin and Smith [13], Duda and Hart [33], Devroye [31], Vapnik [96] and many others. While a number of ideas from these older areas have proven relevant to the present study, there is a fundamental and significant difference between previous models and the model we consider: the explicit emphasis here on the *computational efficiency* of learning algorithms.

The model we use, sometimes known as the *distribution-free* model or the model of *probably approximately correct* learning, was introduced by L.G. Valiant [93] in 1984 and has been the catalyst for a renaissance of research in formal models of machine learning known as *computational learning theory*. Briefly, Valiant's framework departs from models used in inductive inference and statistical pattern recognition in one or more of three basic directions:

The demand that a learning algorithm identify the hidden target rule *exactly* is relaxed to allow *approximations*. Most inductive inference models require that the learning algorithm eventually converge on a rule that is functionally equivalent to the target rule.

The demand for computational efficiency is now an explicit and central concern. Inductive inference models typically seek learning algorithms that perform exact identification "in the limit"; the classes of functions considered are usually so large (e.g., the class of all recursive functions) that improved computational complexity results are not possible. While one occasionally finds complexity results in the pattern recognition literature (particularly in the area of required sample size), computational efficiency is in general a secondary concern.

The demand is made for general learning algorithms that perform well against any probability distribution on the data. This gives rise to the expres-

sion *distribution-free*. Statistical pattern recognition models often deal with special distributions; the notable instances in which general classes of distributions are addressed (for example, the work of Vapnik and Chervonekis [97], Vapnik [96], Pollard [81], Dudley [34] and others) have found widespread application in our model and related models.

The simultaneous consideration of all three of these departures can be regarded as a step towards a more realistic model, since the most remarkable examples of learning, those which occur in humans and elsewhere in Nature, appear to be imperfect but rapid and general.

Research in computational learning theory clearly has some relationship with empirical machine learning research conducted in the field of artificial intelligence. As might be expected, this relationship varies in strength and relevance from problem to problem. Ideally, the two fields would complement each other in a significant way, with experimental research suggesting new theorems to be proven, and vice-versa. Many of the problems tackled by artificial intelligence, however, appear extremely complex and are poorly understood in their biological incarnations, to the point that they are currently beyond mathematical formalization. The research presented here does not pretend to address such problems. However, the fundamental hypothesis of this research is that there are important practical and philosophically interesting problems in learning that *can* be formalized and that therefore must obey the same "computational laws" that appear elsewhere in computer science.

This book, along with other research in computational learning theory, can be regarded as a first step towards discovering how such laws apply to our model of machine learning. Here we restrict our attention to programs that attempt to learn an unknown target rule (or *concept*) chosen from a known concept class on the basis of examples of the target concept. This is known as *learning from examples*. Valiant's model considers learning from examples as a starting point, with an emphasis on computational complexity. Learning algorithms are required to be *efficient*, in the standard polynomial-time sense. The question we therefore address and partially answer in these pages is: *What does complexity theory have to say about machine learning from examples?*

As we shall see, the answer to this question has many parts. We begin in Chapter 2 by giving the precise definition of the distribution-free model, along with the motivations for this model. We also provide a detailed example of an

efficient algorithm for a natural learning problem in this model, and give some needed facts and notation. Chapter 3 provides an overview of some recent research in computational learning theory, in both the distribution-free model and other models. Here we also state formally a theorem due to Blumer, Ehrenfeucht, Haussler and Warmuth known as *Occam's Razor* that we will appeal to frequently.

Our first results are presented in Chapter 4. Here we describe several useful tools for determining whether a concept class is efficiently learnable. These include methods for *composing* existing learning algorithms to obtain new learning algorithms for more powerful concept classes, and a notion of *reducibility* that allows us to show that one concept class is "just as hard" to learn as another. This latter notion, which has subsequently been developed by Pitt and Warmuth, plays a role analogous to that of polynomial-time reductions in complexity theory.

Chapter 5 is an extensive study of a variant of the distribution-free model which allows errors to be present in the examples given to a learning algorithm. Such considerations are obviously crucial in any model that aspires to reality. Here we study the largest rate of error that can be tolerated by efficient learning algorithms, emphasizing worst-case or *malicious errors* but also considering classification noise. We give general upper bounds on the error rate that can be tolerated that are based on various combinatorial properties of concept classes, as well as efficient learning algorithms that approach these optimal rates.

Chapter 6 presents information-theoretic lower bounds (that is, bounds that hold regardless of the amount of computation time) on the number of examples required for learning in our sense, including a general lower bound that can be applied to any concept class.

In Chapter 7 we prove that several natural and simple concept classes are not efficiently learnable in the distribution-free setting. These classes include concepts represented by Boolean formulae, deterministic finite automata, and a simple class of neural networks. In contrast to previous hardness results for learning, these results hold regardless of the form in which a learning algorithm represents it hypothesis. The results rely on some standard assumptions on the intractability of several well-studied number theoretic problems (such as the difficulty of factoring), and they suggest and formalize an interesting du-

ality between learning, where one desires an efficient algorithm for classifying future examples solely on the basis of given examples, and public-key cryptography, where one desires easily computed encoding and decoding functions whose behavior on future messages cannot be efficiently inferred from previous messages. As a non-learning application of these results, we are able to obtain rather strong hardness results for approximating the optimal solution for various combinatorial optimization problems, including a generalization of the well-known graph coloring problem.

In Chapter 8 we give efficient algorithms for learning powerful concept classes when the distribution on examples is uniform. Here we are motivated either by evidence that learning in a distribution-free manner is intractable or the fact that the learnability of the class has remained unresolved despite repeated attacks. Such partial positive results are analogous to results giving efficient average-case algorithms for problems whose worst-case complexity is *NP*-complete.

Finally, Chapter 9 demonstrates the equivalence of two natural models of learning with examples, and relates this to other recently shown equivalences. In addition to allowing us to transform existing learning algorithms to new algorithms meeting different performance criteria, such results give evidence for the robustness of the original model, since it is invariant to reasonable but apparently significant modifications. We give conclusions and mention some important open problems and areas for further research in Chapter 10.

We feel that the results presented here and elsewhere in computational learning theory demonstrate that a wide variety of topics in theoretical computer science and other branches of mathematics have a direct and significant bearing on natural problems in machine learning. We hope that this line of research will continue to illuminate the phenomenon of efficient machine learning, both in the model studied here and in other natural models.

A word on the background assumed of the reader: it is assumed that the reader is familiar with the material that might be found in a good first-year graduate course in theoretical computer science, and thus is comfortable with the analysis of algorithms and notions such as *NP*-completeness. We refer the reader to Aho, Hopcroft and Ullman [3], Cormen, Leiserson and Rivest [30], and Garey and Johnson [39]. Familiarity with basic results from probability theory and public-key cryptography is also helpful, but not necessary.

2

Definitions and Motivation for Distribution-free Learning

In this chapter we give definitions and motivation for the model of machine learning we study. This model was first defined by Valiant [93] in 1984. In addition to the basic definitions and notation, we provide a detailed example of an efficient algorithm in this model, give the form of Chernoff bounds we use, define the Vapnik-Chervonenkis dimension, and define a number of classes of representations whose learnability we will study.

2.1 Representing subsets of a domain

Concept classes and their representation. Let X be a set called a *domain* (also sometimes referred to as the *instance space*). We think of X as containing encodings of all objects of interest to us in our learning problem. For example, each instance in X may represent a different object in a particular room, with discrete attributes representing properties such as color, and continuous values representing properties such as height. The goal of a learning algorithm is then to infer some unknown subset of X, called a *concept*, chosen from a known *concept class*. (The reader familiar with the pattern recognition literature may regard the assumption of a known concept class as representing the *prior knowledge* of the learning algorithm.) In this setting, we might imagine a child attempting to learn to distinguish chairs from non-chairs among all the

physical objects in its environment. This particular concept is but one of many concepts in the class, each of which the child might be expected to learn and each of which is a set of objects that are related in some natural and interesting manner. For example, another concept might consist of all metal objects in the environment. On the other hand, we would not expect a randomly chosen subset of objects to be an interesting concept, since as humans we do not expect these objects to bear any natural and useful relation to one another. Thus we are primarily interested in the learnability of concept classes that are expressible as relatively *simple rules* over the domain instances.

For computational purposes we always need a way of *naming* or *representing* concepts. Thus, we formally define a *representation class over* X to be a pair (σ, C), where $C \subseteq \{0,1\}^*$ and σ is a mapping $\sigma : C \to 2^X$ (here 2^X denotes the power set of X). In the case that the domain X has real-valued components, we sometimes assume $C \subseteq (\{0,1\} \cup R)^*$, where R is the set of real numbers. For $c \in C$, $\sigma(c)$ is called a *concept* over X; the image space $\sigma(C)$ is the *concept class* that is *represented* by (σ, C). For $c \in C$, we define $pos(c) = \sigma(c)$ (the *positive examples* of c) and $neg(c) = X - \sigma(c)$ (the *negative examples* of c). The domain X and the mapping σ will usually be clear from the context, and we will simply refer to the *representation class* C. We will sometimes use the notation $c(x)$ to denote the value of the characteristic function of $\sigma(c)$ on the domain point x; thus $x \in pos(c)$ ($x \in neg(c)$, respectively) and $c(x) = 1$ ($c(x) = 0$, respectively) are used interchangeably. We assume that domain points $x \in X$ and representations $c \in C$ are efficiently encoded using any of the standard schemes (see Garey and Johnson [39]), and denote by $|x|$ and $|c|$ the length of these encodings measured in bits (or in the case of real-valued domains, some other reasonable measure of length that may depend on the model of arithmetic computation used; see Aho, Hopcroft and Ullman [3]).

Parameterized representation classes. We will often study *parameterized* classes of representations. Here we have a stratified domain $X = \cup_{n \geq 1} X_n$ and representation class $C = \cup_{n \geq 1} C_n$. The parameter n can be regarded as an appropriate measure of the complexity of concepts in $\sigma(C)$ (such as the number of domain attributes), and we assume that for a representation $c \in C_n$ we have $pos(c) \subseteq X_n$ and $neg(c) = X_n - pos(c)$. For example, X_n may be the set $\{0,1\}^n$, and C_n

the class of all Boolean formulae over n variables whose length is at most n^2. Then for $c \in C_n$, $\sigma(c)$ would contain all satisfying assignments of the formula c.

Efficient evaluation of representations. In general, we will be primarily concerned with learning algorithms that are computationally efficient. In order to prevent this demand from being vacuous, we need to insure that the *hypotheses* output by a learning algorithm can be efficiently evaluated as well. For example, it would be of little use from a computational standpoint to have a learning algorithm that terminates rapidly but then outputs as its hypothesis a complicated system of differential equations that can only be evaluated using a lengthy stepwise approximation method (although such an hypothesis may be of considerable theoretical value for the model it provides of the concept being learned). Thus if C is a representation class over X, we say that C is *polynomially evaluatable* if there is a (probabilistic) polynomial-time *evaluation algorithm* A that on input a representation $c \in C$ and a domain point $x \in X$ outputs $c(x)$. For parameterized C, an alternate and possibly more general definition is that of *nonuniformly polynomially evaluatable*. Here for each $c \in C_n$, there is a (probabilistic) *evaluation circuit* A_c that on input $x \in X_n$ outputs $c(x)$, and the size of A_c is polynomial in $|c|$ and n. Note that a class being nonuniformly polynomially evaluatable simply means that it contains only "small" representations, that is, representations that can be written down in polynomial time. All representation classes considered here are polynomially evaluatable. It is worth mentioning at this point that Schapire [90] has shown that if a representation class is not nonuniformly polynomially evaluatable, then it is not efficiently learnable in our model. Thus, perhaps not surprisingly we see that classes that are not polynomially evaluatable constitute "unfair" learning problems.

Samples. A *labeled example* from a domain X is a pair $< x, b >$, where $x \in X$ and $b \in \{0, 1\}$. A *labeled sample* $S = < x_1, b_1 >, \ldots, < x_m, b_m >$ from X is a finite sequence of labeled examples from X. If C is a representation class, a *labeled example of* $c \in C$ is a labeled example $< x, c(x) >$, where $x \in X$. A *labeled sample of* c is a labeled sample S where each example of S is a labeled example of c. In the case where all labels b_i or $c(x_i)$ are 1 (0, respectively), we may omit the labels and simply write S as

a list of points $x_1, \ldots, x_m$, and we call the sample a *positive* (*negative*, respectively) sample.

We say that a representation h and an example $< x, b >$ *agree* if $h(x) = b$; otherwise they *disagree*. We say that a representation h and a sample S are *consistent* if h agrees with each example in S; otherwise they are *inconsistent*.

2.2 Distribution-free learning

Distributions on examples. On any given execution, a learning algorithm for a representation class C will be receiving examples of a single distinguished representation $c \in C$. We call this distinguished c the *target representation*. Examples of the target representation are generated probabilistically as follows: let D_c^+ be a fixed but arbitrary probability distribution over $pos(c)$, and let D_c^- be a fixed but arbitrary probability distribution over $neg(c)$. We call these distributions the *target distributions*. When learning c, learning algorithms will be given access to two oracles, POS and NEG, that behave as follows: oracle POS (NEG, respectively) returns in unit time a positive (negative, respectively) example of the target representation, drawn randomly according to the target distribution D_c^+ (D_c^-, respectively).

The distribution-free model is sometimes defined in the literature with a single target distribution over the entire domain; the learning algorithm is then given labeled examples of the target concept drawn from this distribution. We choose to explicitly separate the distributions over the positive and negative examples to facilitate the study of algorithms that learn using only positive examples or only negative examples. These models, however, are equivalent with respect to polynomial-time computation, as is shown by Haussler et al. [51].

We think of the target distributions as representing the "real world" distribution of objects in the environment in which the learning algorithm must perform; these distributions are separate from, and in the informal sense, independent from the underlying target representation. For instance, suppose that the target concept were that of "life-threatening situations". Certainly the situations "oncoming tiger" and "oncoming

truck" are both positive examples of this concept. However, a child growing up in a jungle is much more likely to witness the former event than the latter, and the situation is reversed for a child growing up in an urban environment. These differences in probability are reflected in *different* target distributions for the *same* underlying target concept. Furthermore, since we rarely expect to have precise knowledge of the target distributions at the time we design a learning algorithm (and in particular, since the usually studied distributions such as the uniform and normal distributions are typically quite unrealistic to assume), ideally we seek algorithms that perform well under *any* target distributions. This apparently difficult goal will be moderated by the fact that the hypothesis of a learning algorithm will be required to perform well only against the distributions on which the algorithm was trained.

Given a fixed target representation $c \in C$, and given fixed target distributions D_c^+ and D_c^-, there is a natural measure of the *error* (with respect to c, D_c^+ and D_c^-) of a representation h from a representation class H. We define $e_c^+(h) = D_c^+(neg(h))$ (i.e., the weight of the set $neg(h)$ under the probability distribution D_c^+) and $e_c^-(h) = D_c^-(pos(h))$ (the weight of the set $pos(h)$ under the probability distribution D_c^-). Note that $e_c^+(h)$ (respectively, $e_c^-(h)$) is simply the probability that a random positive (respectively, negative) example of c is identified as negative (respectively, positive) by h. If both $e_c^+(h) < \epsilon$ and $e_c^-(h) < \epsilon$, then we say that h is an *ϵ-good* hypothesis (with respect to c, D_c^+ and D_c^-); otherwise, h is *ϵ-bad*. We define the *accuracy* of h to be the value $\min(1 - e_c^+(h), 1 - e_c^-(h))$.

It is worth noting that our definitions so far assume that the hypothesis h is deterministic. However, this need not be the case; for example, we can instead it define $e_c^+(h)$ to be the probability that h classifies a random positive example of c as negative, where the probability is now over both the random example and the coin flips of h. All of the results presented here hold under these generalized definitions.

When the target representation c is clear from the context, we will drop the subscript c and simply write D^+, D^-, e^+ and e^-.

In the definitions that follow, we will demand that a learning algorithm produce with high proability an ϵ-good hypothesis regardless of the target representation and target distributions. While at first this may seem like a strong criterion, note that the error of the hypothesis output is always measured with respect to the same target distributions on which

the algorithm was trained. Thus, while it is true that certain examples of the target representation may be extremely unlikely to be generated in the training process, these same examples intuitively may be "ignored" by the hypothesis of the learning algorithm, since they contribute a negligible amount of error. Continuing our informal example, the child living in the jungle may never be shown an oncoming truck as an example of a life-threatening situation, but provided he remains in the environment in which he was trained, it is unlikely that his inability to recognize this danger will ever become apparent. Regarding this child as the learning algorithm, the distribution-free model would demand that if the child were to move to the city, he quickly would "re-learn" the concept of life-threatening situations in this new environment (represented by new target distributions), and thus recognize oncoming trucks as a potential danger. This versatility and generality in learning seem to agree with human experience.

Learnability. Let C and H be representation classes over X. Then C is *learnable from examples by* H if there is a (probabilistic) algorithm A with access to POS and NEG, taking inputs ϵ, δ, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any inputs $0 < \epsilon, \delta < 1$, algorithm A halts and outputs a representation $h_A \in H$ that with probability greater than $1 - \delta$ satisfies $e^+(h_A) < \epsilon$ and $e^-(h_A) < \epsilon$.

We call C the *target class* and H the *hypothesis class*; the output $h_A \in H$ is called the *hypothesis* of A. A will be called a *learning algorithm* for C. If C and H are polynomially evaluatable, and A runs in time polynomial in $1/\epsilon, 1/\delta$ and $|c|$ then we say that C is *polynomially learnable from examples by* H; if C is parameterized we also allow the running time of A to have polynomial dependence on the parameter n.

Allowing the learning algorithm to have a time dependence on the representation size $|c|$ can potentially serve two purposes: first, it lets us discuss the polynomial-time learnability of parameterized classes containing representations whose length is super-polynomial in the parameter n (such as the class of all DNF formulae) in a meaningful way. In general, however, when studying parameterized Boolean representation classes, we will instead place an *explicit* polynomial length bound on the representations in C_n for clarity; thus, we will study classes such as all DNF formulae in which the formula length is bounded by some

polynomial in the total number of variables. Such a restriction makes polynomial dependence on both $|c|$ and n redundant, and thus we may simply consider polynomial dependence on the complexity parameter n. The second use of the dependence on $|c|$ is to allow more refined complexity statements for those representation classes which already have a polynomial length bound. Thus, for example, every conjunction over n Boolean variables has length at most n, but we may wish to consider the time or number of examples required when only $s << n$ variables are present in the target conjunction. This second use is one that we will occasionally take advantage of.

We will drop the phrase "from examples" and simply say that C is *learnable by* H, and C is *polynomially learnable by* H. We say C is *polynomially learnable* to mean that C is polynomially learnable by H for some polynomially evaluatable H. We will sometimes call ϵ the *accuracy parameter* and δ the *confidence parameter*.

Thus, we ask that for any target representation and any target distributions, a learning algorithm finds an ϵ-good hypothesis with probability at least $1 - \delta$. A primary goal of research in this model is to discover which representation classes C are polynomially learnable.

Note that in the above definitions, we allow the learning algorithm to output hypotheses from some class H that is possibly different from C, as opposed to the natural choice $C = H$. While in general we assume that H is at least as powerful as C (that is, $C \subseteq H$), we will see that in some cases for computational reasons we may not wish to restrict H beyond it being polynomially evaluatable. If the algorithm produces an accurate and easily evaluated hypothesis, then our learning problem is essentially solved, and the actual form of the hypothesis is of secondary concern. A major theme of this book is the importance of allowing a wide choice of representations for a learning algorithm.

We refer to Valiant's model as the *distribution-free* model, to emphasize that we seek algorithms that work for any target distributions. It is also known in the literature as the *probably approximately correct* model. We also occasionally refer to the model as that of *strong learnability*, in contrast with the notion of *weak learnability* defined below.

Weak learnability. We will also consider a distribution-free model in which the hypothesis of the learning algorithm is required to perform only

slightly better than random guessing.

Let C and H be representation classes over X. Then C is *weakly learnable from examples by H* if there is a polynomial p and a (probabilistic) algorithm A with access to POS and NEG, taking input δ, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any input value $0 < \delta < 1$, algorithm A halts and outputs a representation $h_A \in H$ that with probability greater than $1 - \delta$ satisfies $e^+(h_A) < 1/2 - 1/p(|c|)$ and $e^-(h_A) < 1/2 - 1/p(|c|)$.

Thus, the accuracy of h_A must be at least $1/2 + 1/p(|c|)$. A will be called a *weak learning* algorithm for C. If C and H are polynomially evaluatable, and A runs in time polynomial in $1/\delta$ and $|c|$ we say that C is *polynomially weakly learnable by H* and C is *polynomially weakly learnable* if it is weakly learnable by H for some polynomially evaluatable H. In the case that the target class C is parameterized, we allow the polynomial p and the running time to depend on the parameter n. Again, we will usually explicitly restrict $|c|$ to be polynomial in n, and thus may assume p depends on n alone.

We may intuitively think of weak learning as the ability to detect some slight bias separating positive and negative examples, where the advantage gained over random guessing diminishes as the complexity of the problem grows. Our main use of the weak learning model is in proving the strongest possible hardness results in Chapter 7. We also give a weak learning algorithm for uniform target distributions in Chapter 8, and in Chapter 9 we discuss models equivalent to weak learning. Recently Goldman et al. have investigated the sample size required for weak learning, independent of computation time [43].

Positive-only and negative-only learning algorithms. We will sometimes study learning algorithms that need only positive examples or only negative examples. If A is a learning algorithm for a representation class C, and A makes no calls to the oracle NEG (respectively, POS), then we say that A is a *positive-only* (respectively, *negative-only*) learning algorithm, and C is *learnable from positive examples* (*learnable from negative examples*). Analogous definitions are made for positive-only and negative-only weak learnability. Note that although the learning algorithm receives only one type of examples, the hypothesis output must

still be accurate with respect to *both* the positive and negative distributions.

Several learning algorithms in the distribution-free model are positive-only or negative-only. The study of positive-only and negative-only learning is important for at least two reasons. First, it helps to quantify more precisely what kind of information is required for learning various representation classes. Second, it is crucial for applications where, for instance, negative examples are rare but must be classified accurately when they do occur.

Distribution-specific learnability. The models for learnability described above demand that a learning algorithm work regardless of the distributions on the examples. We will sometimes relax this condition, and consider these models under restricted target distributions, for instance the uniform distribution. Here the definitions are the same as before, except that we ask that the performance criteria for learnability be met only under these restricted target distributions.

2.3 An example of efficient learning

We now illustrate how the distribution-free model works in the very basic case of monomials, which are conjunctions of literals over Boolean variables. Suppose we are interested in a set of Boolean variables describing the animal kingdom. For concreteness, we will give the variables descriptive names, rather than referring to them with abstract symbols such as x_i. The variable set for animals might include variables describing the physical appearance of the animals (such as **is_large, has_claws, has_mane, has_four_legs** and **has_wings**); variables describing various motor skills (such as **can_fly, walks_on_two_legs** and **can_speak**); variables describing the animal's habitat (**is_wild, lives_in_circus**); as well as variables describing more scientific classifications (**is_mammal**), and many others.

We wish to construct a monomial to distinguish lions from non-lions. For the variables mentioned above, an appropriate conjunction might be

c = **is_mammal** *and* **is_large** *and* **has_claws** *and* **has_four_legs**.

In this example, the probability distribution D^+ is interpreted as reflecting the natural world regarding lions. For instance, each of the four variables appearing in c must be true (i.e., assigned the value 1) with probability 1 in D^+; this simply reflects the fact that, for example, *all* lions are mammals. Since we are assuming here that lions can be represented exactly by monomials, it follows that some variables must be true in D^+ with probability 1.

Other variables are true in D^+ with smaller probabilities. We might expect the variable **has_mane** to be true with probability approximately 1/2, if there are roughly equal numbers of male and female lions. Similarly, we expect the variable **walks_on_two_legs** to be true with relatively low probability, and **has_wings** to be true with probability 0.

Notice that there may be dependencies of arbitrary complexity between variables in the distributions. The variable **is_wild** may be true with very high probability in D^+ if most lions live in the wild, but the probability that *both* **is_wild** and **lives_in_circus** are true is 0. A slightly more subtle dependency might be that even though few lions can walk on two legs, almost all of those that live in the circus can walk on two legs.

In an analogous manner, the negative distribution D^- is intended to reflect the examples of non-lions in the animal world, and again there are many dependencies. Animals with wings may comprise only a small fraction of those animals that are not lions, but the probability that an animal with wings can fly is very high (but not 1, due to flightless birds such as penguins). Note that for simplicity, we have chosen an example that is monotone — no variable appears negated in the monomial c. A natural example of nonmonotonicity might be a monomial for female lions, where we would need to include the *negation* of the variable **has_mane**.

Thus, in this domain, a learning algorithm must infer a monomial over the animal variables that performs well as a classifier of lions and non-lions. Note that the meaning of "performs well" is intimately related to the distributions D^+ and D^-. In the distributions described above, it may be that the monomial c is the only good approximation of the concept, depending on the exact probabilities in the distributions, and the value of the error parameter ϵ. However, if the distributions D^+ and D^- give non-zero weight only to animals for which the variable **lives_in_circus** is true, the monomial consisting of the sole variable **has_claws** might suffice to accurately distinguish lions from the

other animals, if there are very few clawed animals in the circus besides the lions. Note that these conjunctive formulae are not intended as Platonic descriptions of categories. The only requirement on the monomials is that they distinguish with sufficient accuracy categories in the real world as specified by D^+ and D^-.

We now describe an algorithm A for learning monomials over n variables with arbitrary distributions D^+ and D^-. The analysis of this algorithm in the distribution-free model is due to Valiant [93]. Although the monomial output by A has error less than ϵ on both distributions, A needs only examples drawn from D^+ in order to learn; thus A is a positive-only algorithm.

The idea behind the algorithm is the following: suppose that the variable x_i appears in the monomial c being learned. Then in a randomly drawn positive example, x_i is *always* assigned the value 1. Thus, if some variable x_j is assigned the value 0 in a positive example, we are certain that x_j does not appear in c, and thus may delete x_j from the current hypothesis. The algorithm A is:

$h_A \leftarrow x_1\overline{x}_1x_2\overline{x}_2 \cdots x_n\overline{x}_n$;
for $i := 1$ **to** m **do**
begin
  $\vec{v} \leftarrow POS$;
  for $j := 1$ **to** n **do**
    if $v_j = 0$ **then**
      delete x_j from h_A;
    else
      delete $\overline{x}_j$ from h_A;
end
output h_A.

Here v_j denotes the jth bit of $\vec{v}$.

How can algorithm A err? Only by failing to delete some variable x_j that does not appear in c. An exact bound on the value of the outer loop counter m such that the error incurred by such failures is larger than ϵ with probability less than δ can be deduced to be $(2n/\epsilon)(\ln 2n + \ln 1/\delta)$ by a rough analysis. Intuitively, if the variable x_j is false in D^+ with probability $\epsilon/2n$ or smaller, then we incur error at most $\epsilon/2n$ on D^+ and zero error on D^- by failing to

delete x_j. The total error incurred on D^+ by all such failures is then at most $(\epsilon/2n)2n = \epsilon$, since there are at most $2n$ literals in all. On the other hand, if x_j is false with probability at least $\epsilon/2n$ in D^+ then we expect to delete x_j within about $2n/\epsilon$ positive examples.

In the case of our lions example, the variables **can_speak**, **can_fly**, and **has_wings** will be deleted from the hypothesis immediately, since no lion can speak or has wings (i.e., every positive example assigns the value 0 to these variables). With high probability, we would also expect the attributes **walks_on_two_legs**, **lives_in_circus**, and **has_mane** to be deleted, because each of these variables is false with some significant probability in the positive examples. Depending on the exact value of ϵ and the precise probabilities in D^+, the variable **is_wild** may also be deleted. However, the four variables appearing in c will certainly not be deleted.

In this example, the two sources of error that a learning algorithm is prone to can be exemplified as follows. First, it is possible that rare midget lions exist but have not occurred in the training set of examples. In other words, the attribute **is_large** should have been deleted from the hypothesis monomial, but has not been. This is not serious, since the learned monomial will only misclassify future examples that are infrequent in D^+. Second, it is possible that the randomly drawn training set contained a very unrepresentative set of lions, all of which can walk on two legs. In this case the learned monomial will include this variable, and hence misclassify many future examples. While there is no ultimate defense against either of these two kinds of error, the distribution-free model allows the probabilities of their occurrence to be controlled by the parameters ϵ and δ respectively.

2.4 Other definitions and notation

Sample complexity. Let A be a learning algorithm for a representation class C. Then we denote by $S_A(\epsilon, \delta)$ the number of calls to the oracles POS and NEG made by A on inputs ϵ, δ; this is a worst-case measure over all possible target representations in C and all target distributions D^+ and D^-. In the case that C is a parameterized representation class, we also allow S_A to depend on the parameter n. We call the function

S_A the *sample complexity* or *sample size* of A. We denote by S_A^+ and S_A^- the number of calls of A to POS and NEG, respectively.

Chernoff bounds. We shall make extensive use of the following bounds on the area under the tails of the binomial distribution. For $0 \leq p \leq 1$ and m a positive integer, let $LE(p, m, r)$ denote the probability of at most r successes in m independent trials of a Bernoulli variable with probability of success p, and let $GE(p, m, r)$ denote the probability of at least r successes. Then for $0 \leq \alpha \leq 1$,

Fact CB1. $LE(p, m, (1-\alpha)mp) \leq e^{-\alpha^2 mp/2}$

and

Fact CB2. $GE(p, m, (1+\alpha)mp) \leq e^{-\alpha^2 mp/3}$

These bounds in the form they are stated are from the paper of Angluin and Valiant [14]; see also Chernoff [28]. Although we will make frequent use of Fact CB1 and Fact CB2, we will do so in varying levels of detail, depending on the complexity of the calculation involved. However, we are primarily interested in Chernoff bounds for the following consequence of Fact CB1 and Fact CB2: given an event E of probability p, we can obtain an estimate $\hat{p}$ of p by drawing m points from the distribution and letting $\hat{p}$ be the frequency with which E occurs in this sample. Then for m polynomial in $1/p$ and $1/\alpha$, $\hat{p}$ satisfies $p/2 < \hat{p} < 2p$ with probability at least $1-\alpha$. If we also allow m to depend polynomially on $1/\beta$, we can obtain an estimate $\hat{p}$ such that $p - \beta < \hat{p} < p + \beta$ with probability at least $1-\alpha$.

The Vapnik-Chervonenkis dimension. Let C be a representation class over X. Let $Y \subseteq X$, and define

$$\Pi_C(Y) = \{Z \subseteq Y : Z = Y \cap pos(c) \text{ for some } c \in C\}.$$

If we have $\Pi_C(Y) = 2^Y$, then we say that Y is *shattered* by C. Then we define

$$\text{VCD}(C) = \max\{|Y| : Y \text{ is shattered by } C\}.$$

If this maximum does not exist, then $\text{VCD}(C)$ is infinite. The Vapnik-Chervonenkis was originally introduced in the paper of Vapnik and Chervonenkis [97] and was first studied in the context of the distribution-free model by Blumer et al. [25]. Our main use of the Vapnik-Chervonenkis dimension will be in Chapter 6.

Notational conventions. Let $E(x)$ be an event and $\psi(x)$ a random variable that depend on a parameter x that takes on values in a set X. Then for $X' \subseteq X$, we denote by $\mathbf{Pr}_{x \in X'}[E(x)]$ the probability that E occurs when x is drawn uniformly at random from X'. Similarly, $\mathbf{E}_{x \in X'}[\psi(x)]$ is the expected value of ψ when x is drawn uniformly at random from X'. We also need to work with distributions other than the uniform distribution; thus if P is a distribution over X we use $\mathbf{Pr}_{x \in P}[E(x)]$ and $\mathbf{E}_{x \in P}[\psi(x)]$ to denote the probability of E and the expected value of ψ, respectively, when x is drawn according to the distribution P. When E or ψ depend on several parameters that are drawn from different distributions we use multiple subscripts. For example, $\mathbf{Pr}_{x_1 \in P_1, x_2 \in P_2, x_3 \in P_3}[E(x_1, x_2, x_3)]$ denotes the probability of event E when x_1 is drawn from distribution P_1, x_2 from P_2, and x_3 from P_3.

2.5 Some representation classes

We now define some of the representation classes whose learnability we will study. For the Boolean circuit or formulae representation classes, the domain X_n is always $\{0,1\}^n$ and the mapping σ simply maps each circuit to its set of satisfying assignments. The classes defined below are all parameterized; for each class we will define the subclasses C_n, and then C is defined by $C = \cup_{n \geq 1} C_n$.

Monomials: The representation class M_n consists of all conjunctions of literals over the Boolean variables $x_1, \ldots, x_n$.

kCNF: For any constant k, the representation class $k\mathrm{CNF}_n$ consists of all Boolean formulae of the form $C_1 \wedge \cdots \wedge C_l$, where each clause C_i is a disjunction of at most k literals over the Boolean variables $x_1, \ldots, x_n$. Note that $M_n = 1\mathrm{CNF}_n$.

kDNF: For any constant k, the representation class $k\mathrm{DNF}_n$ consists of all Boolean formulae of the form $T_1 \vee \cdots \vee T_l$, where each term T_i is a conjunction of at most k literals over the Boolean variables $x_1, \ldots, x_n$.

k-clause CNF: For any constant k, the representation class k-CLAUSE-CNF$_n$ consists of all conjunctions of the form $C_1 \wedge \cdots \wedge C_k$, where each C_i is a disjunction of literals over the Boolean variables $x_1, \ldots, x_n$.

k-term DNF: For any constant k, the representation class k-TERM-DNF$_n$ consists of all disjunctions of the form $T_1 \vee \cdots \vee T_k$, where each T_i is a monomial over the Boolean variables $x_1, \ldots, x_n$.

CNF: The representation class CNF$_n$ consists of all formulae of the form $C_1 \wedge \cdots \wedge C_l$, where each C_i is a disjunction of literals over the Boolean variables $x_1, \ldots, x_n$.

DNF: The representation class DNF$_n$ consists of all formulae of the form $T_1 \vee \cdots \vee T_l$, where each T_i is a disjunction of literals over the Boolean variables $x_1, \ldots, x_n$.

Boolean Formulae: The representation class BF$_n$ consists of all Boolean formulae over the Boolean variables $x_1, \ldots, x_n$.

Boolean Threshold Functions: A Boolean threshold function over the Boolean variables $x_1, \ldots, x_n$ is defined by a pair (Y, l), where $Y \subseteq \{x_1, \ldots, x_n\}$ and $0 \leq l \leq n$. A point $\vec{v} \in \{0,1\}^n$ is a positive example if and only if at least l of the bits in Y are set to 1 in $\vec{v}$. We let BTF$_n$ denote the class of all such representations.

Symmetric Functions: A *symmetric function* over the Boolean variables $x_1, \ldots, x_n$ is a Boolean function whose output is invariant under all permutations of the input bits. Such a function can be represented by a Boolean array of size $n+1$, where the ith entry indicates whether the function is 0 or 1 on all inputs with exactly i bits set to 1. We denote by SF$_n$ the class of all such representations.

Decision Lists: A *decision list* [84] is a list $L = < (T_1, b_1), \ldots, (T_l, b_l) >$, where each T_i is a monomial over the Boolean variables $x_1, \ldots, x_n$ and each $b_i \in \{0,1\}$. For $\vec{v} \in \{0,1\}^n$, we define $L(\vec{v})$ as follows: $L(\vec{v}) = b_j$ where $1 \leq j \leq l$ is the least value such that $\vec{v}$ satisfies the monomial T_j; if there is no such j then $L(\vec{v}) = 0$. We denote the class of all such representations by DL$_n$. For any constant k, if each monomial T_i has at most k literals, then we have a *k-decision list*, and we denote the class of all such representations by kDL$_n$.

Decision Trees: A *decision tree* over Boolean variables $x_1, \ldots, x_n$ is a binary tree with labels chosen from $\{x_1, \ldots, x_n\}$ on the internal nodes, and labels from $\{0, 1\}$ on the leaves. Each internal node's left branch is viewed as the 0-branch; the right branch is the 1-branch. Then a value $\vec{v} \in \{0,1\}^n$ defines a path in a decision tree T as follows: if an internal node is labeled with x_i, then we follow the 0-branch of that node if $v_i = 0$, otherwise we follow the 1-branch. $T(\vec{v})$ is then defined to be the label of the leaf that is reached on this path. We denote the class of all such representations by DT_n.

Boolean Circuits: The representation class CKT_n consists of all Boolean circuits over input variables $x_1, \ldots, x_n$.

Threshold Circuits: A *threshold gate* over input variables $x_1, \ldots, x_n$ is defined by a value $1 \leq t \leq n$ such that the gate outputs 1 if and only if at least t of the input bits are set to 1. We let TC_n denote the class of all circuits of threshold gates over $x_1, \ldots, x_n$. For constant d, $d\mathrm{TC}_n$ denotes the class of all threshold circuits in TC_n with depth at most d.

Acyclic Finite Automata: The representation class ADFA_n consists of all deterministic finite automata that accept only strings of length n, that is, all deterministic finite automata M such that the language $L(M)$ accepted by M satisfies $L(M) \subseteq \{0,1\}^n$.

We will also consider the following representation classes over Euclidean space R^n.

Linear Separators (Half-spaces): Consider the class consisting of all half-spaces (either open or closed) in R^n, represented by the $n+1$ coefficients of the separating hyperplane. We denote by LS_n the class of all such representations.

Axis-parallel Rectangles: An axis-parallel rectangle in R^n is the cross product of n open or closed intervals, one on each coordinate axis. Such a rectangle could be represented by a list of the interval endpoints. We denote by APR_n the class of all such representations.

3

Recent Research in Computational Learning Theory

In this chapter we give an overview of some recent results in the distribution-free learning model, and in related models. We begin by discussing some of the basic learning algorithms and hardness results that have been discovered. We then summarize results that give sufficient conditions for learnability via the Vapnik-Chervonenkis dimension and Occam's Razor. We conclude the chapter with a discussion of extensions and restrictions of the distribution-free model that have been considered in the literature. Where it is relevant to results presented here, we will also discuss other previous research in greater detail throughout the text.

The summary provided here is far from exhaustive; for a more detailed sampling of recent research in computational learning theory, we refer the reader to the *Proceedings of the Workshop on Computational Learning Theory* [53, 85, 38].

3.1 Efficient learning algorithms and hardness results

In his initial paper defining the distribution-free model [93], Valiant also gives the first polynomial-time learning algorithms in this model. Analyzing the algorithm discussed in the example of Section 2.3, he shows that the class of

monomials is polynomially learnable, and extends this algorithm to prove that for any fixed k, the classes kCNF and kDNF are polynomially learnable (with time complexity $O(n^k)$). For each of these algorithms, the hypothesis class is the same as the target class; that is, in each case C is polynomially learnable by C.

Pitt and Valiant [78] subsequently observe that the representation classes represented by k-TERM-DNF and k-CLAUSE-CNF are properly contained within the classes kCNF and kDNF, respectively. Combined with the results of Valiant [93], this shows that for fixed k, the class k-TERM-DNF is polynomially learnable by kCNF, and the class k-CLAUSE-CNF is polynomially learnable by kDNF. More surprising, Pitt and Valiant prove that for any fixed $k \geq 2$, learning k-TERM-DNF by k-TERM-DNF and learning k-CLAUSE-CNF by k-CLAUSE-CNF are *NP*-hard problems.

The results of Pitt and Valiant are important in that they demonstrate the tremendous computational advantage that may be gained by a judicious change of hypothesis representation. This can be viewed as a limited but provable confirmation of the rule of thumb in artificial intelligence that *representation is important.* By moving to a more powerful hypothesis class H instead of insisting on the more "natural" choice $H = C$, we move from an *NP*-hard problem to a polynomial-time solution. This may be explained intuitively by the observation that while the constraint $H = C$ may be significant enough to render the learning task intractable, a richer hypothesis representation allows a greater latitude for expressing the learned formula. Later we shall see that using a larger hypothesis class inevitably requires a larger sample complexity; thus the designer of a learning algorithm may sometimes be faced with a trade-off between computation time and required sample size. We will return to the subject of hardness results for learning momentarily.

Other positive results for polynomial-time learning include the algorithm of Haussler [48] for learning the class of *internal disjunctive* Boolean formulae. His algorithm is notable for the fact that the time complexity depends linearly on the size of the target formula, but only logarithmically on the total number of variables n; thus if there are many "irrelevant" attributes, the time required will be quite modest. This demonstrates that there need not be explicit *focusing* mechanisms in the definitions of the distribution-free model for identifying those variables which are relevant for a learning algorithm, but rather this task can be incorporated into the algorithms themselves. Similar results are

given for linearly separable classes by Littlestone [73], and recently a model of learning in the presence of *infinitely* many irrelevant attributes was proposed by Blum [20].

Rivest [84] considers k-decision lists, and gives a polynomial-time algorithm learning kDL by kDL for any constant k. He also proves that kDL properly includes both kCNF and kDNF. Ehrenfeucht and Haussler [35] study decision trees. They define a measure of how balanced a decision tree is called the *rank*. For decision trees of a fixed rank r, they give a polynomial-time recursive learning algorithm that always outputs a rank r decision tree. They also note that k-decision lists are decision trees of rank 1, if we allow conjunctions of length k in the nodes of the decision tree. Ehrenfeucht and Haussler apply their results to show that for any fixed polynomial $p(n)$, decision trees with at most $p(n)$ nodes can be learned in time linear in $n^{O(\log n)}$, $1/\epsilon$ and $\log 1/\delta$, thus giving a super-polynomial but sub-exponential time solution.

Abe [1] gives a polynomial-time algorithm for learning a class of formal languages known as *semi-linear sets*. Helmbold, Sloan and Warmuth [55] give techniques for learning *nested differences* of classes already known to be polynomially learnable. These include classes such as the class of all subsets of Z^k closed under addition and subtraction and the class of nested differences of rectangles in the plane. Some of their results extend the composition methods given in Chapter 4.

There are many efficient algorithms that learn representation classes defined over Euclidean (real-valued) domains. Most of these are based on the pioneering work of Blumer, Ehrenfeucht, Haussler and Warmuth [25] on learning and the Vapnik-Chervonenkis dimension, which will be discussed in greater detail later. These algorithms show the polynomial learnability of, among others, the class of all rectangles in n-dimensional space, and the intersection of n half-planes in 2-dimensional space.

We now return to our discussion of hardness results. In discussing hardness results, we distinguish between two types: *representation-based* hardness results and *representation-independent* hardness results. Briefly, representation-based hardness results state that for some *fixed* representation classes C and H, learning C by H is hard in some computational sense (such as NP-hardness). Thus, the aforementioned result of Pitt and Valiant [78] on the difficulty of learning k-TERM-DNF by k-TERM-DNF is representation-based. In contrast,

a representation-independent hardness result says that for fixed C and *any* polynomially evaluatable H, learning C by H is hard.

Representation-based hardness results are interesting for a number of reasons, two of which we have already mentioned: they can be used to give formal verification to the importance of hypothesis representation, and for practical reasons it is important to study the *least* expressive class H that can be used to learn C, since the choice of hypothesis representation can greatly affect resource complexity (such as the number of examples required) even for those classes already known to be polynomially learnable.

However, since a representation-based hardness result dismisses the polynomial learnability of C only with respect to the *fixed* hypothesis class H, such results leave something to be desired in the quest to classify learning problems as "easy" or "hard". For example, we may be perfectly willing to settle for an efficient algorithm learning C by H for some more expressive H if we know that learning C by C is NP-hard. Thus for practical purposes we must regard the polynomial learnability of C as being unresolved until we either find an efficient learning algorithm or we prove that learning C by H is hard for *any* reasonable H, that is, until we prove a representation-independent hardness result for C.

Gold [41] gave the first representation-based hardness results that apply to the distribution-free model of learning. He proves that the problem of finding the smallest deterministic finite automaton consistent with a given sample is NP-complete; the results of Haussler et al. [51] can be easily applied to Gold's result to prove that learning deterministic finite automata of size n by deterministic finite automata of size n cannot be accomplished in polynomial time unless $RP = NP$. There are some technical issues involved in properly defining the problem of learning finite automata in the distribution-free model; see Pitt and Warmuth [79] for details. Gold's results were improved by Li and Vazirani [69], who show that finding an automaton $9/8$ larger than the smallest consistent automaton is still NP-complete.

As we have already discussed, Pitt and Valiant [78] prove that for $k \geq 2$, learning k-TERM-DNF by k-TERM-DNF is NP-hard by giving a randomized reduction from a generalization of the graph coloring problem. Even stronger, for $k \geq 6$, they prove that even if the hypothesis DNF formulae is allowed to have $2k-3$ terms, k-TERM-DNF cannot be learned in polynomial time unless

$RP = NP$. These results hold even when the target formulae are restricted to be monotone and the hypothesis formulae is allowed to be nonmonotone. Dual results hold for the problem of learning k-CLAUSE-CNF. Pitt and Valiant also prove that μ-formulae (Boolean formulae in which each variable occurs at most once, sometimes called *read-once*) cannot be learned by μ-formulae in polynomial time, and that Boolean threshold functions cannot be learned by Boolean threshold functions in polynomial time, unless $RP = NP$.

Pitt and Valiant [78] also give representation-based hardness results for a model called *heuristic learnability*. Here the hypothesis class may actually be *less* expressive than the target class; the conditions imposed on the hypothesis are weakened accordingly. In this model they prove that the problem of finding a monomial that has error at most ϵ with respect to the negative target distribution of a target DNF formulae and error at most $1 - c$ with respect to the positive target distribution (provided such a monomial exists) is NP-hard with respect to randomized reductions, for any constant $0 < c < 1$. They prove a similar result regarding the problem of finding an hypothesis μ-formulae that has negative error 0 and positive error at most $1 - e^{-n^3}$ on the distributions for a target μ-formulae.

Pitt and Warmuth [80] dramatically improved the results of Gold by proving that deterministic finite automata of size n cannot be learned in polynomial time by deterministic finite automata of size n^α for any fixed value $\alpha \geq 1$ unless $RP = NP$. Their results leave open the possibility of an efficient learning algorithm using deterministic finite automata whose size depends on ϵ and δ, or an algorithm using some entirely different representation of the sets accepted by automata. This possibility is addressed by the results in Chapter 7.

Hancock [46] has shown that learning decision trees of size n by decision trees of size n cannot be done in polynomial time unless $RP = NP$. Representation-based hardness results for learning various classes of neural networks can also be derived from the results of Judd [57] and Blum and Rivest [22].

The first representation-independent hardness results for the distribution-free model follow from the work of Goldreich, Goldwasser and Micali [45], whose true motivation was to find easy-to-compute functions whose output on random inputs appears random to all polynomial-time algorithms. A simplified and weakened statement of their result is that the class of polynomial-size

Boolean circuits is not polynomially learnable by *any* polynomially evaluatable H, provided that there exists a one-way function (see Yao [102]). Pitt and Warmuth [79] defined a general notion of reducibility for learning (discussed further in Section 3.2) and gave a number of other representation classes that are not polynomially learnable under the same assumption by giving reductions from the learning problem for polynomial-size circuits. One of the main contributions of the research presented here is representation-independent hardness results for much simpler classes than those addressed by Goldreich et al. [45] or Pitt and Warmuth [79], among them the classes of Boolean formulae, acyclic deterministic finite automata and constant-depth threshold circuits.

3.2 Characterizations of learnable classes

Determining whether a representation class is polynomially learnable is in some sense a two-step process. We first must determine if a polynomial number of *examples* will even suffice (in an information-theoretic sense) to specify a good hypothesis with high probability. Once we determine that a polynomial-size sample is sufficient, we can then turn to the *computational* problem of efficiently inferring a good hypothesis from the small sample. This division of the learning problem into a *sample complexity* component and a *computational complexity* component will influence our thinking throughout the book.

For representation classes over finite discrete domains (such as $\{0,1\}^n$), an important step towards characterizing the polynomially learnable classes was taken by Blumer et al. [24, 25] in their study of *Occam's Razor*. Their result essentially gives an upper bound on the sample size required for learning C by H, and shows that the general technique of finding an hypothesis that is both consistent with the sample drawn and significantly shorter than this sample is sufficient for distribution-free learning. Thus, if one can efficiently perform *data compression* on a random sample, then one can learn efficiently. Since we will appeal to this result frequently in the text, we shall state it here formally as a theorem.

Theorem 3.1 *(Blumer et al. [24, 25]) Let C and H be polynomially evaluatable parameterized Boolean representation classes. Fix $\alpha \geq 1$ and $0 \leq \beta < 1$, and let A be an algorithm that on input a labeled sample S of some $c \in C_n$,*

consisting of m positive examples of c drawn from D^+ and m negative examples of c drawn from D^-, outputs an hypothesis $h_A \in H_n$ that is consistent with S and satisfies $|h_A| \leq n^\alpha m^\beta$. Then A is a learning algorithm for C by H; the sample size required is

$$m = O\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \left(\frac{n^\alpha}{\epsilon}\log\frac{n^\alpha}{\epsilon}\right)^{\frac{1}{1-\beta}}\right).$$

Let $|S| = mn$ denote the number of bits in the sample S. Note that if A instead outputs h_A satisfying $|h_A| \leq n^{\alpha'}|S|^\beta$ for some fixed $\alpha' \geq 1$ and $0 \leq \beta < 1$ then $|h_A| \leq n^{\alpha'}(mn)^\beta = n^{\alpha'+\beta}m^\beta$, so A satisfies the conditon of Theorem 3.1 for $\alpha = \alpha' + \beta$. This formulation of Occam's Razor will be of particular use to us in Section 7.6.

In a paper of Haussler et al. [51], a partial converse of Theorem 3.1 is given: conditions are stated under which the polynomial learnability of a representation class implies a polynomial-time algorithm for the problem of finding an hypothesis representation consistent with an input sample of an unknown target representation. These conditions are obtained by a straightforward generalization of techniques developed by Pitt and Valiant [78]. In almost all the natural cases in finite domains, these conditions as well as those of Theorem 3.1 are met, establishing an important if and only if relation: C is polynomially learnable by H if and only if there is an algorithm finding with high probability hypotheses in H consistent with an input sample generated by a representation in C. Subsequent papers by Board and Pitt [26] and Schapire [90] consider the stronger and philosophically interesting converse of Theorem 3.1 in which one actually uses a polynomial-time learning algorithm not just for finding a consistent hypothesis, but for performing data compression on a sample. Recently generalizations of Occam's Razor to models more complicated than concept learning have been given by Kearns and Schapire [63].

One drawback of Theorem 3.1 is that the hypothesis output by the learning algorithm must have a polynomial-size representation as a string of bits for the result to apply. Thus, it is most appropriate for discrete domains, where instances are specified as finite strings of bits, and does not apply well to representation classes over real-valued domains, where the specification of a single instance may not have any finite representation as a bit string. This led Blumer et al. to seek a *general* characterization of the sample complexity of

learning *any* representation class. They show that the Vapnik-Chervonenkis dimension essentially provides this characterization: namely, at least $\Omega(\text{VCD}(C))$ examples are required for the distribution-free learning of C, and $O(\text{VCD}(C))$ are sufficient for learning (ignoring for the moment the dependence on ϵ and δ), with any algorithm finding a consistent hypothesis in C being a learning algorithm. Thus, the classes that are learnable in *any* amount of time in the distribution-free model are exactly those classes with finite Vapnik-Chervonenkis dimension. These results will be discussed in greater detail in Chapter 6, where we improve the lower bound on sample complexity given by Blumer et al. [25]. Recently many of the ideas contained in the work of Blumer et al. have been greatly generalized by Haussler [50], who applies uniform convergence techniques developed by many authors to determine sample size bounds for relatively unrestricted models of learning.

As we have mentioned, the results of Blumer et al. apply primarily to the *sample* complexity of learning. A step towards characterizing what is *polynomially* learnable was taken by Pitt and Warmuth [79]. They define a natural notion of polynomial-time reducibility between learning problems, analogous to the notion of reducibility in complexity theory and generalizing simple reductions given here in Section 4.3 and by Littlestone [73]. Pitt and Warmuth are able to give partial characterizations of the complexity of learning various representation classes by finding "learning-complete" problems for these representations classes. For example, they prove that if deterministic finite automata are polynomially learnable, then the class of all languages accepted by log-space Turing machines is polynomially learnable.

3.3 Results in related models

A number of restrictions and extensions of the basic model of Valiant have been considered. These modifications are usually proposed either in an attempt to make the model more realistic (e.g., adding noise to the sample data) or to make the learning task easier in cases where distribution-free learning appears difficult. One may also modify the model in order to more closely examine the resources required for learning, such as space complexity.

For instance, for classes for which learning is known to be intractable

in some precise sense or whose polynomial learnability is unresolved, there are a number of learning algorithms whose performance is guaranteed under restricted target distributions. In addition to the results presented here in Chapter 8, the papers of Benedek and Itai [16], Natarajan [76], Kearns and Pitt [62] and Li and Vitanyi [70] also consider distribution-specific learnability. Recently Linial, Mansour and Nisan [71] applied Fourier transform methods to obtain the first sub-exponential time algorithm for learning DNF under uniform distributions.

Instead of restricting the target distributions to make learning easier, we can also add additional information about the target representation in the form of *queries*. For example, it is natural to allow a learning algorithm to make *membership* queries, that is, to ask for the value of $c(x)$ of the target representation $c \in C$ on points $x \in X$ of the algorithm's choosing. There are a number of interesting query results in Valiant's original paper [93], as well as a series of excellent articles giving algorithms and hardness results by Angluin [7, 8, 9]. Results of Angluin have recently been improved by Rivest and Schapire [86, 87], who consider the problem of inferring a finite automaton with active but non-reversible experimentation. Berman and Roos [18] give an algorithm for learning "one-counter" languages with membership queries. Recently Angluin, Hellerstein and Karpinski [11] gave an algorithm for efficiently learning "read-once" Boolean formulae (i.e., μBF) using membership queries; a large subclass of these can be learned using *non-adaptive* membership queries (where all queries are chosen before any are answered) by the results of Goldman, Kearns and Schapire [42]. If in addition to membership queries we allow *equivalence queries* (where the algorithm is provided with counterexamples to conjectured hypotheses), then there are efficient learning algorithms for Horn sentences due to Angluin, Frazier and Pitt [10] and restricted types of decision trees due to Hancock [47].

Towards the goal of making the distribution-free model more realistic, there are many results now on learning with noise that will be discussed in Chapter 5. Haussler [50] generalizes the model to the problem of learning a function that performs well even in the absence of any assumptions on how the examples are generated; in particular, the examples (x, y) (where y may now be more complicated than a simple $\{0, 1\}$ classification) may be such that y has no prescribed functional dependence on x. Kearns and Schapire [63] apply Haussler's very general but non-computational results to the specific prob-

lem of efficiently learning *probabilistic concepts*, in which examples have some probability of being positive and some probability of being negative, but this uncertainty has some structure that may be exploited by a learning algorithm. Such a framework is intended to model situations such as weather prediction, in which "hidden variables" may result in apparently probabilistic behavior, yet meaningful predictions can often be made. Blum [20] defines a model in which there may be infinitely many attributes in the domain, but short lists of attributes suffice to describe most common objects.

There have also been a number of different models of learnability proposed recently that share the common emphasis on computational complexity. Among these are the *mistake counting* or *on-line* models. Here each example (chosen either randomly, as in Valiant's model, or perhaps by some other method) is presented unlabeled (that is, with no indication as to whether it is positive or negative) to the learning algorithm. The algorithm must then make a guess or *prediction* of the label of the example, only after which is it told the correct label. Two measures of performance in these models are the *expected* number of mistakes of prediction (in the case where examples are generated probabilistically) and the *absolute* number of mistakes (in the case where the examples are generated by deterministic means or by an adversary). These models were defined by Haussler, Littlestone and Warmuth [73, 52]. Particularly notable is the algorithm of Littlestone [73] which learns the class of linearly separable Boolean functions with a small absolute mistake bound. Other recent papers [52, 51] also include results relating the mistake-counting models to the distribution-free model. Littlestone's paper relates the mistake-counting models to a model of equivalence queries. Other on-line learning algorithms are given by Littlestone and Warmuth [74], who consider a *weighted majority* method of learning, and Goldman, Rivest and Schapire [44], who investigate the varying effects of letting the learner, a teacher, and an adversary choose the sequence of examples.

Other interesting extensions to Valiant's basic model include the work of Linial, Mansour and Rivest [72], who consider a model of "dynamic sampling" (see also Haussler et al. [51]), and Rivest and Sloan [89], who consider a model of "reliable and useful" learning that allows a learning algorithm to draw upon a library of previously learned representations. Interesting resource complexity studies of distribution-free learning include research on learning in parallel models of computation due to Vitter and Lin [99] and Berger, Shor

and Rompel [17], and investigations of the space complexity of learning due to Floyd [37] and Schapire [90].

The curious reader should be warned that there are several variants of the basic distribution-free model in the literature, each with its own technical advantages. In response to the growing confusion resulting from this proliferation of models, Haussler et al. [51] show that almost all of these variants are in fact equivalent with respect to polynomial-time computation. This allows researchers to work within the definitions that are most convenient for the problems at hand, and frees the reader from undue concern that the results are sensitive to the small details of the model. Related equivalences are given here in Chapter 9 and by Schapire [90].

4

Tools for Distribution-free Learning

4.1 Introduction

In this chapter we describe some general tools for constructing efficient learning algorithms and for relating the difficulty of learning one representation class to that of learning other representation classes. In Section 4.2, we show that under certain conditions it is possible to construct new learning algorithms for representation classes that can be appropriately decomposed into classes for which efficient learning algorithms already exist. These new algorithms use the existing algorithms as black box subroutines, and thus are a demonstration of how systems that learn may successfully build upon knowledge already acquired. Similar issues have been investigated from a different angle by Rivest and Sloan [89].

In the Section 4.3, we introduce a simple notion of *reducibility* for Boolean circuit learning problems. These efficient reductions work by creating new variables whose addition allows the target representation to be expressed more simply than with the original variables. Thus we see that the presence of "relevant subconcepts" may make the learning problem simpler from a computational standpoint or from our standpoint as researchers. Reducibility allows us to show that learning representation class C_1 is just as hard as learning C_2, and thus plays a role analogous to polynomial-time reductions in complexity theory. A general notion of reducibility and a complexity-theoretic framework for learning have subsequently been developed by Pitt and Warmuth [79].

Although we are primarily interested here in polynomial-time learnability, the results presented in this chapter are easily generalized to higher time complexities.

4.2 Composing learning algorithms to obtain new algorithms

Suppose that C_1 is polynomially learnable by H_1, and C_2 is polynomially learnable by H_2. Then it is easy to see that the class $C_1 \cup C_2$ is polynomially learnable by $H_1 \cup H_2$: we first assume that the target representation c is in the class C_1 and run algorithm A_1 for learning C_1. We then test the hypothesis h_1 output by A_1 on a polynomial-size random sample of c to determine with high probability if it is ϵ-good (this can be done efficiently using Fact CB1 and Fact CB2). If h_1 is ϵ-good, we halt; otherwise, we run algorithm A_2 for learning C_2 and use the hypothesis h_2 output by A_2. This algorithm demonstrates one way in which existing learning algorithms can be composed to learn more powerful representation classes, and it generalizes to any polynomial number of unions of polynomially learnable classes. Are there more interesting ways to compose learning algorithms, possibly learning classes more complicated than simple unions?

In this section we describe techniques for composing existing learning algorithms to obtain new learning algorithms for representation classes that are formed by combining members of the (already) learnable classes with logical operations. In contrast to the case of simple unions, the members of the resulting composite class are not members of any of the original classes. Thus, rather than simply increasing the *size* of the learnable class, we are actually "bootstrapping" (using the terminology of Helmbold, Sloan and Warmuth [55]) the existing algorithms in order to learn a new *type* of representation.

We apply the results to obtain polynomial-time learning algorithms for two classes of Boolean formulae not previously known to be polynomially learnable. Recently in Helmbold et al. [55] a general composition technique has been proposed and carefully analyzed in several models of learnability.

If $c_1 \in C_1$ and $c_2 \in C_2$ are representations, the concept defined by the

representation $c_1 \vee c_2$ is given by $pos(c_1 \vee c_2) = pos(c_1) \cup pos(c_2)$. Note that $c_1 \vee c_2$ may not be an element of either C_1 or C_2. Similarly, $pos(c_1 \wedge c_2) = pos(c_1) \cap pos(c_2)$. We then define $C_1 \vee C_2 = \{c_1 \vee c_2 : c_1 \in C_1, c_2 \in C_2\}$ and $C_1 \wedge C_2 = \{c_1 \wedge c_2 : c_1 \in C_1, c_2 \in C_2\}$.

Theorem 4.1 *Let C_1 be polynomially learnable by H_1, and let C_2 be polynomially learnable by H_2 from negative examples. Then $C_1 \vee C_2$ is polynomially learnable by $H_1 \vee H_2$.*

Proof: Let A_1 be a polynomial-time algorithm for learning C_1 by H_1, and A_2 a polynomial-time negative-only algorithm for learning C_2 by H_2. We describe a polynomial-time algorithm A for learning $C_1 \vee C_2$ by $H_1 \vee H_2$ that uses A_1 and A_2 as subroutines.

Let $c = c_1 \vee c_2$ be the target representation in $C_1 \vee C_2$, where $c_1 \in C_1$ and $c_2 \in C_2$, and let D^+ and D^- be the target distributions on $pos(c)$ and $neg(c)$, respectively. Let S_{A_1} be the number of examples needed by algorithm A_1.

Since $neg(c) \subseteq neg(c_2)$, the distribution D^- may be regarded as a distribution on $neg(c_2)$, with $D^-(x) = 0$ for $x \in neg(c_2) - neg(c)$. Thus A first runs the negative-only algorithm A_2 to obtain a representation $h_2 \in H_2$ for c_2, using the examples generated from D^- by NEG. This simulation is done with accuracy parameter ϵ/kS_{A_1} and confidence parameter $\delta/5$, where k is a constant that can be determined by applying Fact CB1 and Fact CB2 in the analysis below. A_2 then outputs an $h_2 \in H_2$ satisfying with high probability $e^-(h_2) < \epsilon/kS_{A_1}$. Note that although we are unable to bound $e^+(h_2)$ directly (because D^+ is not a distribution over $pos(c_2)$), the fact that the simulation of the negative-only algorithm A_2 must work for any target distribution on $pos(c_2)$ implies that h_2 must satisfy with high probability

$$\begin{aligned} &\mathbf{Pr}_{x \in D^+}[x \in neg(h_2) \text{ and } x \in pos(c_2)] \\ &\quad \leq \mathbf{Pr}_{x \in D^+}[x \in neg(h_2) | x \in pos(c_2)] \\ &\quad < \frac{\epsilon}{kS_{A_1}}. \end{aligned} \tag{4.1}$$

A next attempts to determine if $e^+(h_2) < \epsilon$. A takes $O(1/\epsilon \ln 1/\delta)$ examples from POS and uses these examples to compute an estimate $\hat{p}$ for the value of

$e^+(h_2)$. Using Fact CB1 it can be shown that if $e^+(h_2) \geq \epsilon$, then with high probability $\hat{p} > \epsilon/2$. Using Fact CB2 it can be shown that if $e^+(h_2) \leq \epsilon/4$, then with high probability $\hat{p} \leq \epsilon/2$. Thus, if $\hat{p} \leq \epsilon/2$ then A guesses that $e^+(h_2) \leq \epsilon$. In this case A halts with $h_A = h_2$ as the hypothesis.

On the other hand, if $\hat{p} > \epsilon/2$ then A guesses that $e^+(h_2) \geq \epsilon/4$. In this case A runs A_1 in order to obtain an h_1 that is ϵ-good with respect to D^- and also with respect to that portion of D^+ on which h_2 is wrong. More specifically, A runs A_1 with accuracy parameter ϵ/k and confidence parameter $\delta/5$ according to the following distributions: each time A_1 calls NEG, A supplies A_1 with a negative example of c drawn according to the target distribution D^-; each such example is also a negative example of c_1 since $neg(c) \subseteq neg(c_1)$. Each time A_1 calls POS, A draws from the target distribution D^+ until a point $x \in neg(h_2)$ is obtained. Since the probability of drawing such an x is exactly $e^+(h_2)$, if $e^+(h_2) \geq \epsilon/4$ then the time needed to obtain with high probability S_{A_1} points in $neg(h_2)$ is polynomial in $1/\epsilon$, $1/\delta$ and S_{A_1} by Fact CB1. Now

$$\begin{aligned} &\mathbf{Pr}_{x \in D^+}[x \in neg(h_2) \text{ and } x \in neg(c_1)] \\ &\quad \leq \mathbf{Pr}_{x \in D^+}[x \in neg(h_2) \text{ and } x \in pos(c_2)] \\ &\quad < \frac{\epsilon}{k S_{A_1}} \end{aligned} \tag{4.2}$$

by Inequality 4.1 and the fact that for $x \in pos(c)$, $x \in neg(c_1)$ implies $x \in pos(c_2)$. Since A_1 needs at most S_{A_1} positive examples and h_2 satisfies Inequality 4.2, with high probability all of the positive examples x given to A_1 in this simulation satisfy $x \in pos(c_1)$ for k a large enough constant. Following this simulation, A_1 with high probability outputs h_1 satisfying $e^-(h_1) < \epsilon/k$ and also

$$\begin{aligned} &\mathbf{Pr}_{x \in D^+}[x \in neg(h_1) \text{ and } x \in neg(h_2)] \\ &\quad < \mathbf{Pr}_{x \in D^+}[x \in neg(h_1) | x \in neg(h_2)] \\ &\quad < \frac{\epsilon}{k}. \end{aligned} \tag{4.3}$$

Setting $h_A = h_1 \vee h_2$, we have $e^+(h_A) < \epsilon$ by Inequality 4.3 and $e^-(h_A) < \epsilon$, as desired. Note that the time required by this simulation is polynomial in the time required by A_1 and the time required by A_2. □

The following dual to Theorem 4.1 has a similar proof:

Theorem 4.2 *Let C_1 be polynomially learnable by H_1, and let C_2 be polynomially learnable by H_2 from positive examples. Then $C_1 \wedge C_2$ is polynomially learnable by $H_1 \wedge H_2$.*

As corollaries we have that the following classes of Boolean formulae are polynomially learnable:

Corollary 4.3 *For any fixed k, let $k\text{CNF} \vee k\text{DNF} = \cup_{n \geq 1}(k\text{CNF}_n \vee k\text{DNF}_n)$. Then $k\text{CNF} \vee k\text{DNF}$ is polynomially learnable by $k\text{CNF} \vee k\text{DNF}$.*

Corollary 4.4 *For any fixed k, let $k\text{CNF} \wedge k\text{DNF} = \cup_{n \geq 1}(k\text{CNF}_n \wedge k\text{DNF}_n)$. Then $k\text{CNF} \wedge k\text{DNF}$ is polynomially learnable by $k\text{CNF} \wedge k\text{DNF}$.*

Proofs of Corollaries 4.3 and 4.4 follow from Theorems 4.1 and 4.2 and the algorithms of Valiant [93] for learning kCNF from positive examples and kDNF from negative examples. Note that algorithms obtained in Corollaries 4.3 and 4.4 use both positive and negative examples. Following Theorem 6.1 of Section 6.2 we show that the representation classes $k\text{CNF} \vee k\text{DNF}$ and $k\text{CNF} \wedge k\text{DNF}$ *require* both positive and negative examples for polynomial learnability, regardless of the hypothesis class.

Under the stronger assumption that both C_1 and C_2 are learnable from positive examples, we can prove the following result, which shows that the classes that are polynomially learnable from positive examples are closed under conjunction of representations. A partial converse to this theorem is investigated by Natarajan [76].

Theorem 4.5 *Let C_1 be polynomially learnable by H_1 from positive examples, and let C_2 be polynomially learnable by H_2 from positive examples. Then the class $C_1 \wedge C_2$ is polynomially learnable by $H_1 \wedge H_2$ from positive examples.*

Proof: Let A_1 be a polynomial-time positive-only algorithm for learning C_1 by H_1, and let A_2 be a polynomial-time positive-only algorithm for learning C_2 by H_2. We describe a polynomial-time positive-only algorithm A for learning $C_1 \wedge C_2$ by $H_1 \wedge H_2$ that uses A_1 and A_2 as subroutines.

Let $c = c_1 \wedge c_2$ be the target representation in $C_1 \wedge C_2$, where $c_1 \in C_1$ and $c_2 \in C_2$, and let D^+ and D^- be the target distributions on $pos(c)$ and $neg(c)$. Since $pos(c) \subseteq pos(c_1)$, A can use A_1 to learn a representation $h_1 \in H_1$ for c_1 using the positive examples from D^+ generated by POS. A simulates algorithm A_1 with accuracy parameter $\epsilon/2$ and confidence parameter $\delta/2$, and obtains $h_1 \in H_1$ that with high probability satisfies $e^+(h_1) \leq \epsilon/2$. Note that although we are unable to directly bound $e^-(h_1)$ by $\epsilon/2$, we must have

$$\begin{aligned}
&\mathbf{Pr}_{x \in D^-}[x \in pos(h_1) - pos(c_1)] \\
&\quad = \mathbf{Pr}_{x \in D^-}[x \in pos(h_1) \textit{ and } x \in neg(c_1)] \\
&\quad \leq \mathbf{Pr}_{x \in D^-}[x \in pos(h_1)) | x \in neg(c_1)] \\
&\quad < \frac{\epsilon}{2}
\end{aligned}$$

since A_1 must work for any fixed distribution on $neg(c_1)$. Similarly, A simulates algorithm A_2 with accuracy parameter $\epsilon/2$ and confidence parameter $\delta/2$ to obtain an hypothesis $h_2 \in H_2$ that with high probability satisfies $e^+(h_2) \leq \epsilon/2$ and $\mathbf{Pr}_{x \in D^-}[x \in pos(h_2) - pos(c_2)] \leq \epsilon/2$. Then we have

$$e^+(h_1 \wedge h_2) \leq e^+(h_1) + e^+(h_2) \leq \epsilon.$$

We now bound $e^-(h_1 \wedge h_2)$ as follows:

$$\begin{aligned}
&e^-(h_1 \wedge h_2) \\
&\quad = \mathbf{Pr}_{x \in D^-}[x \in pos(h_1 \wedge h_2) - pos(c_1 \wedge c_2)] \\
&\quad = \mathbf{Pr}_{x \in D^-}[x \in pos(h_1) \cap pos(h_2) \cap neg(c_1 \wedge c_2)] \\
&\quad = \mathbf{Pr}_{x \in D^-}[x \in pos(h_1) \cap pos(h_2) \cap (neg(c_1) \cup neg(c_2))] \\
&\quad = \mathbf{Pr}_{x \in D^-}[x \in (pos(h_1) \cap pos(h_2) \cap neg(c_1)) \cup (pos(h_1) \cap pos(h_2) \cap neg(c_2))] \\
&\quad \leq \mathbf{Pr}_{x \in D^-}[x \in pos(h_1) \cap pos(h_2) \cap neg(c_1)] \\
&\qquad + \mathbf{Pr}_{x \in D^-}[x \in pos(h_1) \cap pos(h_2) \cap neg(c_2)] \\
&\quad \leq \mathbf{Pr}_{x \in D^-}[x \in pos(h_1) \cap neg(c_1)] + \mathbf{Pr}_{x \in D^-}[x \in pos(h_2) \cap neg(c_2)] \\
&\quad = \mathbf{Pr}_{x \in D^-}[x \in pos(h_1) - pos(c_1)] + \mathbf{Pr}_{x \in D^-}[x \in pos(h_2) - pos(c_2)] \\
&\quad \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.
\end{aligned}$$

The time required by this simulation is polynomial in the time taken by A_1 and A_2. □

The proof of Theorem 4.5 generalizes to allow any fixed number k of conjuncts of representations in the target class. Thus, if $C_1, \ldots, C_k$ are polynomially learnable from positive examples by $H_1, \ldots, H_k$ respectively, then the class $C_1 \wedge \cdots \wedge C_k$ is polynomially learnable by $H_1 \wedge \cdots \wedge H_k$ from positive examples. In the case that the component classes are parameterized, we can actually allow k to be any fixed polynomial function of n.

We can also prove the following dual to Theorem 4.5:

Theorem 4.6 *Let C_1 be polynomially learnable by H_1 from negative examples, and let C_2 be polynomially learnable by H_2 from negative examples. Then $C_1 \vee C_2$ is polynomially learnable by $H_1 \vee H_2$ from negative examples.*

Again, if $C_1, \ldots, C_k$ are polynomially learnable from negative examples by $H_1, \ldots, H_k$ respectively, then the class $C_1 \vee \cdots \vee C_k$ is polynomially learnable by $H_1 \vee \cdots \vee H_k$ from negative examples, for any fixed value k (where k may be polynomial in the complexity parameter n).

We can also use Theorems 4.1, 4.2, 4.5 and 4.6 to characterize the conditions under which the class $C_1 \vee C_2$ (respectively, $C_1 \wedge C_2$) is polynomially learnable by $C_1 \vee C_2$ (respectively, $C_1 \wedge C_2$). Figures 4.1 and 4.2 summarize this information, where a "YES" entry indicates that for C_1 and C_2 polynomially learnable as indicated, $C_1 \vee C_2$ (respectively, $C_1 \wedge C_2$) is always polynomially learnable by $C_1 \vee C_2$ (respectively, $C_1 \wedge C_2$), and an entry "*NP*-hard" indicates that the learning problem is *NP*-hard for some choice of C_1 and C_2. All *NP*-hardness results follow from the results of Pitt and Valiant [78].

4.3 Reductions between learning problems

In traditional complexity theory, the notion of polynomial-time reducibility has proven extremely useful for comparing the computational difficulty of problems whose exact complexity or tractability is unresolved. Similarly, in computational learning theory, we might expect that given two representation classes C_1 and C_2 whose polynomial learnability is unresolved, we may still be able to prove conditional statements to the effect that if C_1 is polynomially learnable,

$C_1 \vee C_2$ polynomially learnable by $C_1 \vee C_2$?	C_1 polynomially learnable by C_1 from POS	C_1 polynomially learnable by C_1 from NEG	C_1 polynomially learnable by C_1 from POS and NEG
C_2 polynomially learnable by C_2 from POS	NP-hard in some cases	YES from POS and NEG	NP-hard in some cases
C_2 polynomially learnable by C_2 from NEG	YES from POS and NEG	YES from NEG	YES from POS and NEG
C_2 polynomially learnable by C_2 from POS and NEG	NP-hard in some cases	YES from POS and NEG	NP-hard in some cases

Figure 4.1: Polynomial learnability of $C_1 \vee C_2$ by $C_1 \vee C_2$.

$C_1 \wedge C_2$ polynomially learnable by $C_1 \wedge C_2$?	C_1 polynomially learnable by C_1 from POS	C_1 polynomially learnable by C_1 from NEG	C_1 polynomially learnable by C_1 from POS and NEG
C_2 polynomially learnable by C_2 from POS	YES from POS	YES from POS and NEG	YES from POS and NEG
C_2 polynomially learnable by C_2 from NEG	YES from POS and NEG	NP-hard in some cases	NP-hard in some cases
C_2 polynomially learnable by C_2 from POS and NEG	YES from POS and NEG	NP-hard in some cases	NP-hard in some cases

Figure 4.2: Polynomial learnability of $C_1 \wedge C_2$ by $C_1 \wedge C_2$.

then C_2 is polynomially learnable. This suggests a notion of reducibility between learning problems. Such a notion may also provide learning algorithms for representation classes that reduce to classes already known to be learnable.

In this section we describe polynomial-time reductions between learning problems for classes of Boolean circuits. These reductions are general and involve simple variable substitutions. Similar transformations have been given for the mistake-bounded model of learning by Littlestone [73]. Recently the notion of reducibility among learning problems has been elegantly generalized and developed into a complexity theory for polynomial learnability by Pitt and Warmuth [79].

The basic idea behind the reductions can be illustrated by the following simple example: suppose we have an efficient learning algorithm A for monomials, and we wish to devise an algorithm for the class 2CNF. Note that any 2CNF formula can be written as a *monomial* over the $O(n^2)$ variables of the form $z_{i,j} = (x_i \vee x_j)$. Thus we can use algorithm A to efficiently learn 2CNF simply by giving A examples of length n^2 in which each bit simulates the value of one of the created variables $z_{i,j}$ on an example of length n of the target 2CNF formula. In the remainder of the section we formalize and generalize these ideas.

If $C = \cup_{n \geq 1} C_n$ is a parameterized class of Boolean circuits, we say that C is *naming invariant* if for any circuit $c(x_1, \ldots, x_n) \in C_n$, and any permutation π of $\{1, \ldots, n\}$, we have $c(x_{\pi(1)}, \ldots, x_{\pi(n)}) \in C_n$. We say that C is *upward closed* if for $n \geq 1$, $C_n \subseteq C_{n+1}$. Note that all of the classes of Boolean circuits studied here are both naming invariant and upward closed.

Theorem 4.7 *Let $C = \cup_{n \geq 1} C_n$ be a parameterized class of Boolean circuits that is naming invariant and upward closed. Let G be a set of Boolean circuits, each over k inputs (where k is a constant). Let C'_n be the class of circuits obtained by choosing any $c(x_1, \ldots, x_n) \in C_n$, and replacing one or more of the inputs x_i to c with any circuit $g_i(x_{i_1}, \ldots, x_{i_k})$, where $g_i \in G$, and each $x_{i_j} \in \{x_1, \ldots, x_n\}$ (thus, the circuit obtained is still over the variables $x_1, \ldots, x_n$). Let $C' = \cup_{n \geq 1} C'_n$. Then if C is polynomially learnable, C' is polynomially learnable.*

Proof: Let A be a polynomial-time learning algorithm for C. We describe

a polynomial-time learning algorithm A' for C' that uses algorithm A as a subroutine. For each circuit $g_i \in G$, A' creates n^k new variables $z_1^i, \ldots, z_{n^k}^i$. Let $X_1, \ldots, X_{n^k}$ denote all ordered lists of k variables chosen from $x_1, \ldots, x_n$, with repetition allowed. The intention is that z_j^i will simulate the value of the circuit g_i when g_i is given the variable list X_j as inputs.

Whenever algorithm A requests a positive (or negative) example, A' takes a positive (or negative) example $(v_1, \ldots, v_n) \in \{0,1\}^n$ of the target circuit $c'(x_1, \ldots, x_n) \in C'_n$. Let $c_j^i \in \{0,1\}$ be the value assigned to z_j^i by the simulation described above. Then A' gives the example

$$(v_1, \ldots, v_n, c_1^1, \ldots, c_{n^k}^1, \ldots, c_1^{|G|}, \ldots, c_{n^k}^{|G|})$$

to algorithm A. Since $c'(x_1, \ldots, x_n)$ was obtained by substitutions on some $c \in C_n$, and since C is naming invariant and upward closed, there is a circuit in $C_{n+|G|n^k}$ that is consistent with all the examples we generate by this procedure (it is just c' with each occurrence of the circuit g_i replaced by the variable z_j^i that simulates the correct inputs to the occurrence of g_i). Thus A must output an ϵ-good hypothesis

$$h_A(x_1, \ldots, x_n, z_1^1, \ldots, z_{n^k}^1, \ldots, z_1^{|G|}, \ldots, z_{n^k}^{|G|}).$$

We then obtain an ϵ-good hypothesis over n variables by defining

$$h_{A'}(v_1, \ldots, v_n) = h_A(v_1, \ldots, v_n, c_1^1, \ldots, c_{n^k}^1, \ldots, c_1^{|G|}, \ldots, c_{n^k}^{|G|})$$

for any $(v_1, \ldots, v_n) \in \{0,1\}^n$, where each c_j^i is computed as described above. This completes the proof. □

Note that if the learning algorithm A for C uses only positive examples or only negative examples, this property is preserved by the reduction of Theorem 4.7. As a corollary of Theorem 4.7 we have that for most natural Boolean circuit classes, the monotone learning problem is no harder than the general learning problem:

Corollary 4.8 *Let $C = \cup_{n \geq 1} C_n$ be a parameterized class of Boolean circuits that is naming invariant and upward closed. Let monotone C be the class containing all monotone circuits in C. Then if monotone C is polynomially learnable, C is polynomially learnable.*

Proof: In the statement of Theorem 4.7, let $G = \{\overline{y}\}$. Then all of the literals $\overline{x}_1, \ldots, \overline{x}_n$ can be obtained as instances of the single circuit in G. □

Theorem 4.7 says that the learning problem for a class of Boolean circuits does not become harder if an unknown subset of the variables is replaced by a constant-sized set of circuits whose inputs are unknown. The following result says this is also true if the number of substitution circuits is larger, but the order and inputs are known.

Theorem 4.9 *Let $C = \cup_{n \geq 1} C_n$ be a parameterized class of Boolean circuits that is naming invariant and upward closed. Let $p(n)$ be a fixed polynomial, and let the description of the $p(n)$-tuple $(g_1^n, \ldots, g_{p(n)}^n)$ be computable in time polynomial in n, where each g_i^n is a Boolean circuit over n variables. Let C_n' consist of circuits of the form*

$$c(g_1^n(x_1, \ldots, x_n), \ldots, g_{p(n)}^n(x_1, \ldots, x_n))$$

where $c \in C_{p(n)}$. Let $C' = \cup_{n \geq 1} C_n'$. Then if C is polynomially learnable, C' is polynomially learnable.

Proof: Let A be a polynomial-time learning algorithm for C. We describe a polynomial-time learning algorithm A' for C' that uses algorithm A as a subroutine. Similar to the proof of Theorem 4.7, A' creates new variables $z_1, \ldots, z_{p(n)}$. The intention is that z_i will simulate $g_i^n(x_1, \ldots, x_n)$.

When algorithm A requests a positive or a negative example, A' takes a positive or negative example $(v_1, \ldots, v_n) \in \{0,1\}^n$ of the target circuit $c'(x_1, \ldots, x_n) \in C_n'$ and sets $c_i = g_i^n(v_1, \ldots, v_n)$. A' then gives the vector $(c_1, \ldots, c_{p(n)})$ to A. As in the proof of Theorem 4.7, A must output an ϵ-good hypothesis h_A over $p(n)$ variables. We then define $h_{A'}(v_1, \ldots, v_n) = h_A(c_1, \ldots, c_{p(n)})$, for any $v_1, \ldots, v_n \in \{0,1\}^n$, where each c_i is computed as described above. □

If the learning algorithm A for C uses only positive examples or only negative examples, this property is preserved by the reduction of Theorem 4.9. We can apply this result to demonstrate that μ circuits, or *read-once* circuits, are no easier to learn than general circuits.

Corollary 4.10 *Let $C = \cup_{n \geq 1} C_n$ be a parameterized class of Boolean circuits that is naming invariant and upward closed. Let μC consist of all circuits in C in which each variable occurs at most once (i.e., the fan-out of each input variable is at most 1). Then if μC is polynomially learnable, C is polynomially learnable.*

Proof: Let $c \in C$, and let l be the maximum number of times any variable occurs (i.e., the largest fan-out) in c. Then in the statement of Theorem 4.9, let $p(n) = ln$ and $g^n_{in+j} = x_j$ for $0 \leq i \leq l-1$ and $1 \leq j \leq n$ (thus, $g^n_{in+j} = x_j$ is essentially a *copy* of x_j). Note that if we do not know the value of l, we can try successively larger values, testing the hypothesis each time until an ϵ-good hypothesis is obtained. □

Corollaries 4.8 and 4.10 are particularly useful for simplifying the learning problem for classes whose polynomial-time learnability is in question. For example, if we let $\mathrm{DNF}^{p(n)}$ be the class of all DNF formulae in which the length is bounded by some polynomial $p(n)$ (where n is the number of variables), and monotone μDNF is the class of DNF formulae in which no variable occurs more than once and no variable occurs negated, then we have:

Corollary 4.11 *If monotone $\mu\mathrm{DNF}^{p(n)}$ (respectively, monotone $\mu\mathrm{CNF}^{p(n)}$) is polynomially learnable, then $\mathrm{DNF}^{p(n)}$ (respectively, $\mathrm{CNF}^{p(n)}$) is polynomially learnable.*

It is important to note that the substitutions suggested by Theorems 4.7 and 4.9 and their corollaries do not preserve the underlying target distributions. For example, it does *not* follow from Corollary 4.11 that if monotone μDNF is polynomially learnable under uniform target distributions (as is shown in Chapter 8) then DNF is polynomially learnable under uniform distributions.

5

Learning in the Presence of Errors

5.1 Introduction

In this chapter we study a practical extension to the distribution-free model of learning: the presence of errors (possibly maliciously generated by an adversary) in the sample data. Thus far we have made the idealized assumption that the oracles *POS* and *NEG* always faithfully return untainted examples of the target representation drawn according to the target distributions. In many environments, however, there is always some chance that an erroneous example is given to the learning algorithm. In a training session for an expert system, this might be due to an occasionally faulty teacher; in settings where the examples are being transmitted electronically, it might be due to unreliable communication equipment.

Since one of the strengths of Valiant's model is the lack of assumptions on the probability distributions from which examples are drawn, we seek to preserve this generality by making no assumptions on the *nature* of the errors that occur. That is, we wish to avoid demanding algorithms that work under any target distributions while at the same time assuming that the errors in the examples have some "nice" form. Such well-behaved sources of error seem difficult to justify in a real computing environment, where the rate of error may be small, but data may become badly mangled by highly unpredictable forces whenever errors do occur, for example in the case of hardware errors. Thus, we study a worst-case or *malicious* model of errors, in which the errors are generated by an adversary whose goal is to foil the learning algorithm.

The study of learning from examples with malicious errors was initiated by Valiant [94], where it is assumed that there is a fixed probability β of an error occurring independently on each request for an example. This error may be of an arbitrary nature — in particular, it may be chosen by an adversary with unbounded computational resources, and exact knowledge of the target representation, the target distributions, and the current internal state of the learning algorithm.

In this chapter we study the *optimal malicious error rate* $E_{MAL}(C)$ for a representation class C — that is, the largest value of β that can be tolerated by any learning algorithm (not necessarily polynomial time) for C. Note that we expect the optimal error rate to depend on ϵ and δ (and n in the case of a parameterized target class C). An upper bound on $E_{MAL}(C)$ corresponds to a hardness result placing limitations on the rate of error that can be tolerated; lower bounds on $E_{MAL}(C)$ are obtained by giving algorithms that tolerate a certain rate of error.

Using a proof technique called the method of induced distributions, we obtain general upper bounds on $E_{MAL}(C)$ and apply these results to many representation classes. We also obtain lower bounds on $E_{MAL}^{poly}(C)$ (the largest rate of malicious error tolerated by a polynomial-time learning algorithm for C) by giving efficient learning algorithms for these same classes and analyzing their error tolerance. In several cases the upper and lower bounds on $E_{MAL}^{poly}(C)$ meet. A canonical method of transforming standard learning algorithms into error-tolerant algorithms is given, and we give approximation-preserving reductions between standard combinatorial optimization problems such as set cover and natural problems of learning with errors. Several of our results also apply to a more benign model of *classification noise* defined by Angluin and Laird [12], in which the underlying target distributions are unaltered, but there is some probability that a positive example is incorrectly classified as being negative, and vice-versa.

Several themes are brought out. One is that error tolerance need not come at the expense of efficiency or simplicity. We show that there are representation classes for which the optimal malicious error rate can be achieved by algorithms that run in polynomial time and are easily coded. For example, we show that a polynomial-time algorithm for learning monomials with errors due to Valiant [94] tolerates the largest malicious error rate possible for any algorithm that uses only positive examples, polynomial-time or otherwise. We give an

efficient learning algorithm for the class of symmetric functions that tolerates the optimal malicious error rate and uses an optimal number of examples.

Another theme is the importance of using both positive and negative examples whenever errors (either malicious errors or classification noise errors) are present. Several existing learning algorithms use only positive examples or only negative examples (see e.g. Valiant [93] and Blumer et al. [25]). We demonstrate strong upper bounds on the tolerable error rate when only one type is used, and show that this rate can be provably increased when both types are used. In addition to proving this for the class of symmetric functions, we give an efficient algorithm that provides a strict increase in the malicious error rate over the positive-only algorithm of Valiant [94] for the class of monomials.

A third theme is that there are strong ties between learning with errors and more traditional problems in combinatorial optimization. We give a reduction from learning monomials with errors to a generalization of the weighted set cover problem, and give an approximation algorithm for this problem (generalizing the greedy algorithm analyzed by several authors [29, 56, 77]) that is of independent interest. This approximation algorithm is used as a subroutine in a learning algorithm that tolerates an improved error rate for monomials. In the other direction, we prove that for M the class of monomials, approaching the optimal error rate $E_{MAL}(M)$ with a polynomial-time algorithm using hypothesis space M is at least as hard as finding an efficient approximation algorithm with an improved performance guarantee for the set cover problem. This suggests that there are classes for which the optimal error rate that can be tolerated *efficiently* may be considerably smaller than the optimal *information-theoretic* rate. The best approximation known for the set cover problem remains the greedy algorithm analyzed by Chvatal [29], Johnson [56], Lovasz [75], and Nigmatullin [77]. Finally, we give a canonical reduction that allows many learning with errors problems to be studied as equivalent optimization problems, thus allowing one to sidestep some of the difficulties of analysis in the distribution-free model. Similar results are given for the error-free model by Haussler et al. [51].

We now give a brief survey of other studies of error in the distribution-free model. Valiant [94] modified his initial definitions of learnability to include the presence of errors in the examples. He also gave a generalization of his algorithm for learning monomials from positive examples, and analyzed the rate of malicious error tolerated by this algorithm. Valiant's results led him

to suggest the possibility that "the learning phenomenon is only feasible with very low error rates" (at least in the distribution-free setting with malicious errors); some of the results presented in this chapter can be viewed as giving formal verification of this intuition. On the other hand, some of our algorithms provide hope that if one can somehow reliably control the *rate* of error to a small amount, then errors of an arbitrary nature can be compensated for by the learning process.

Angluin and Laird [12] subsequently modified Valiant's definitions to study a non-malicious model of errors, defined in Section 5.2 as the *classification noise model*. Their results demonstrate that under stronger assumptions on the nature of the errors, large rates of error can be tolerated by polynomial-time algorithms for nontrivial representation classes. Shackelford and Volper [91] investigated the classification noise model further, and Sloan [92] and Laird [67] discuss a number of variants of both the malicious error and classification noise models.

5.2 Definitions and notation for learning with errors

Oracles with malicious errors. Let C be a representation class over a domain X, and let $c \in C$ be the target representation with target distributions D^+ and D^-. For $0 \leq \beta < 1/2$, we define two *oracles with malicious errors*, POS^{β}_{MAL} and NEG^{β}_{MAL}, that behave as follows: when oracle POS^{β}_{MAL} (respectively, NEG^{β}_{MAL}) is called, with probability $1-\beta$, a point $x \in pos(c)$ (respectively, $x \in neg(c)$) randomly chosen according to D^+ (respectively, D^-) is returned, as in the error-free model; but with probability β, a point $x \in X$ on which absolutely no assumptions can be made is returned. In particular, this point may be dynamically and maliciously chosen by an adversary who has knowledge of c, D^+, D^-, β and the internal state of the learning algorithm. This adversary also has unbounded computational resources. For convenience we assume that the adversary does not have knowledge of the outcome of future coin flips of the learning algorithm or the points to be returned in future calls to POS^{β}_{MAL} and NEG^{β}_{MAL} (other than those that the adversary may himself decide to generate on future errors). These assumptions may in fact be

removed, as our results will show, resulting in a stronger model where the adversary may choose to modify in any manner a fixed fraction β of the sample to be given to the learning algorithm. Such a model realistically captures situations such as "error bursts", which may occur when transmission equipment malfunctions repeatedly for a short amount of time.

Learning from oracles with malicious errors. Let C and H be representation classes over X. Then for $0 \leq \beta < 1/2$, we say that C is *learnable by H with malicious error rate β* if there is a (probabilistic) algorithm A with access to POS^{β}_{MAL} and NEG^{β}_{MAL}, taking inputs ϵ, δ and β_0, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any input values $0 < \epsilon, \delta < 1$ and $\beta \leq \beta_0 < 1/2$, algorithm A halts and outputs a representation $h_A \in H$ that with probability at least $1 - \delta$ satisfies $e^+(h_A) < \epsilon$ and $e^+(h_A) < \epsilon$.

We will also say that A is a *β-tolerant learning algorithm for C*. In this definition of learning, polynomial-time means polynomial in $1/\epsilon, 1/\delta$ and $1/(1/2 - \beta_0)$, as well as polynomial in n in the case of parameterized C (where as mentioned in Chapter 2, we assume that the length of representations in C_n are bounded by a polynomial in n).

The input β_0 is intended to provide an upper bound on the error rate for the learning algorithm, since in practice we do not expect to have exact knowledge of the "true" error rate β (for instance, it is reasonable to expect the error rate to vary somewhat with time). The dependence on $1/(1/2 - \beta_0)$ for polynomial-time algorithms provides the learning algorithm with more time as the error rate approaches $1/2$, since an error rate of $1/2$ renders learning impossible for any algorithm, polynomial-time or otherwise. However, we will shortly see that the input β_0 and the dependence of the running time on $1/(1/2 - \beta_0)$ are usually unnecessary, since for learning under arbitrary target distributions to be possible we must have $\beta < \epsilon/(1 + \epsilon)$ (under very weak restrictions on C). This is Theorem 5.1. However, we include β_0 in our definitions since these dependencies may be meaningful for learning under restricted target distributions.

It is important to note that in this definition, we are *not* asking learning algorithms to "fit the noise" in the sense of achieving accuracy in predict-

ing the behavior of the tainted oracles POS^{β}_{MAL} and NEG^{β}_{MAL}. Rather, the conditions $e^+(h_A) < \epsilon$ and $e^+(h_A) < \epsilon$ require that the algorithm find a good predictive model of the true underlying target distributions D^+ and D^-, as in the error-free model.

In general, we expect the achievable malicious error rate to depend upon the desired accuracy ϵ and confidence δ, as well as on the parameter n in the case of parameterized representation classes. We now make definitions that will allow us to study the largest rate $\beta = \beta(\epsilon, \delta, n)$ that can be tolerated by any learning algorithm, and by learning algorithms restricted to run in polynomial time.

Optimal malicious error rates. Let A be a learning algorithm for C. We define $E_{MAL}(C, A)$ to be the largest β such that A is a β-tolerant learning algorithm for C; note that $E_{MAL}(C, A)$ is actually a function of ϵ and δ (and n in the case of parameterized C). In the case that the largest such β is not well-defined (for example, A could tolerate progressively larger rates if allowed more time), then $E_{MAL}(C, A)$ is the supremum over all malicious error rates tolerated by A. Then we define the function $E_{MAL}(C)$ to be the pointwise (with respect to ϵ, δ and n in the parameterized case) supremum of $E_{MAL}(C, A)$, taken over all learning algorithms A for C. More formally, if we write $E_{MAL}(C, A)$ and $E_{MAL}(C)$ in functional form, then $E_{MAL}(C)(\epsilon, \delta, n) = \sup_A\{E_{MAL}(C, A)(\epsilon, \delta, n)\}$. Notice that this supremum is taken over *all* learning algorithms, regardless of computational complexity. We will use the notation E^{poly}_{MAL} to denote these same quantities when the quantification is only over polynomial-time learning algorithms — thus, for instance, $E^{poly}_{MAL}(C, A)$ is the largest β such that A is a β-tolerant learning polynomial-time learning algorithm for C, and $E^{poly}_{MAL}(C)$ is the largest malicious error rate tolerated by any polynomial-time learning algorithm for C.

$E_{MAL,+}(C)$ will be used to denote E_{MAL} with quantification only over positive-only learning algorithms for C; Similar definitions are made for the negative-only malicious error rate $E_{MAL,-}$, and polynomial-time positive-only and polynomial-time negative-only malicious error rates $E^{poly}_{MAL,+}$ and $E^{poly}_{MAL,-}$.

Oracles with classification noise. Some of our results will also apply to a more benign model of errors defined by Angluin and Laird [12], which we will call the *classification noise* model. Here we have oracles POS^{β}_{CN}

and NEG^{β}_{CN} that behave as follows: as before, with probability $1-\beta$, POS^{β}_{CN} returns a point drawn randomly according to the target distribution D^+. However, with probability β, POS^{β}_{CN} returns a point drawn randomly according to the *negative* target distribution D^-. Similarly, with probability $1-\beta$, NEG^{β}_{CN} draws from the correct distribution D^- and with probability β draws from D^+. This model is easily seen to be equivalent (modulo polynomial time) to a model in which a learning algorithm asks for a labeled example without being allowed to specify whether this example will be positive or negative; then the noisy oracle draws from the underlying target distributions (each with equal probability), but with probability β returns an incorrect classification with the example drawn.

These oracles are intended to model a situation in which the learning algorithm's "teacher" occasionally misclassifies a positive example as negative, and vice-versa. However, this misclassification is benign in the sense that the erroneous example is always drawn according to the "natural" environment as represented by the target distributions; thus, only the classification label is subject to error. In contrast, errors in the malicious model may involve not only misclassification, but alteration of the examples themselves, which may not be generated according to any probability distribution at all. As an example, the adversary generating the errors may choose to give significant probability to examples that have zero probability in the true target distributions. We will see throughout the chapter that these added capabilities of the adversary have a crucial effect on the error rates that can be tolerated.

Learning from oracles with classification noise. Let C and H be representation classes over X. Then for $0 \leq \beta < 1/2$, we say that C is *learnable by H with classification noise rate β* if there is a (probabilistic) algorithm A with access to POS^{β}_{CN} and NEG^{β}_{CN}, taking inputs ϵ, δ and β_0, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any input values $0 < \epsilon, \delta < 1$ and $\beta \leq \beta_0 < 1/2$, algorithm A halts and outputs a representation $h_A \in H$ that with probability at least $1-\delta$ satisfies $e^+(h_A) < \epsilon$ and $e^+(h_A) < \epsilon$.

Polynomial time here means polynomial in $1/\epsilon, 1/\delta$ and $1/(1/2-\beta_0)$, as well as the polynomial in n in the case of parameterized C. As opposed to the malicious case, the input β_0 is relevant here, even in the case of

arbitrary target distributions, since classification noise rates approaching $1/2$ can be tolerated by polynomial-time algorithms for some nontrivial representation classes [12].

Optimal classification noise rates. Analogous to the malicious model, we define *classification noise rates* E_{CN}, $E_{CN,+}$ and $E_{CN,-}$ for an algorithm A and representation class C, as well as polynomial-time classification noise rates E_{CN}^{poly}, $E_{CN,+}^{poly}$ and $E_{CN,-}^{poly}$.

5.3 Absolute limits on learning with errors

In this section we prove theorems bounding the achievable error rate for both the malicious error and classification noise models. These bounds are absolute in the sense that they apply to *any* learning algorithm, regardless of its computational complexity, the number of examples it uses, the hypothesis space it uses, and so on. Our first such result states that the malicious error rate must be smaller than the desired accuracy ϵ. This is in sharp contrast to the classification noise model, where Angluin and Laird [12] proved, for example, $E_{CN}^{poly}(k\text{DNF}_n) \geq c_0$ for all n and any constant $c_0 < 1/2$.

Let us call a representation class C *distinct* if there exist representations $c_1, c_2 \in C$ and points $u, v, w, x \in X$ satisfying $u \in pos(c_1), u \in neg(c_2)$, $v \in pos(c_1), v \in pos(c_2)$, $w \in neg(c_1), w \in pos(c_2)$, and $x \in neg(c_1), x \in neg(c_2)$.

Theorem 5.1 *Let C be a distinct representation class. Then*

$$E_{MAL}(C) < \frac{\epsilon}{1+\epsilon}.$$

Proof: We use a technique that we will call the *method of induced distributions*: we choose $l \geq 2$ representations $\{c_i\}_{i\in\{1,\ldots,l\}} \subseteq C$, along with l pairs of target distributions $\{D_{c_i}^+\}_{i\in\{1,\ldots,l\}}$ and $\{D_{c_i}^-\}_{i\in\{1,\ldots,l\}}$. These representations and target distributions are such that for any $i \neq j$, $1 \leq i, j \leq l$, c_j is ϵ-bad with respect to the distributions $D_{c_i}^+, D_{c_i}^-$. Then adversaries $\{ADV_{c_i}\}_{i\in\{1,\ldots,l\}}$ are constructed for generating any errors when c_i is the target representation such that the behavior of the oracle POS_{MAL}^{β} is identical regardless of which

c_i is the target representation; the same is true for the oracle NEG^{β}_{MAL}, thus making it impossible for any learning algorithm to distinguish the true target representation, and essentially forcing the algorithm to "guess" one of the c_i.

In the case of Theorem 5.1, this technique is easily applied, with $l = 2$, as follows: let $c_1, c_2 \in C$ and $u, v, w, x \in X$ be as in the definition of distinct. Define the following target distributions for c_1:

$$\begin{aligned} D^+_{c_1}(u) &= \epsilon \\ D^+_{c_1}(v) &= 1 - \epsilon \end{aligned}$$

and

$$\begin{aligned} D^-_{c_1}(w) &= \epsilon \\ D^-_{c_1}(x) &= 1 - \epsilon. \end{aligned}$$

For c_2, the target distributions are:

$$\begin{aligned} D^+_{c_2}(v) &= 1 - \epsilon \\ D^+_{c_2}(w) &= \epsilon \end{aligned}$$

and

$$\begin{aligned} D^-_{c_2}(u) &= \epsilon \\ D^-_{c_2}(x) &= 1 - \epsilon. \end{aligned}$$

Note that these distributions are such that any representation that disagrees with the target representation on one of the points u, v, w, x is ϵ-bad with respect to the target distributions. Now if c_1 is the target representation, then the adversary ADV_{c_1} behaves as follows: on calls to POS^{β}_{MAL}, ADV_{c_1} always returns the point w whenever an error occurs; on calls to NEG^{β}_{MAL}, ADV_{c_1} always returns the point u whenever an error occurs. Under these definitions, the oracle POS^{β}_{MAL} draws a point from an *induced* distribution $I^+_{c_1}$ that is determined by the joint behavior of the distribution $D^+_{c_1}$ and the adversary ADV_{c_1}, and is given by

$$\begin{aligned} I^+_{c_1}(u) &= (1 - \beta)\epsilon \\ I^+_{c_1}(v) &= (1 - \beta)(1 - \epsilon) \\ I^+_{c_1}(w) &= \beta \end{aligned}$$

where β is the malicious error rate. Similarly, the oracle NEG^{β}_{MAL} draws from an induced distribution $I^{-}_{c_1}$:

$$\begin{aligned} I^{-}_{c_1}(u) &= \beta \\ I^{-}_{c_1}(w) &= (1-\beta)\epsilon \\ I^{-}_{c_1}(x) &= (1-\beta)(1-\epsilon). \end{aligned}$$

For target representation c_2, the adversary ADV_{c_2} always returns the point u whenever a call to POS^{β}_{MAL} results in an error, and always returns the point w whenever a call to NEG^{β}_{MAL} results in an error. Then the oracle POS^{β}_{MAL} draws from the induced distribution

$$\begin{aligned} I^{+}_{c_2}(u) &= \beta \\ I^{+}_{c_2}(v) &= (1-\beta)(1-\epsilon) \\ I^{+}_{c_2}(w) &= (1-\beta)\epsilon \end{aligned}$$

and the oracle NEG^{β}_{MAL} from the induced distribution

$$\begin{aligned} I^{-}_{c_2}(u) &= (1-\beta)\epsilon \\ I^{-}_{c_2}(w) &= \beta \\ I^{-}_{c_2}(x) &= (1-\beta)(1-\epsilon). \end{aligned}$$

It is easily verified that if $\beta = \epsilon/(1+\epsilon)$, then the distributions $I^{+}_{c_1}$ and $I^{+}_{c_2}$ are identical, and that $I^{-}_{c_1}$ and $I^{-}_{c_2}$ are identical; if $\beta > \epsilon/(1+\epsilon)$, the adversary may always choose to flip a biased coin, and be "honest" (i.e., draw from the correct target distribution) when the outcome is heads, thus reducing the effective error rate to exactly $\epsilon/(1+\epsilon)$. Thus, under these distributions and adversaries, the behavior of the oracles POS^{β}_{MAL} and NEG^{β}_{MAL} is identical regardless of the target representation. This implies that any algorithm that produces an ϵ-good hypothesis for target representation c_1 with probability at least $1-\delta$ under the distributions $D^{+}_{c_1}$ and $D^{-}_{c_1}$ must fail to output an ϵ-good hypothesis for target representation c_2 with probability at least $1-\delta$ under the distributions $D^{+}_{c_2}$ and $D^{-}_{c_2}$, thus proving the theorem. □

Note that Theorem 5.1 actually holds for any fixed ϵ. An intuitive interpretation of the result is that if we desire 90 percent accuracy from the hypothesis, there must be less than about 10 percent error.

We emphasize that Theorem 5.1 bounds the achievable malicious error rate for *any* learning algorithm, regardless of computational complexity, sample complexity or the hypothesis class. Thus, for distinct C, we always have $E_{MAL}(C) \leq \epsilon/(1+\epsilon) = O(\epsilon)$. All of the representation classes studied here are distinct. We shall see in Theorem 5.7 of Section 5.4 that any hypothesis that nearly minimizes the number of disagreements with a large enough sample from POS^{β}_{MAL} and NEG^{β}_{MAL} is ϵ-good with high probability provided $\beta < \epsilon/4$. Thus, for the finite representation classes we study here (such as all the classes over the Boolean domain $\{0,1\}^n$), there is always a (possibly super-polynomial time) exhaustive search algorithm A achieving $E_{MAL}(C,A) = \Omega(\epsilon)$; combined with Theorem 5.1, this gives $E_{MAL}(C) = \Theta(\epsilon)$ for these classes. However, we will primarily be concerned with achieving the largest possible malicious error rate in polynomial time.

We now turn our attention to positive-only and negative-only learning in the presence of errors, where we will see that for many representation classes, the absolute bounds on the achievable error rate are even stronger than those given by Theorem 5.1.

Let C be a representation class. We will call C *positive t-splittable* if there exist representations $c_1, \ldots, c_t \in C$ and points $u_1, \ldots, u_t \in X$ and $v \in X$ satisfying all of the following conditions:

$$\begin{aligned} u_i &\in pos(c_j), i \neq j, 1 \leq i,j \leq t \\ u_j &\in neg(c_j), 1 \leq j \leq t \\ v &\in pos(c_i), 1 \leq i \leq t. \end{aligned}$$

Similarly, C is *negative t-splittable* if we have

$$\begin{aligned} u_i &\in neg(c_j), i \neq j, 1 \leq i,j \leq t \\ u_j &\in pos(c_j), 1 \leq j \leq t \\ v &\in neg(c_i), 1 \leq i \leq t. \end{aligned}$$

Note that if $\text{VCD}(C) = d$, then C is both positive and negative d-splittable. The converse does not necessarily hold.

Theorem 5.2 *Let C be positive t-splittable (respectively, negative t-splittable). Then for $\delta \leq 1/t$,*

$$E_{MAL,+}(C) < \frac{\epsilon}{t-1}$$

(respectively, $E_{MAL,-}(C) < \frac{\epsilon}{t-1}$*).*

Proof: The proof is by the method of induced distributions. We prove only the case that C is positive t-splittable; the proof for C negative t-splittable is similar. Let $c_1, \ldots, c_t \in C$ and $u_1, \ldots, u_t, v \in X$ be as in the definition of positive t-splittable. For target representation c_j, define the target distributions $D^+_{c_j}$ over $pos(c_j)$ and $D^-_{c_j}$ over $neg(c_j)$ as follows:

$$\begin{aligned} D^+_{c_j}(u_i) &= \frac{\epsilon}{t-1}, 1 \leq i \leq t, i \neq j \\ D^+_{c_j}(v) &= 1 - \epsilon \end{aligned}$$

and

$$D^-_{c_j}(u_j) = 1.$$

For target representation c_j, the errors on calls to POS^{β}_{MAL} are generated by an adversary ADV_{c_j} who always returns the point u_j whenever an error occurs. Then under these definitions, POS^{β}_{MAL} draws a point from a distribution $I^+_{c_j}$ induced by the distribution $D^+_{c_j}$ and the adversary ADV_{c_j}. This distribution is

$$\begin{aligned} I^+_{c_j}(u_i) &= (1-\beta)\frac{\epsilon}{t-1}, 1 \leq i \leq t, i \neq j \\ I^+_{c_j}(v) &= (1-\beta)(1-\epsilon) \\ I^+_{c_j}(u_j) &= \beta. \end{aligned}$$

If $\beta = (1-\beta)(\epsilon/(t-1))$, then the induced distributions $I^+_{c_j}$ are all identical for $1 \leq j \leq t$. Solving, we obtain $\beta = (\epsilon/(t-1))/(1+\epsilon/(t-1)) < \epsilon/(t-1)$. Now let $\beta \geq \epsilon/(t-1)$, and assume A is a β-tolerant positive-only learning algorithm for C. If c_j is the target representation, then with probability at least $1-\delta$, $u_i \in pos(h_A)$ for some $i \neq j$, otherwise $e^+(h_A) \geq \epsilon$ under the induced distribution $I^+_{c_j}$. Let k be such that

$$\mathbf{Pr}[u_k \in pos(h_A)] = \max_{1 \leq i \leq t}\{\mathbf{Pr}[u_i \in pos(h_A)]\}$$

where the probability is taken over all sequences of examples given to A by the oracle POS^{β}_{MAL} and the coin tosses of A. Then we must have

$$\mathbf{Pr}[u_k \in pos(h_A)] \geq \frac{1-\delta}{t-1}.$$

Choose $\delta < 1/t$. Then with probability at least δ, $e^-(h_A) = 1$ when c_k is the target representation, with distributions $D^+_{c_k}$ and $D^-_{c_k}$ and adversary ADV_{c_k}. This contradicts the assumption that A is a β-tolerant learning algorithm, and the theorem follows. □

Note that the restriction $\delta < 1/t$ in the proof of Theorem 5.2 is apparently necessary, since a learning algorithm may always randomly choose a u_j to be a positive example, and make all other u_i negative examples; the probability of failing to learn under the given distributions is then only $1/t$. It would be interesting to find a different proof that removed this restriction, or to prove that it is required.

As in the case of Theorem 5.1, Theorem 5.2 is an upper bound on the achievable malicious error rate for *all* learning algorithms, regardless of hypothesis representation, number of examples used or computation time. For any representation class C, by computing a value t such C is t-splittable, we can obtain upper bounds on the positive-only and negative-only error rates for that class. As examples, we state such results as corollaries for a few of the representation classes studied here. Even in cases where the representation class is known to be not learnable from only positive or only negative examples in polynomial time (for example, we show in Section 6.2 that monomials are not polynomially learnable from negative examples), the bounds on $E_{MAL,+}$ and $E_{MAL,-}$ are relevant since they also hold for algorithms that do not run in polynomial time.

Corollary 5.3 *Let M_n be the class of monomials over $x_1, \ldots, x_n$. Then*

$$E_{MAL,+}(M_n) < \frac{\epsilon}{n-1}$$

and

$$E_{MAL,-}(M_n) < \frac{\epsilon}{n-1}.$$

Corollary 5.4 *For fixed k, let $k\mathrm{DNF}_n$ be the class of kDNF formulae over $x_1, \ldots, x_n$. Then*

$$E_{MAL,+}(k\mathrm{DNF}_n) = O\left(\frac{\epsilon}{n^k}\right)$$

and

$$E_{MAL,-}(k\mathrm{DNF}_n) = O\left(\frac{\epsilon}{n^k}\right).$$

Corollary 5.5 *Let* SF_n *be the class of symmetric functions over* $x_1, \ldots, x_n$. *Then*

$$E_{MAL,+}(\mathrm{SF}_n) < \frac{\epsilon}{n-1}$$

and

$$E_{MAL,-}(\mathrm{SF}_n) < \frac{\epsilon}{n-1}.$$

Proofs of these corollaries follow from the Vapnik-Chervonenkis dimension of the representation classes and Theorem 5.2. Note that the proof of Theorem 5.2 shows that these corollaries actually hold for any fixed ϵ and n.

We note that Theorem 5.2 and its corollaries also hold for the classification noise model. To see this it suffices to notice that the adversaries ADV_{c_j} in the proof of Theorem 5.2 simulated the classification noise model. Thus, for classification noise we see that the power of using both positive and negative examples may be dramatic: for kCNF we have $E_{CN}^{poly}(k\mathrm{CNF}_n) \geq c_0$ for any $c_0 < 1/2$ due to Angluin and Laird [12] but $E_{CN,+}(k\mathrm{CNF}_n) = O(\epsilon/n^k)$ by Theorem 5.2. (By Theorem 6.1 of Section 6.2, kCNF is not learnable in polynomial time from negative examples even in the error-free model.) In fact, we can give a bound on $E_{CN,+}$ and $E_{CN,-}$ that is weaker but more general, and applies to almost any representation class. Note that by exhaustive search techniques, we have that for any small constant α, $E_{CN}(C) \geq 1/2 - \alpha$ for any finite representation class C. Thus the following result demonstrates that for representation classes over finite domains in the classification noise model, the advantage of using both positive and negative examples is almost always significant.

We will call a representation class C *positive* (respectively, *negative*) *incomparable* if there are representations $c_1, c_2 \in C$ and points $u, v, w \in X$ satisfying $u \in pos(c_1), u \in neg(c_2), v \in pos(c_1), v \in pos(c_2)$ (respectively, $v \in neg(c_1), v \in neg(c_2)$), $w \in neg(c_1), w \in pos(c_2)$.

Theorem 5.6 *Let* C *be positive (respectively, negative) incomparable. Then*

$$E_{CN,+}(C) < \frac{\epsilon}{1+\epsilon}$$

(respectively, $E_{CN,-}(C) < \frac{\epsilon}{1+\epsilon}$*).*

Proof: By the method of induced distributions. We do the proof for the case that C is positive incomparable; the proof when C is negative incomparable is similar. Let $c_1, c_2 \in C$ and $u, v, w \in X$ be as in the definition of positive incomparable. For target representation c_1, we define distributions

$$\begin{aligned} D^+_{c_1}(u) &= \epsilon \\ D^+_{c_1}(v) &= 1-\epsilon \end{aligned}$$

and

$$D^-_{c_1}(w) = 1.$$

Then in the classification noise model, the oracle POS^{β}_{CN} draws from the induced distribution

$$\begin{aligned} I^+_{c_1}(u) &= (1-\beta)\epsilon \\ I^+_{c_1}(v) &= (1-\beta)(1-\epsilon) \\ I^+_{c_1}(w) &= \beta. \end{aligned}$$

For target representation c_2, define distributions

$$\begin{aligned} D^+_{c_2}(v) &= 1-\epsilon \\ D^+_{c_2}(w) &= \epsilon \end{aligned}$$

and

$$D^-_{c_2}(u) = 1.$$

Then for target representation c_2, oracle POS^{β}_{CN} draws from the induced distribution

$$\begin{aligned} I^+_{c_2}(u) &= \beta \\ I^+_{c_2}(v) &= (1-\beta)(1-\epsilon) \\ I^+_{c_2}(w) &= (1-\beta)\epsilon. \end{aligned}$$

For $\beta = \epsilon/(1+\epsilon)$, distributions $I^+_{c_1}$ and $I^+_{c_2}$ are identical. Any positive-only algorithm learning c_1 under $D^+_{c_1}$ and $D^-_{c_1}$ with probability at least $1-\delta$ must fail with probability at least $1-\delta$ when learning c_2 under $D^+_{c_2}$ and $D^-_{c_2}$. □

Thus, for positive (respectively, negative) incomparable C, $E_{CN,+}(C) = O(\epsilon)$ (respectively, $E_{CN,-}(C) = O(\epsilon)$). All of the representation classes studied here are both positive and negative incomparable. Note that the proof of Theorem 5.6 depends upon the assumption that a learning algorithm has only an upper bound on the noise rate, not the exact value; thus, the *effective* noise rate may be less than the given upper bound. However, this is a reasonable assumption in most natural environments. This issue does not arise in the malicious model, where the adversary may always choose to draw from the correct target distribution with some fixed probability, thus reducing the effective error rate to any value less than or equal to the given upper bound.

5.4 Efficient error-tolerant learning

Given the absolute upper bounds on the achievable malicious error rate of Section 5.3, we now wish to find *efficient* algorithms tolerating a rate that comes as close as possible to these bounds, or give evidence for the computational difficulty of approaching the optimal error rate. In this section we give efficient algorithms for several representation classes and analyze their tolerance to malicious errors.

We begin by giving a generalization of Occam's Razor (Theorem 3.1) for the case when errors are present in the examples.

Let C and H be representation classes over X. Let A be an algorithm accessing POS^{β}_{MAL} and NEG^{β}_{MAL}, and taking inputs $0 < \epsilon, \delta < 1$. Suppose that for target representation $c \in C$ and $0 \leq \beta < \epsilon/4$, A makes m calls to POS^{β}_{MAL} and receives points $u_1, \ldots, u_m \in X$, and m calls to NEG^{β}_{MAL} and receives points $v_1, \ldots, v_m \in X$, and outputs $h_A \in H$ satisfying with probability at least $1 - \delta$:

$$|\{u_i : u_i \in neg(h_A)\}| \leq \frac{\epsilon}{2}m \tag{5.1}$$

$$|\{v_i : v_i \in pos(h_A)\}| \leq \frac{\epsilon}{2}m. \tag{5.2}$$

Thus, with high probability, h_A is consistent with at least a fraction $1 - \epsilon/2$ of the sample received from the faulty oracles POS^{β}_{MAL} and NEG^{β}_{MAL}. We will call such an A a *β-tolerant Occam algorithm for C by H*.

Theorem 5.7 *Let $\beta < \epsilon/4$, and let A be a β-tolerant Occam algorithm for C by H. Then A is a β-tolerant learning algorithm for C by H; the sample size required is $m = O(1/\epsilon \ln 1/\delta + 1/\epsilon \ln |H|)$. If A is such that only Condition 5.1 (respectively, Condition 5.2) above holds, then $e^+(h_A) < \epsilon$ (respectively, $e^-(h_A) < \epsilon$) with probability at least $1 - \delta$.*

Proof: We prove the statement where A meets Condition 5.1; the case for Condition 5.2 is similar. Let $h \in H$ be such that $e^+(h) \geq \epsilon$. Then the probability that h agrees with a point received from the oracle POS^{β}_{MAL} is bounded above by

$$(1-\beta)(1-\epsilon) + \beta \leq 1 - \frac{3\epsilon}{4}$$

for $\beta < \epsilon/4$. Thus the probability that h agrees with at least a fraction $1-\epsilon/2$ of m examples received from POS^{β}_{MAL} is

$$LE\left(\frac{3\epsilon}{4}, m, \frac{\epsilon}{2}m\right) \leq e^{-m\epsilon/24}$$

by Fact CB1. From this it follows that the probability that *some* $h \in H$ with $e^+(h) \geq \epsilon$ agrees with a fraction $1 - \epsilon/2$ of the m examples is at most $|H|e^{-m\epsilon/24}$. Solving $|H|e^{-m\epsilon/24} \leq \delta/2$, we obtain $m \geq 24/\epsilon(\ln |H| + \ln 2/\delta)$. This proves that any h meeting Condition 5.1 is with high probability ϵ-good with respect to D^+, completing the proof. □

To demonstrate that the suggested approach of finding a nearly consistent hypothesis is in fact a feasible one, we note that if c is the target representation, then the probability that c fails to agree with at least a fraction $1 - \epsilon/2$ of m examples received from POS^{β}_{MAL} is

$$GE\left(\frac{\epsilon}{4}, m, \frac{\epsilon}{2}m\right) \leq \frac{\delta}{2}$$

for $\beta \leq \epsilon/4$ and m as in the statment of Theorem 5.7 by Fact CB2.

Thus, in the presence of errors of any kind, finding an $\epsilon/2$-good hypothesis is as good as learning, provided that $\beta < \epsilon/4$. This fact can be used to prove the correctness of the learning algorithms of the following two theorems due to Valiant.

Theorem 5.8 *(Valiant [94]) Let M_n be the class of monomials over $x_1, \ldots, x_n$. Then*

$$E^{poly}_{MAL,+}(M_n) = \Omega\left(\frac{\epsilon}{n}\right).$$

Theorem 5.9 *(Valiant [94]) For fixed k, let $k\mathrm{DNF}_n$ be the class of kDNF formulae over $x_1, \ldots, x_n$. Then*

$$E^{poly}_{MAL,-}(k\mathrm{DNF}_n) = \Omega\left(\frac{\epsilon}{n^k}\right).$$

Similar results are obtained by duality for the class of disjunctions (learnable from negative examples) and kCNF (learnable from positive examples); that is, $E^{poly}_{MAL,-}(1\mathrm{DNF}_n) = \Omega(\epsilon/n)$ and $E^{poly}_{MAL,+}(k\mathrm{CNF}_n) = \Omega(\epsilon/n^k)$. Note that the class of monomials (respectively, kDNF) is not polynomially learnable even in the error-free case from negative (respectively, positive) examples by Theorem 6.1 of Section 6.2.

Combining Theorems 5.8 and 5.9 with Corollaries 5.3 and 5.4 we have $E^{poly}_{MAL,+}(M_n) = \Theta(\epsilon/n)$ and $E^{poly}_{MAL,-}(k\mathrm{DNF}_n) = \Theta(\epsilon/n^k)$, thus proving that the algorithms of Valiant [94] tolerate the optimal malicious error rate with respect to positive-only and negative-only learning. The algorithm given in the following theorem, similar to those of Valiant [94], proves an analogous result for efficiently learning symmetric functions from only one type of examples in the presence of errors.

Theorem 5.10 *Let SF_n be the class of symmetric functions over $x_1, \ldots, x_n$. Then*

$$E^{poly}_{MAL,+}(\mathrm{SF}_n) = \Omega\left(\frac{\epsilon}{n}\right).$$

Proof: Let $\beta \leq \epsilon/8n$. The positive-only algorithm A maintains an integer array P indexed $0, \ldots, n$ and initialized to contain 0 at each location. A takes m (calculated below) examples from POS^{β}_{MAL}, and for each vector $\vec{v}$ received, increments $P[index(\vec{v})]$, where $index(\vec{v})$ is the number of bits set to 1 in $\vec{v}$. The hypothesis h_A is defined as follows: all vectors of index i are contained in $pos(h_A)$ if and only if $P[i] \geq (\epsilon/4n)m$; otherwise all vectors of index i are negative examples of h_A.

Note that h_A can disagree with at most a fraction $(\epsilon/4n)(n+1) < \epsilon/2$ of the m vectors received from POS^{β}_{MAL}, so $e^+(h_A) < \epsilon$ with high probability by Theorem 5.7. To prove that $e^-(h_A)$ with high probability, suppose that all vectors of index i are negative examples of the target representation (call such an i a *negative index*). Then the probability that a vector of index i is received on a call to POS^{β}_{MAL} is at most $\beta \leq \epsilon/8n$, since this occurs only when there is an error on a call to POS^{β}_{MAL}. Thus the probability of receiving $(\epsilon/4n)m$ vectors of index i in m calls to POS^{β}_{MAL} is

$$GE\left(\frac{\epsilon}{8n}, m, \frac{\epsilon}{4n}m\right) \leq e^{-m\epsilon/24n}$$

by Fact CB2. The probability that *some* negative index is classified as a positive index by h_A is thus at most

$$(n+1)e^{-m\epsilon/24n} \leq \frac{\delta}{2}$$

for $m = O((n/\epsilon)(\ln n + \ln 1/\delta))$. Thus with high probability, $e^-(h_A) = 0$, completing the proof. □

Thus, with Corollary 5.5 we have $E^{poly}_{MAL,+}(\mathrm{SF}_n) = \Theta(\epsilon/n)$. We can give a dual of the above algorithm to prove $E^{poly}_{MAL,-}(\mathrm{SF}_n) = \Theta(\epsilon/n)$ as well. The number of examples required by the algorithm of Theorem 5.10 is a factor of n larger than the lower bound of Corollary 6.13 for the error-free case; whether this increase is necessary for positive-only algorithms in the presence of malicious errors is an open problem.

The next theorem demonstrates that using both positive and negative examples can significantly increase the tolerated error rate in the malicious model.

Theorem 5.11 *Let* SF_n *be the class of symmetric functions over* $x_1, \ldots, x_n$. *Then*

$$E^{poly}_{MAL}(\mathrm{SF}_n) = \Omega(\epsilon).$$

Proof: Algorithm A maintains two integer arrays P and N, each indexed $0, \ldots, n$ and initialized to contain 0 at each location. A first takes m (calculated below) examples from POS^{β}_{MAL} and for each vector $\vec{v}$ received, increments

$P[index(\vec{v})]$, where $index(\vec{v})$ is the number of bits set to 1 in $\vec{v}$. A then takes m examples from NEG^{β}_{MAL} and increments $N[index(\vec{v})]$ for each vector $\vec{v}$ received. The hypothesis h_A is computed as follows: all vectors of index i are contained in $pos(h_A)$ if and only if $P[i] \geq N[i]$; otherwise, all vectors of index i are contained in $neg(h_A)$.

We now show that for sufficiently large m, A is an $\epsilon/8$-tolerant Occam algorithm. For $0 \leq i \leq n$, let $d_i = \min(P[i], N[i])$. Then $d = \sum_{i=0}^{n} d_i$ is the number of vectors in the sample of size $2m$ with which h_A disagrees. Now for each i, either $P[i]$ or $N[i]$ is a lower bound on the number e_i of malicious errors received that have index i; let $e = \sum_{i=0}^{n} e_i$. Note that $e \geq d$. Now the probability that e exceeds $(\epsilon/4)(2m)$ in m calls POS^{β}_{MAL} and m calls to NEG^{β}_{MAL} for $\beta \leq \epsilon/8$ is

$$GE(\frac{\epsilon}{8}, 2m, \frac{\epsilon}{4}2m) \leq \delta$$

for $m = O(1/\epsilon \ln 1/\delta)$ by Fact CB2. Thus, with high probability the number of disagreements d of h_A on the examples received is less than $(\epsilon/2)m$. This shows that A is an $\epsilon/8$-tolerant Occam algorithm for SF, and thus is a learning algorithm for SF by Theorem 5.7 for $m = O(1/\epsilon \ln 1/\delta + n/\epsilon)$. □

Thus, by Theorems 5.1 and 5.11 we have $E^{poly}_{MAL}(\mathrm{SF}_n) = \Theta(\epsilon)$ in contrast with $E^{poly}_{MAL,+}(\mathrm{SF}_n) = \Theta(\epsilon/n)$ and $E^{poly}_{MAL,-}(\mathrm{SF}_n) = \Theta(\epsilon/n)$, a provable increase by using both types of examples. This is also our first example of a nontrivial class for which the optimal error rate $\Theta(\epsilon)$ of Theorem 5.1 can be achieved by an efficient algorithm. Furthermore, the sample complexity of algorithm A above meets the lower bound (within a constant factor) for the error-free case given in Corollary 6.13; thus we have an algorithm with optimal sample complexity that tolerates the largest possible malicious error rate. This also demonstrates that it may be difficult to prove general theorems providing hard trade-offs between sample size and error rate.

We note that the proof of Theorem 5.11 relies only on the fact that there is a small number of equivalence classes of $\{0,1\}^n$ (namely, the sets of vectors with an equal number of bits set to 1) on which each symmetric function is constant. The same result thus holds for any Boolean representation class with this property.

Now that we have given some simple and efficient error-tolerant algorithms, we turn to the more abstract issue of general-purpose methods of making algo-

rithms more tolerant to errors. It is reasonable to ask whether for an arbitrary representation class C, polynomial learnability of C implies polynomial learnability of C with malicious error rate β, for some nontrivial value of β that depends on C, ϵ and δ. The next theorem answers this in the affirmative by giving an efficient technique for converting any learning algorithm into an error-tolerant learning algorithm.

Theorem 5.12 *Let A be a polynomial-time learning algorithm for C with sample complexity $S_A(\epsilon, \delta)$, and let $s = S_A(\epsilon/8, 1/2)$. Then for $\epsilon \leq 1/2$,*

$$E_{MAL}^{poly}(C) = \Omega\left(\frac{\ln s}{s}\right).$$

Proof: We describe a polynomial-time algorithm A' that tolerates the desired error rate and uses A as a subroutine. Note that S_A (and hence, s) may also depend upon n in the case of parameterized C.

Algorithm A' will run algorithm A many times with accuracy parameter $\epsilon/8$ and confidence parameter $1/2$. The probability that no errors occur during a single such run is $(1-\beta)^s$. For $\beta \leq \ln s/s$ we have

$$(1-\beta)^s \geq \left(1 - \frac{\ln s}{s}\right)^s \geq \frac{1}{s^2}.$$

(This lower bound can be improved to $1/s^\alpha$ for any constant $\alpha > 1$ provided there is a sufficiently small constant upper bound on ϵ.) Thus, on a single run of A there is probability at least $(1-\delta)1/s^2 = 1/2s^2$ that no errors occur and A outputs an $\epsilon/8$-good hypothesis h_A (call a run of A when this occurs a *successful* run). A' will run A r times. In r runs of A, the probability that no successful run of A occurs is at most

$$\left(1 - \frac{1}{2s^2}\right)^r < \frac{\delta}{3}$$

for $r > 2s^2 \ln 3/\delta$. Let $h_A^1, \ldots, h_A^r$ be the hypotheses output by A on these r runs. Suppose h_A^i is an ϵ-bad hypothesis with respect to the target distributions; without loss of generality, suppose $e^+(h_A^i) \geq \epsilon$. Then the probability that h_A^i agrees with an example returned by the oracle POS_{MAL}^{β} is then at

most $(1-\beta)(1-\epsilon)+\beta \leq 1-3\epsilon/4$ for $\beta \leq \epsilon/8$. Thus, the probability that h_A^i agrees with at least a fraction $1-\epsilon/2$ of m examples returned by POS_{MAL}^{β} is

$$LE(\frac{3\epsilon}{4}, m, \frac{\epsilon}{2}m) \leq e^{-m\epsilon/24}$$

by Fact CB1. Then it follows that the probability that *some* h_A^i with $e^+(h_A^i) \geq \epsilon$ agrees with a fraction $1-\epsilon/2$ of the m examples returned by POS_{MAL}^{β} is at most

$$re^{-m\epsilon/24} < \frac{\delta}{3}$$

for $m = O(1/\epsilon \ln r/\delta)$. Using Fact CB2, it can be shown that for $\beta \leq \epsilon/8$ the probability of an $\epsilon/8$-good h_A^i failing to agree with at least a fraction $1-\epsilon/2$ of the m examples is smaller than $\delta/3$.

Thus, if A is run r times and the resulting hypotheses are tested against m examples from both POS_{MAL}^{β} and NEG_{MAL}^{β}, then with probability at least $1-\delta$ the hypothesis with the fewest disagreements is in fact an ϵ-good hypothesis. Note that if A runs in polynomial time, A' also runs in polynomial time. □

Note that the trick used in the proof of Theorem 5.12 to eliminate the dependence of the tolerated error rate on δ is general: we may always set $\delta = 1/2$ and run A repeatedly to get a good hypothesis with high probability (provided we are willing to sacrifice a possible increase in the number of examples used). This technique has also been noted in the error-free setting by Haussler et al. [51].

It is shown in Theorem 6.10 that any learning algorithm A for a representation class C must have sample complexity

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \text{VCD}(C)\right)\right).$$

Suppose that a learning algorithm A achieves this optimal sample complexity (we will see in Section 6.3.1 that this holds for many existing learning algorithms). Then applying Theorem 5.12, we immediately obtain an algorithm for C that tolerates a malicious error rate of

$$\Omega\left(\frac{\text{VCD}(C)}{\epsilon}\ln\frac{\text{VCD}(C)}{\epsilon}\right).$$

This rate is also the best that can be obtained by applying Theorem 5.12. By applying this technique to the algorithm of Valiant [93] for the class of monomials in the error-free model, we obtain the following corollary:

Corollary 5.13 *Let M_n be the class of monomials over $x_1, \ldots, x_n$. Then*

$$E^{poly}_{MAL}(M_n) = \Omega\left(\frac{\epsilon}{n}\ln\frac{n}{\epsilon}\right).$$

This improves the malicious error rate tolerated by the polynomial-time algorithm of Valiant [94] in Theorem 5.8 by a logarithmic factor. Furthermore, since $E^{poly}_{MAL,+}(M) = \Theta(\epsilon/n)$ this proves that, as in the case of symmetric functions, using both oracles improves the tolerable error rate. Similarly, a slight improvement over the malicious error rate given in Theorem 5.9 for kDNF can also be shown. For decision lists, we can apply the algorithm of Rivest [84] and the sample size bounds given following Corollary 6.16 to obtain the following:

Corollary 5.14 *Let $k\mathrm{DL}_n$ be the class of k-decision lists over $x_1, \ldots, x_n$. Then*

$$E^{poly}_{MAL}(k\mathrm{DL}_n) = \Omega\left(\frac{\epsilon}{n^k}\right).$$

Despite the small improvement in the tolerable error rate for monomials of Corollary 5.13, there is still a significant gap between the absolute upper bound of $\epsilon/(1+\epsilon)$ on the achievable malicious error rate for monomials implied by Theorem 5.1 and the $\Omega(\epsilon/n\ln n/\epsilon)$ polynomial-time error rate of Corollary 5.13. We now describe further improvements that allow the error rate to primarily depend only on the number of *relevant* variables. We describe an algorithm tolerating a larger error rate for the class M^s_n of monomials with at most s literals, where s may depend on n, the total number of variables. Our algorithm will tolerate a larger rate of error when the number s of relevant attributes is considerably smaller than the total number of variables n. Other improvements in the performance of learning algorithms in the presence of many irrelevant attributes are investigated by Littlestone [73] and Blum [20].

We note that by applying Theorem 5.2 we can show that even for M^1_n, the class of monomials of length 1, the positive-only and negative-only malicious

error rates are bounded by $\epsilon/(n-1)$. This is again an absolute bound, holding regardless of the computational complexity of the learning algorithm. Thus, the positive-only algorithm of Valiant [94] in Theorem 5.8 cannot exhibit an improved error rate when restricted to the subclass M_n^s for *any* value of s.

Our error-tolerant learning algorithm for monomials is based on an approximation algorithm for a generalization of the set cover problem that we call the partial cover problem, which is defined below. This approximation algorithm is of independent interest and has found application in other learning algorithms [62, 98]. Our analysis and notation rely heavily on the work of Chvatal [29]; the reader may find it helpful to read his paper first.

The Partial Cover Problem:

Input: Finite sets $S_1, \ldots, S_n$ with positive real costs $c_1, \ldots, c_n$, and a positive fraction $0 < p \leq 1$. We assume without loss of generality that $\cup_{i=1}^{n} S_i = \{1, \ldots, m\} = T$ and we define $J = \{1, \ldots, n\}$.

Output: $J^* \subseteq J$ such that

$$|\bigcup_{j \in J^*} S_j| \geq pm$$

(we call such a J^* a *p-cover* of the S_i) and such that $cost_{PC}(J^*) = \sum_{j \in J^*} c_j$ is minimized.

Following Chvatal [29], for notational convenience we identify a partial cover $\{S_{j_1}, \ldots, S_{j_s}\}$ with the index set $\{j_1, \ldots, j_s\}$.

The partial cover problem is NP-hard, since it contains the set cover problem as a special case ($p = 1$) [39]. We now give a greedy approximation algorithm G for the partial cover problem.

Algorithm G:

Step 1. Initialize $J^* = \emptyset$.

Step 2. Set $q = pm - |\bigcup_{j \in J^*} S_j|$ (thus q is the number of still-uncovered elements that we must cover in order to have a p-cover). For each $j \notin J^*$, if $|S_j| > q$, delete any $|S_j| - q$ elements from S_j (delete excess elements from any remaining set that covers more than q elements).

Step 3. If $|\bigcup_{j \in J^*} S_j| \geq pm$ then halt and output J^*, since J^* is a p-cover.

Step 4. Find a k minimizing the ratio $c_k/|S_k|$. Add k to J^*, and replace each S_j by $S_j - S_k$. Return to Step 2.

Chvatal [29] shows that the greedy algorithm for the set cover problem cannot do better than $H(m)$ times the cost of an optimal cover, where $H(m) = \sum_{i=1}^{m} 1/i = \Theta(\log m)$. By a padding argument, this can also be shown to hold for algorithm G above, for any fixed p. We now prove that G can always achieve this approximation bound within a constant factor.

Theorem 5.15 *Let I be an instance of partial cover and let $opt_{PC}(I)$ denote the cost of an optimal p-cover for I. Then the cost of the p-cover J^* produced by algorithm G satisfies*

$$cost_{PC}(J^*) \leq (2H(m) + 3) opt_{PC}(I).$$

Proof: Let J_{opt} be an optimal p-cover (i.e., $cost_{PC}(J_{opt}) = opt_{PC}(I)$). Let

$$T_{opt} = \bigcup_{j \in J_{opt}} S_j$$

(these are the elements covered by J_{opt}) and

$$T^* = \bigcup_{j \in J^*} S_j$$

(these are the elements covered by J^*) where J^* is the p-cover output by algorithm G. Notice that $|T_{opt}| \geq pm$ since J_{opt} is a p-cover.

Let S_j^r be set of elements remaining in the set S_j immediately before Step 2 in algorithm G is executed for the rth time (i.e., at the start of the rth iteration of Steps 2-4). By appropriate renaming of the S_j, we may assume without loss of generality that $J^* = \{1, \ldots, r\}$ (recall that J^* is the set of *indices* of sets chosen by algorithm G) immediately after Step 4 is executed for the rth time (i.e., at the end of the rth iteration of Steps 2-4). Let $J^* = \{1, \ldots, t\}$ when G halts, so there are a total of t iterations.

Define $T^{**} = T^* - S_t'$, where S_t' is the union of all elements deleted from the set S_t on all executions of Step 2. Intuitively, T^{**} consists of those elements

that algorithm G "credits" itself with having covered during its execution (as opposed to those elements regarded as "excess" that were covered because G may cover more than the required minimum fraction p). We say that a set S_j is *at capacity* when in Step 2, $|S_j| \geq q$. Note that once S_j reaches capacity, it remains at capacity until it is chosen in Step 4 or until G halts. This is because if l elements are removed from S_j on an execution of Step 4, the value of q in Step 2 will decrease by at least l on the next iteration. Furthermore, since G halts the first time a set at capacity is chosen, and by the above definitions S_t is the last set chosen by G, we have that $T^{**} = \cup_{r=1}^{t} S_r^r$. Thus we have $|S_t'| = |T^*| - pm$ and $|T^{**}| = pm$.

The set S_r^r can be regarded as the set of previously uncovered elements that are added to T^{**} on the rth iteration. We wish to amortize the cost c_r over the elements covered. For each $i \in T^*$, we define a number y_i, which is intuitively the cost we paid to put i in T^*:

$$y_i = \begin{cases} c_r/|S_r^r| & \text{if for some } r,\ i \in S_r^r \\ 0 & i \text{ is not in } T^{**} \end{cases}$$

Since for $i \in T^* - T^{**}$, $y_i = 0$, we have

$$\begin{aligned} \sum_{i \in T^{**}} y_i &= \sum_{i \in T^*} y_i \\ &= \sum_{r=1}^{t} \sum_{i \in S_r^r} y_i \\ &= \sum_{r=1}^{t} c_r \\ &= \sum_{j \in J^*} c_j \\ &= cost_{PC}(J^*). \end{aligned}$$

Thus to bound $cost_{PC}(J^*)$, we now bound $\sum_{i \in T^{**}} y_i$ in two parts, first bounding $\sum_{i \in T^{**} - T_{opt}} y_i$ and then bounding $\sum_{i \in T^{**} \cap T_{opt}} y_i$.

Lemma 5.16

$$\sum_{i \in T^{**} - T_{opt}} y_i \leq (H(m) + 2) opt_{PC}(I).$$

Proof: If $T^{**} \subseteq T_{opt}$ then the lemma follows trivially. We therefore assume $T^{**} \not\subseteq T_{opt}$. Since $|T_{opt}| \geq pm$ and $|T^{**}| = pm$, this implies $T_{opt} - T^{**} \neq \emptyset$. Pick $j \in J_{opt}$ such that

$$\frac{c_j}{|S_j - T^{**}|}$$

is minimized. Now

$$\begin{aligned}
\frac{opt_{PC}(I)}{|T_{opt} - T^{**}|} &= \frac{\sum_{i \in J_{opt}} c_i}{|\cup_{i \in J_{opt}} (S_i - T^{**})|} \\
&\geq \frac{\sum_{i \in J_{opt}} c_i}{\sum_{i \in J_{opt}} |S_i - T^{**}|} \\
&\geq \frac{c_j}{|S_j - T^{**}|}.
\end{aligned}$$

Thus

$$opt_{PC}(I) \geq |T_{opt} - T^{**}| \frac{c_j}{|S_j - T^{**}|}.$$

Let r_0 be the first execution of Step 2 in which $|S_j| > q$ (i.e., S_j reaches capacity on the r_0th iteration). We will analyze the behavior of G before and after the r_0th iteration separately. Let T_0^{**} denote the set of elements that were added to T^{**} prior to the r_0 iteration. For each $i \in T_0^{**} - T_{opt}$, the cost y_i must satisfy

$$y_i \leq \frac{c_j}{|S_j - T^{**}|}$$

because otherwise G would have already added S_j to J^*. Since $|T_{opt} - T^{**}| \geq |T^{**} - T_{opt}|$ we have

$$\begin{aligned}
\sum_{i \in T_0^{**} - T_{opt}} y_i &\leq \sum_{i \in T_0^{**} - T_{opt}} \frac{c_j}{|S_j - T^{**}|} \\
&\leq |T_{opt} - T^{**}| \frac{c_j}{|S_j - T^{**}|} \\
&\leq opt_{PC}(I).
\end{aligned}$$

For iterations $r \geq r_0$, whenever an element i is added to $T_1^{**} = T^{**} - T_0^{**}$, an element is deleted from S_j in Step 2, since S_j is at capacity. We charge y_i to this element as follows:

$$\sum_{i \in T_1^{**} - T_{opt}} y_i \leq \sum_{i \in T_1^{**}} y_i$$

$$
\begin{aligned}
&= \sum_{r=r_0}^{t} \sum_{i \in S_r^r} y_i \\
&\leq \sum_{r=r_0}^{t-1} \sum_{i \in S_r^r} y_i + c_j
\end{aligned}
$$

(because on iteration t, both S_j and S_t are at capacity, so $c_t \leq c_j$)

$$
\leq \sum_{r=r_0}^{t-1} \frac{c_r}{|S_r^r|} |S_j^r - S_j^{r+1}| + c_j
$$

(because since S_j is at capacity, $|S_j^r - S_j^{r+1}| = |S_r^r|$)

$$
\leq \sum_{r=r_0}^{t-1} \frac{c_j}{|S_j^r|} |S_j^r - S_j^{r+1}| + c_j
$$

(because otherwise G would have chosen S_j at time r)

$$
\begin{aligned}
&= c_j \sum_{r=r_0}^{t-1} \frac{1}{|S_j^r|} |S_j^r - S_j^{r+1}| + c_j \\
&\leq c_j H(|S_j|) + c_j \\
&= c_j (H(|S_j|) + 1).
\end{aligned}
$$

Combining the two parts, we have

$$
\begin{aligned}
\sum_{i \in T^{**} - T_{opt}} y_i &= \sum_{i \in T_0^{**} - T_{opt}} y_i + \sum_{i \in T_1^{**} - T_{opt}} y_i \\
&\leq opt_{PC}(I) + c_j (H(m) + 1) \\
&\leq (H(m) + 2) opt_{PC}(I).
\end{aligned}
$$

□(Lemma 5.16)

Lemma 5.17

$$
\sum_{i \in T^{**} \cap T_{opt}} y_i \leq (H(m) + 1) opt_{PC}(I).
$$

Proof: We generalize the idea used by Chvatal [29]. For $j \in J_{opt}$ and $S_j \bigcap T^{**} \neq \emptyset$,

$$\begin{aligned}\sum_{i \in S_j \cap T^{**}} y_i &= \sum_{r=1}^{t} \sum_{i \in S_j \cap S_r^r} y_i \\ &\leq \sum_{r=1}^{t-1} \frac{c_r}{|S_r^r|} |S_j^r - S_j^{r+1}| + c_j\end{aligned}$$

(because the average cost of elements in S_t is lower than in S_j, and we are summing over at most $|S_j|$ elements)

$$\leq \sum_{r=1}^{s} \frac{c_j}{|S_j^r|} |S_j^r - S_j^{r+1}| + c_j$$

(where $s = \min\{\max\{k : S_j^k \neq \emptyset\}, t\}$)

$$\begin{aligned}&= c_j \sum_{r=1}^{s} \frac{1}{|S_j^r|} |S_j^r - S_j^{r+1}| + c_j \\ &\leq c_j H(|S_j|) + c_j \\ &= c_j (H(|S_j|) + 1).\end{aligned}$$

Now by the above,

$$\begin{aligned}\sum_{i \in T^{**} \cap T_{opt}} y_i &\leq \sum_{j \in J_{opt}} \sum_{i \in S_j \cap T^{**}} y_i \\ &\leq \sum_{j \in J_{opt}} (H(m) + 1) c_j \\ &\leq (H(m) + 1) opt_{PC}(I).\end{aligned}$$

□(Lemma 5.17)

Combining Lemmas 5.16 and 5.17, we have

$$\begin{aligned}\sum_{j \in J^*} c_j &= \sum_{i \in T^*} y_i \\ &= \sum_{i \in T^{**}} y_i \\ &= \sum_{i \in T^{**} - T_{opt}} y_i + \sum_{i \in T^{**} \cap T_{opt}} y_i \\ &\leq (H(m) + 2) opt_{PC}(I) + (H(m) + 1) opt_{PC}(I) \\ &= (2H(m) + 3) opt_{PC}(I).\end{aligned}$$

This completes the proof of Theorem 5.15. □

We now use algorithm G as a subroutine in constructing our error-tolerant learning algorithm for M_n^s.

Theorem 5.18 *Let M_n^s be the class of monomials over $x_1, \ldots, x_n$ containing at most s literals. Then*

$$E_{MAL}^{poly}(M_n^s) = \Omega\left(\frac{\epsilon}{s \log \frac{s \log n}{\epsilon}}\right).$$

Proof: We construct an Occam algorithm A for M_n^s that tolerates the desired malicious error rate, and uses the algorithm G for the partial cover problem as a subroutine.

Let $0 \leq \beta < \epsilon/8$, and let $c \in M_n^s$ be the target monomial. A first takes m_N points from the oracle NEG_{MAL}^{β}, where $m_N = O(1/\epsilon \ln 1/\delta + 1/\epsilon \ln |M_n^s|)$ as in the statement of Theorem 5.7. Let S denote the multiset of points received by A from NEG_{MAL}^{β}. For $1 \leq i \leq n$, define the multisets

$$S_i^0 = \{\vec{v} \in S : v_i = 0\}$$

and

$$S_i^1 = \{\vec{v} \in S : v_i = 1\}.$$

We now define a pairing between monomials and partial covers as follows: the literal x_i is paired with the partial cover consisting of the single set S_i^0 and the literal $\overline{x}_i$ is paired with the partial cover consisting of the single set S_i^1. Then any monomial c is paired with the partial cover obtained by including exactly those S_i^0 and S_i^1 that are paired with the literals appearing in c. Note that the multiset $neg(c) \cap S$ contains exactly those vectors that are covered by the corresponding partial cover.

Now with high probability, there must be some collection of the S_i^0 and S_i^1 that together form a $1 - \epsilon/2$ cover of S: namely, if (without loss of generality) the target monomial $c \in M_n^s$ is

$$c = x_1 \cdots x_r \overline{x}_{r+1} \cdots \overline{x}_s$$

then with high probability the sets

$$S_1^0, \ldots, S_r^0, S_{r+1}^1, \ldots, S_s^1$$

form a $1 - \epsilon/2$ cover of S, since for $\beta \leq \epsilon/8$, the probability that the target monomial c disagrees with a fraction larger than $\epsilon/2$ of a sample of size m_N from NEG_{MAL}^{β} can be shown to be smaller than $\delta/2$ by Fact CB2.

Thus, A will input the sets $S_1^0, \ldots, S_n^0, S_1^1, \ldots, S_n^1$ and the value $p = 1 - \epsilon/2$ to algorithm G. The costs for these sets input to G are defined below. However, note that regardless of these costs, if h_G is the monomial paired with the p-cover output by G, then since $|neg(h_G) \cap S| \geq (1 - \epsilon/2)m_N$ (where $neg(h_G) \cap S$ is interpreted as a multiset), $e^-(h_G) < \epsilon$ with high probability by Theorem 5.7. We now show that for β as in the statement of the theorem, we can choose the costs input to G so as to force $e^+(h_G) < \epsilon$ as well.

For any monomial c, let $p(c)$ denote the probability that c disagrees with a vector returned by POS_{MAL}^{β}, and let $cost_{PC}(c)$ denote the cost of the partial cover that is paired with c. To determine the costs of the sets input to G, A next samples POS_{MAL}^{β} enough times (determined by application of Facts CB1 and CB2) to obtain an estimate for $p(x_i)$ and $p(\overline{x}_i)$ for $1 \leq i \leq n$ that is accurate within a multiplicative factor of 2 — that is, if $\hat{p}(x_i)$ is the estimate computed by A, then $p(x_i)/2 \leq \hat{p}(x_i) \leq 2p(x_i)$ with high probability for each i. The same bounds hold for the estimate $\hat{p}(\overline{x}_i)$. Then the cost for set S_i^0 input to G by A is $\hat{p}(x_i)$ and the cost for set S_i^1 is $\hat{p}(\overline{x}_i)$.

Note that for any monomial $c = x_1 \cdots x_r \overline{x}_{r+1} \cdots \overline{x}_s$, we have with high probability

$$\begin{aligned} p(c) &\leq p(x_1) + \cdots + p(x_r) + p(\overline{x}_{r+1}) + \cdots + p(\overline{x}_s) \\ &\leq 2\hat{p}(x_1) + \cdots + 2\hat{p}(x_r) + 2\hat{p}(\overline{x}_{r+1}) + \cdots + 2\hat{p}(\overline{x}_s) \\ &= 2cost_{PC}(c). \end{aligned}$$

By Theorem 5.18, the output h_G of G must satisfy

$$cost_{PC}(h_G) \leq (H(m_N) + 2)cost_{PC}(c_{opt}) \tag{5.3}$$

where c_{opt} is the monomial paired with a p-cover of minimum cost. But for the target monomial c we have

$$p(c) \leq \beta \tag{5.4}$$

$$2sp(c) \geq cost_{PC}(c) \tag{5.5}$$

where Equation 5.4 holds absolutely and Equation 5.5 holds with high probability, since c contains at most s literals.

From Equations 5.3, 5.4 and 5.5 we obtain with high probability

$$\begin{aligned} p(h_G) &\leq 2cost_{PC}(h_G) \\ &\leq 2(H(m_N)+2)cost_{PC}(c_{opt}) \\ &\leq 2(H(m_N)+2)cost_{PC}(c) \\ &\leq 4sp(c)(H(m_N)+2) \\ &\leq 4s\beta(H(m_N)+2). \end{aligned}$$

Thus, if we set

$$\beta = \frac{\epsilon}{4s(H(m_N)+2)} = \Omega(\frac{\epsilon}{s \log m_N})$$

then $e^+(h_G) < \epsilon$ with high probability by Theorem 5.7. We can remove the dependence of β on δ by method used in the proof of Theorem 5.12, thus obtaining an error rate of

$$\Omega\left(\frac{\epsilon}{s \log \frac{s \log n}{\epsilon}}\right)$$

completing the proof. □

As an example, if $s = \sqrt{n}$ then Theorem 5.18 gives

$$E^{poly}_{MAL}(M^{\sqrt{n}}_n) = \Omega(\frac{\epsilon}{\sqrt{n} \log \frac{n}{\epsilon}})$$

as opposed to the the bound of $\Omega(\epsilon/n \ln \epsilon/n)$ of Theorem 5.13.

Littlestone [73] shows that the Vapnik-Chervonenkis dimension of M^s_n is $\Theta(s \ln(1+n/s))$. Since the algorithm of Valiant [93] can be modified to have optimal sample complexity for M^s_n, by applying Theorem 5.12 to this modified algorithm we obtain

$$E^{poly}_{MAL}(M^s_n) = \Omega\left(\frac{\epsilon \ln(\frac{s}{\epsilon} \ln(1+\frac{n}{s}))}{s \ln(1+\frac{n}{s})}\right).$$

This lower bound on $E^{poly}_{MAL}(M^s_n)$ is incomparable to that of Theorem 5.18. We may decide at run time which algorithm will tolerate the larger error rate, thus giving

$$E^{poly}_{MAL}(M^s_n) = \Omega\left(\min\left(\frac{\epsilon \ln(\frac{s}{\epsilon}\ln(1+\frac{n}{s}))}{s\ln(1+\frac{n}{s})}, \frac{\epsilon}{s\log\frac{s\log n}{\epsilon}}\right)\right).$$

By using transformation techniques similar to those described in Section 4.3 it can be shown that the algorithm of Theorem 5.18 (as well as that obtained from Theorem 5.12) can be used to obtain an improvement in the error rate over the negative-only algorithm of Valiant [94] for the class $k\text{DNF}_{n,s}$ of kDNF formulae with at most s terms. Briefly, the appropriate transformation regards a kDNF formulae as a 1DNF formulae in a space of $\Theta(n^k)$ variables, one variable for each of the possible terms (monomials) of length at most k.

5.5 Limits on efficient learning with errors

In Section 5.3, we saw that there was an absolute bound of $\epsilon/(1+\epsilon)$ on the achievable malicious error rate for most interesting representation classes. It was also argued there that, at least for our finite representation classes over $\{0,1\}^n$, this bound could always be achieved by a super-polynomial time exhaustive search learning algorithm. Then in Section 5.4 we gave polynomial-time learning algorithms that in some cases achieved the optimal error rate $O(\epsilon)$, but in other cases fell short. These observations raise the natural question of whether for some classes it is possible to prove bounds stronger than $\epsilon/(1+\epsilon)$ on the malicious error rate for learning algorithms constrained to run in polynomial time. In particular, for parameterized representation classes, under what conditions must the error rate tolerated by a polynomial-time learning algorithm decrease as the number of variables n increases? If we informally regard the problem of learning with malicious errors as an optimization problem where the objective is to maximize the achievable error rate in polynomial time, and $\epsilon/(1+\epsilon)$ is the optimal value, then we might expect such hardness results to take the form of hardness results for the approximation of *NP*-hard optimization problems. This is the approach we pursue in this section.

By reducing standard combinatorial optimization problems to learning problems, we state theorems indicating that efficiently learning with an error

rate approaching $\Theta(\epsilon)$ is eventually as hard as approximations for *NP*-hard problems.

In Section 5.4 we gave an error-tolerant algorithm for learning monomials by monomials that was based on an approximation algorithm for a generalization of set cover. Our next theorem gives a reduction in the opposite direction: an algorithm learning monomials by monomials and tolerating a malicious error rate approaching $\Theta(\epsilon)$ can be used to obtain an improved approximation algorithm for set cover.

Theorem 5.19 *Let M_n be the class of monomials over $x_1, \ldots, x_n$. Suppose there is a polynomial-time learning algorithm A for M_n using hypothesis space M_n such that*

$$E_{MAL}^{poly}(M_n, A) = \frac{\epsilon}{r(n)}.$$

Then there is a polynomial-time algorithm for the weighted set cover problem that outputs (with high probability) a cover whose cost is at most $2r(n)$ times the optimal cost, where n is the number of sets.

Proof: We describe an approximation algorithm A' for set cover that uses the learning algorithm A as a subroutine. Given an instance I of set cover with sets $S_1, \ldots, S_n$ and costs $c_1, \ldots, c_n$, let $J_{opt} \subseteq \{1, \ldots, n\}$ be an optimal cover of $T = \cup_{j=1}^{n} S_j = \{1, \ldots, m\}$, where we identify a cover $\{S_{j_1}, \ldots, S_{j_s}\}$ with its index set $\{j_1, \ldots, j_s\}$. Let $cost_{SC}(J)$ denote the set cover cost of any cover J of T, and let $opt_{SC}(I) = cost_{SC}(J_{opt})$. As in the proof of Theorem 5.18, we pair a cover $\{j_1, \ldots, j_s\}$ of T with the monomial $x_{j_1} \cdots x_{j_s}$ over the variables $x_1, \ldots, x_n$. Let c_{opt} be the monomial paired with the optimal cover J_{opt}.

The goal of A' is to simulate algorithm A with the intention that c_{opt} is the target monomial, and use the monomial h_A output by A to obtain the desired cover of T. The examples given to A on calls to NEG_{MAL}^{β} during this simulation will be constructed so as to guarantee that the collection of sets paired with h_A is actually a cover of T, while the examples given to A on calls to POS_{MAL}^{β} guarantee that this cover has a cost within a multiplicative factor of $2r(n)$ of the optimal cost.

We first describe the examples A' generates for A on calls to NEG_{MAL}^{β}. For each $i \in T$, let $\vec{u}_i \in \{0,1\}^n$ be the vector whose jth bit is 0 if and only

if $i \in S_j$, and let the multiset U be $U = \cup_{i \in T}\{\vec{u}_i\}$. Then $\{j_1, \ldots, j_s\}$ is a cover of T if and only if $U \subseteq neg(x_{j_1} \cdots x_{j_s})$. In particular, we must have $U \subseteq neg(c_{opt})$. Thus, define the target distribution D^- for c_{opt} to be uniform over U. Note that this distribution can be generated in polynomial time by A'. On calls of A to NEG^{β}_{MAL}, A' will simply draw from D^-; thus if we regard c_{opt} as the target monomial, there are no errors in the negative examples. A' will simulate A with accuracy parameter $\epsilon \leq 1/|U|$, thus forcing A to output an hypothesis monomial h_A such that $U \subseteq neg(h_A)$; by the above argument, this implies that the collection of sets paired with the monomial h_A is a cover of T. Note that $|U|$ (and therefore $1/\epsilon$) may be super-polynomial in n, but it is polynomial in the size of the instance I.

We now describe the examples A' generates for A on calls to POS^{β}_{MAL}. Instead of defining the target distribution D^+ for c_{opt}, we define an *induced* distribution I^+ from which the oracle POS^{β}_{MAL} will draw. Thus, I^+ will describe the joint behavior of the underlying distribution D^+ on c_{opt} and an adversary generating the malicious errors. For each $1 \leq j \leq n$, let $\vec{v}_j \in \{0,1\}^n$ be the vector whose jth bit is 0, and all other bits are 1. Let $I^+(\vec{v}_j) = c_j$ for each j, where c_j is the cost of the set S_j, and we assume without loss of generality that $\sum_{j=1}^{n} c_j \leq \epsilon/r(n)$ (if not, we can normalize the weights without changing the relative costs of covers). We complete the definition of I^+ by letting $I^+((1,\ldots,1)) = 1 - \sum_{j=1}^{n} c_j$. Then the probability that a monomial $x_{i_1} \cdots x_{i_s}$ disagrees with a point drawn from POS^{β}_{MAL} is exactly $c_{i_1} + \cdots + c_{i_s}$, the cost of the corresponding cover. Thus since $opt_{SC}(I) \leq \sum_{j=1}^{n} c_j \leq \epsilon/r(n) = \beta$, I^+ is an induced distribution for c_{opt} with malicious error rate β. Note that I^+ can be generated by A' in polynomial time. When A requests an example from POS^{β}_{MAL}, A' will simply draw from I^+.

A' will run algorithm A many times with the oracles POS^{β}_{MAL} and NEG^{β}_{MAL} for c_{opt} described above, each time with a progressively smaller value for the accuracy parameter, starting with $\epsilon = 1/|U|$.

Now if $opt_{SC}(I) << \epsilon/r(n)$, then algorithm A may output a monomial h_A whose corresponding cover has a cost much larger than $4r(n) \cdot opt_{SC}(I)$, since h_A is only guaranteed to satisfy $e^+(h_A) < \epsilon$. We solve this problem by repeated scaling: A' first runs algorithm A with the oracles POS^{β}_{MAL} and NEG^{β}_{MAL} as they have been described. After each run, A' divides the accuracy parameter ϵ by 2, so that on some run $\epsilon/2r(n) \leq opt_{SC}(I) \leq \epsilon/r(n)$. On this

run, we may regard I^+ as an induced distribution on the positive examples of c_{opt}, with malicious error rate at most $\beta = \epsilon/r(n) \leq 2opt_{SC}(I)$. Then the error $e^+(h_A)$ on the underlying distribution D^+ over $pos(c_{opt})$ is at most $\epsilon \leq 2r(n)opt_{SC}(I)$. The desired cover is thus the one paired with the monomial h_A. Note that without knowing c_{opt}, we have no way of knowing what the underlying target distribution D^+ is, but it is enough to know that I^+ is a "close" distribution. The only problem with the simulation described occurs when $opt_{SC}(I) << \sum_{j=1}^{n} c_j$, in which case it may take a super-polynomial number of runs of A to guarantee $\epsilon/2r(n) \leq opt_{SC}(I) \leq \epsilon/r(n)$. We solve this by preprocessing: before running the described simulation, A' runs the greedy approximation algorithm analyzed by Chvatal [29] on the set cover instance I, and removes any set whose cost is larger than the entire cost of the greedy cover. Then for the new (smaller) instance I', every cost is within a multiplicative factor of $\log m$ of every other cost. □

Thus, if $r(n) << \log n$, then Theorem 5.19 says that a polynomial time algorithm A for M_n (using hypothesis space M_n) tolerating $E^{poly}_{MAL}(M_n, A) \geq \epsilon/r(n)$ would imply a significant breakthrough in approximation algorithms for set cover, since the best algorithm for this problem remains the greedy method analyzed by Chvatal and others [29, 56, 75, 77]. Note that the proof of Theorem 5.19 in fact shows the result holds for the class of *monotone* monomials.

Theorem 5.18 took an approximation algorithm for an optimization problem (the partial cover problem), and used it as a subroutine in obtaining an error-tolerant learning algorithm for M_n^s. Theorem 5.19 proved that when learning algorithms are restricted to hypothesis class M, any learning algorithm for M yields an algorithm for set cover with only a constant factor blowup in the approximation. Thus, we see that there are strong ties between learning with errors and approximating combinatorial optimization problems. Our goal now is to generalize and strengthen these ideas. We show that for any representation class C, the problem of learning C with errors is equivalent to a combinatorial optimization problem with only a constant factor blowup in the approximation in each direction of the reduction.

For domain X, define a *balanced sample* of X to be a sample

$$S = < x_1, 1 >, \ldots, < x_m, 1 >, < y_1, 0 >, \ldots, < y_m, 0 >$$

where $x_i, y_i \in X, 1 \leq i \leq m$. If C is a representation class over X and $c \in C$,

define

$$cost_{MD}(c,S) = |\{< x_i, 1 >\in S : x_i \in neg(c)\}| \\ +|\{< y_i, 0 >\in S : y_i \in pos(c)\}| + 1.$$

Thus, $cost_{MD}(c,S)$ is simply one more than the number of disagreements between the balanced sample S and the representation c. We now define the following optimization problem for C:

The Minimize Disagreements Problem for C (denoted $MD(C)$):

Input: Balanced sample S of X.

Output: Representation $c \in C$ such that $cost_{MD}(c,S)$ is minimized.

Theorem 5.20 *Let C be a representation class over X. If there exists a polynomial-time algorithm A' for $MD(C)$ that outputs $h_{A'} \in C$ such that $cost_{MD}(h_{A'},S)$ is at most r times the optimal cost, then C is learnable by C by an algorithm A that runs in time polynomial in $1/\epsilon, 1/\delta$ and $\ln|C|$, and satisfies*

$$E^{poly}_{MAL}(C,A) \geq \frac{\epsilon}{8r}.$$

Conversely, if algorithm A learns C by C in polynomial time with error rate $E^{poly}_{MAL}(C,A) \geq \epsilon/r$, then there exists a polynomial-time algorithm A' for $MD(C)$ that outputs (with high probability) $h_{A'} \in C$ such that $cost_{MD}(h_{A'},S)$ is at most $2r$ times the optimal cost.

Proof: Let S be a balanced sample of X, and let A' be an approximation algorithm for $MD(C)$ such that the output $h_{A'}$ satisfies $cost_{MD}(h_{A'},S) \leq r \cdot opt_{MD}(S)$, where

$$opt_{MD} = \min_{h\in C}(cost_{MD}(h,S)).$$

Let $\beta = \epsilon/8r$. To learn C by C in polynomial time with error rate β, we take m random examples $x_1,\ldots,x_m$ from the oracle POS^{β}_{MAL} and m random examples $y_1,\ldots,y_m$ from the oracle NEG^{β}_{MAL}, where m is as in the statement of Theorem 5.7. Let S be the balanced sample consisting of the x_i and y_j. Now with probability at least $1-\delta$, the target representation $c \in C$ disagrees with fewer than $4\beta m$ elements of S by Fact CB2, so $opt_{MD}(S) \leq 4\beta m$ with

high probability. Thus, algorithm A', when given S as input, will satisfy $cost_{MD}(h_{A'}, S) \leq r(4\beta m) = (\epsilon/2)m$. This implies that $h_{A'}$ can disagree with at most a fraction $\epsilon/2$ of the x_i and at most a fraction $\epsilon/2$ of the y_i. By Theorem 5.7, $h_{A'}$ is an ϵ-good hypothesis with high probability.

For the other direction, we use an algorithm A for learning C by C with $\beta = \epsilon/r$ to obtain an approximation algorithm for $MD(C)$ as follows: given the balanced sample S, let $h_{opt} \in C$ be such that $cost_{MD}(h_{opt}, S) = opt_{MD}(S)$ and assume without loss of generality that $m/r \geq opt_{MD}(S)$ (otherwise *any* hypothesis has cost at most $2r$ times the optimal). Define

$$c_0 = \max\{|\{x_i \in S : x_i \in neg(h_{opt})\}|, |\{y_i \in S : y_i \in pos(h_{opt})\}|\}.$$

Note that $opt_{MD}(S) \geq c_0 \geq opt_{MD}(S)/2$. Now let I^+ be the uniform distribution over the x_i, and let I^- be the uniform distribution over the y_i. Then I^+ and I^- can be regarded as induced distributions for h_{opt} with error rate $\beta' = c_0/m$. I^+ is induced by the joint behavior of the uniform distribution D^+ over $\{x_i \in S : x_i \in pos(h_{opt})\}$, and an adversary that draws a point uniformly from $\{x_i \in S : x_i \in neg(h_{opt})\}$; I^- can be decomposed over the y_i in a similar fashion.

Algorithm A' runs algorithm A many times, starting with accuracy parameter $\epsilon = 1$, and drawing from I^+ on each call to POS^{β}_{MAL} and from I^- on each call to NEG^{β}_{MAL}. Note that if h_A is an ϵ-good hypothesis with respect to D^+ and D^-, then we have $cost_{MD}(h_A, S) \leq 2\epsilon m + opt_{MD}(S)$. After each run, A' divides ϵ by 2. On some run of A, $\epsilon/r \leq c_0/2m$, and for this run we have $cost_{MD}(h_A, S) \leq (r+1)opt_{MD}(S) \leq 2r\,opt_{MD}(S)$, as desired. □

Note that this equivalence as it is stated is *representation-based* (see Chapters 3 and 7). With more technical definitions for the problem $MD(C, H)$, we can in fact give a straightforward generalization of Theorem 5.20 for the problem of learning C by H in the presence of malicious errors, giving an equivalent optimization problem. In addition to simplifying the analysis of learning with errors in the distribution-free model — we only need to look at the equivalent optimization problem — these results allow us to weaken our restrictions on the adversary generating the errors. In particular, since there is no guarantee in the Minimize Disagreements problem on how the errors in the input sample are generated, it can be shown that the adversary gains no power by being allowed to see all coin flips of the learning algorithm, and all examples to be received by the learning algorithm *before* he generates the errors. This allows

our model to incorporate faults such as *error bursts*, where all examples are in error for a short amount of time.

Figures 5.1 and 5.2 summarize some of the results in this chapter.

	$E_{MAL,+}(C)$ and $E_{MAL,-}(C)$	$E_{MAL}(C)$	$E_{CN,+}(C)$ and $E_{CN,-}(C)$	$E_{CN}(C)$
Upper bound on the optimal error rate	$\epsilon/(t(C)-1)$	$\epsilon/(1+\epsilon)$	$\epsilon/(1+\epsilon), \epsilon/(t(C)-1)$	$1/2$ [12]

Figure 5.1: Summary of general upper bounds on the optimal error rates for the malicious and noise models. We denote by $t(C)$ the largest value of t such that C is (positive or negative) t-splittable.

Class C	$E^{poly}_{MAL,+}(C)$ $E^{poly}_{MAL,-}(C)$ and $E^{poly}_{CN,+}(C)$ $E^{poly}_{CN,-}(C)$	$E^{poly}_{MAL}(C)$	$E^{poly}_{CN}(C)$
Upper bound M_n	$\Theta(\epsilon/n)$	$O(\epsilon)$	$\Theta(1)$
Lower bound	[94]	$\Omega(\ln(n/\epsilon)\epsilon/n)$	[12]
Upper bound M^s_n	$\Theta(\epsilon/n)$	$O(\epsilon)$ $\Omega((\epsilon/s)\ln((s/\epsilon)\ln(1+n/s))/\ln(1+n/s))$	$\Theta(1)$
Lower bound	[94]	$\Omega((\epsilon/s)(1/\log((s\ln n)/\epsilon)))$	[12]
Upper bound SF_n	$\Theta(\epsilon/n)$	$\Theta(\epsilon)$	$\Theta(1)$
Lower bound			

Figure 5.2: Summary of upper and lower bounds on the optimal polynomial time error rate (malicious and noise models) for the classes of monomials M_n, monomials of length s M^s_n, and symmetric functions SF_n.

6

Lower Bounds on Sample Complexity

6.1 Introduction

In this chapter we consider sample complexity as a resource in its own right. We are interested in the following question: regardless of the amount of *computation* time required, what is the least number of *examples* required to achieve learning in the distribution-free model? Thus, we consider sample complexity from an *information-theoretic* viewpoint.

The study of sample complexity is important for several reasons. Even though our emphasis throughout this book is on the computational expense of learning, lower bounds on the required sample size immediately translate to lower bounds on computation time. Thus determining the sample complexity of learning a representation class is the first step towards determining if the class can be learned *efficiently*. This issue is particularly pressing for classes over infinite domains such as the reals, where it is known that any class with infinite Vapnik-Chervonenkis dimension cannot be learned in any amount of time for reasons of required sample size [25].

A second and equally important motivation is the fact that in many real applications, examples are either in limited supply or expensive to obtain, and thus we must seek algorithms that are as example-efficient as possible (perhaps even at the expense of added computation time). A typical application where examples are available but expensive to obtain is DNA sequencing; an even more constrained setting is that of archeological evidence, where the number

of examples is essentially *fixed*. While at first glance it may seem that under such circumstances an asymptotic lower bound on distribution-free learning involving the parameters ϵ and δ is vacuous, it becomes meaningful if we instead use the lower bound to determine what accuracy and confidence we *can* expect to achieve with the *given* sample size. Recently Goldman et al. have been similarly motivated to study the sample complexity of weak learning [43].

We begin by presenting a lower bound on the number of examples required by any algorithm that learns the class of monomials from negative examples only. This lower bound proves that the number of negative examples required to learn monomials is super-polynomial in the number of attributes, regardless of the amount of time taken by the learning algorithm. This is in contrast to the efficient positive-only learning algorithm due to Valiant [93] which was discussed in detail in Section 2.3, and which requires a number of positive examples that is only linear in the number of attributes. We are able to apply this result to show that certain representation classes require both positive and negative examples for polynomial-time learnability.

We then present a general lower bound for learning *any* representation class C. This general lower bound is based on $\text{VCD}(C)$ and improves the previous best general lower bound of Blumer et al. [25]. A number of applications to specific representation classes are presented. These applications prove that almost all of the existing learning algorithms in the distribution-free model have optimal or near-optimal sample complexity.

6.2 Lower bounds on the number of examples needed for positive-only and negative-only learning

So far we have discussed a number of polynomial-time learning algorithms that require only positive examples or only negative examples. Among other issues, this raises the question of whether every polynomially learnable class is polynomially learnable either from positive examples only or from negative examples only.

In this section we prove a super-polynomial lower bound on the number of

negative examples required for learning monomials. The proof can actually be tightened to give a strictly exponential lower bound, and the proof technique has been generalized by Gereb-Graus [40]. Our bound is information-theoretic in the sense that it holds regardless of the computational complexity and hypothesis class of the negative-only learning algorithm. By duality, we obtain lower bounds on the number of positive examples needed for learning disjunctions, and it follows from our proof that the same bound holds for learning from negative examples any class properly containing monomials (e.g., kCNF) or for learning from positive examples any class properly containing disjunctions (e.g., kDNF). In fact, these results hold even for the monotone restrictions of these classes.

We apply our lower bound to answer negatively the question raised above, showing that for polynomial-time learning, the class $k\text{CNF} \vee k\text{DNF}$ (shown to be polynomially learnable using both positive and negative examples in Chapter 4) *requires* both positive and negative examples. This is another demonstration of the power of using both types of examples, along the lines of the results on learning with errors in Chapter 5.

Theorem 6.1 *Let A be a negative-only learning algorithm for the class of monotone monomials over the variables $x_1, \ldots, x_n$, and fix $\epsilon, \delta \leq 1/n$. Let S_A^- denote the number of negative examples required by A. Then for any constant $k > 0$, $S_A^-(n) = \Omega(n^k)$. This holds even when the target distribution D^- is uniform over the negative examples.*

Proof: Fix $\epsilon = \delta \leq 1/n$. Assume for contradiction that A is a negative-only learning algorithm for monotone monomials such that for ϵ and δ fixed, $S_A^-(n) = n^k$ for some constant k. We call a monomial *monotone dense* if it is monotone and contains at least $n/2$ of the variables $x_1, \ldots, x_n$. Let T be an ordered sequence of (not necessarily distinct) vectors from $\{0,1\}^n$ such that $|T| = n^k$, and let Ψ be the set of all such sequences. If c is a monotone dense monomial, then define $\vec{u}_c \in \{0,1\}^n$ to be the unique vector such that $\vec{u}_c \in pos(c)$ and $\vec{u}_c$ has the fewest bits set to 1. We say that $T \in \Psi$ is a *legal negative* sample for c if T contains no vector $\vec{v}$ such that $\vec{v} \in pos(c)$.

We first define the hard target distributions for a monotone dense target monomial c. Let $D^+(\vec{u}_c) = 1$ and let D^- be uniform over $neg(c)$. Note that $\vec{u}_c \notin pos(h)$ implies $e^+(h) = 1$, so any ϵ-good h must satisfy $\vec{u}_c \in pos(h)$. For

$T \in \Psi$ and c a monotone dense monomial, define the predicate $P(T, c)$ to be 1 if and only if T is a legal negative sample for c and when T is received by A as a sequence of negative examples for c from NEG, A outputs an hypothesis h_A such that $\vec{u}_c \in pos(h_A)$. Note that this definition assumes that A is deterministic. To allow probabilistic algorithms, we simply change the definition to $P(T, c) = 1$ if and only if T is a legal negative sample for c, and when T is given to A, A outputs an hypothesis h_A such that $\vec{u}_c \in pos(h_A)$ with probability at least $1/2$, where the probability is taken over the coin tosses of A.

Now suppose we draw $\vec{v}$ uniformly at random from $\{0,1\}^n$. Then for any monotone dense monomial c we have

$$\mathbf{Pr}_{\vec{v} \in \{0,1\}^n}[\vec{v} \in pos(c)] \leq 2^{n/2} \frac{1}{2^n} = \frac{1}{2^{n/2}}$$

since at most $2^{n/2}$ vectors can satisfy a monotone dense monomial. Thus, if we draw n^k points uniformly at random from $\{0,1\}^n$, the probability that we draw *some* point satisfying c is at most $n^k/2^{n/2} \leq 1/2$ (for n large enough). By this analysis, we conclude that the number of $T \in \Psi$ that are legal negative for c must be at least $|\Psi|/2$. Since D^- is uniform and A is a learning algorithm, at least $(|\Psi|/2)(1-\delta) = (|\Psi|/2)(1-\epsilon)$ of these must satisfy $P(T, c) = 1$. Let $M(n)$ be the number of montone dense monomials over n variables. Then summing over all monotone dense monomials, we obtain

$$\frac{|\Psi|}{2}(1-\epsilon)M(n) \quad \leq \quad \sum_{T \in \Psi} N(T)$$

where $N(T)$ is defined to be the number of monotone dense monomials satisfying $P(T, c) = 1$. From this inequality, and the fact that $N(T)$ is always at most $M(n)$, we conclude that at least $1/8$ of the $T \in \Psi$ must satisfy $N(T) \geq 1/8(1-\epsilon)M(n)$. Since D^- is uniform, and since at least a fraction $1 - n^k/2^{n/2}$ of the $T \in \Psi$ are legal negative for the target monomial c, A has probability at least $1/16$ of receiving a T with such a large $N(T)$ for n large enough. But then the hypothesis h_A output by A has at least $1/8(1-\epsilon)M(n)$ positive examples by definition of the predicate P. Since the target monomial c has at most $2^{n/2}$ positive examples,

$$e^-(h_A) \geq \frac{\frac{1}{8}(1-\epsilon)M(n) - 2^{n/2}}{2^n}$$

and this error must be less than ϵ. But this cannot be true for ϵ a small enough constant and n large enough. Thus, A cannot achieve arbitrarily small error on monotone dense monomials, and the theorem follows. □

An immediate consequence of Theorem 6.1 is that monomials are not polynomially learnable from negative examples (regardless of the hypothesis class). This is in contrast to the fact that monomials are polynomially learnable (by monomials) from positive examples [93]. It also follows that any class that contains the class of monotone monomials (e.g., kCNF) is not polynomially learnable from negative examples.

By duality we have the following lower bound on the number of positive examples needed for learning monotone disjunctions:

Corollary 6.2 *Let A be a positive-only learning algorithm for the class of monotone disjunctions (that is, monotone* 1DNF*) over the variables $x_1, \ldots, x_n$, and fix $\epsilon, \delta \leq 1/n$. Let S_A^+ denote the number of positive examples required by A. Then for any constant $k > 0$, $S_A^+(n) = \Omega(n^k)$. This holds even when the target distribution D^+ is uniform over the positive examples.*

Further, Theorem 6.1 implies that the polynomially learnable representation classes kCNF $\vee$ kDNF and kCNF $\wedge$ kDNF of Corollaries 4.3 and 4.4 require both positive and negative examples for polynomial learnability:

Corollary 6.3 *For any fixed k, any polynomial-time learning algorithm for the representation class k*CNF $\vee$ kDNF *requires both positive and negative examples.*

Corollary 6.4 *For any fixed k, any polynomial-time learning algorithm for the representation class k*CNF $\wedge$ kDNF *requires both positive and negative examples.*

By similar reasoning we obtain the same result for decision lists, for which there is a polynomial-time learning algorithm due to Rivest [84]:

Corollary 6.5 *For any fixed k, any polynomial-time learning algorithm for the representation class k*DL *requires both positive and negative examples.*

We also note that it is possible to obtain results showing that monomials are not polynomially learnable from negative examples that are weaker than Theorem 6.1 but have simpler proofs. For instance, since 2-term DNF is not learnable by 2-term DNF unless $NP = RP$, it follows from Theorem 4.1 that monomials are not polynomially learnable by monomials from negative examples unless $NP = RP$ (these results do not provide any lower bound on sample complexity). However, Theorem 6.1 gives a lower bound on the sample size for negative-only algorithms that holds regardless of the hypothesis class, and is independent of any complexity-theoretic assumption.

6.3 A general lower bound on the number of examples needed for learning

In this section we give a general theorem providing a lower bound on the number of examples needed for learning any representation class. This lower bound is based on the Vapnik-Chervonkis dimension of the representation class, and improves the lower bound first given by Blumer et al. [25]. It is again information-theoretic (i.e., independent of computational complexity or hypothesis class), and is the best general lower bound possible, since in many important cases it is tight.

We begin by proving the main result of this section, a lower bound of $\Omega(\text{VCD}(C)/\epsilon)$ on the number of examples needed; this bound essentially improves separate bounds of $1/\epsilon$ and $\text{VCD}(C)$ given by Blumer et al. [25] to a single bound proportional to the product $\text{VCD}(C)/\epsilon$. We then combine this result with the lower bound of Blumer et al. to obtain our final lower bound in Theorem 6.10.

Theorem 6.6 *Let C be a representation class over X such that $\text{VCD}(C) \geq 2$. Fix $0 < \epsilon \leq 1/32$, $0 < \delta \leq 1/1000$. Then any learning algorithm A for C must use sample size*

$$S_A(\epsilon, \delta) \geq \frac{\text{VCD}(C) - 2}{128\epsilon} = \Omega\left(\frac{\text{VCD}(C)}{\epsilon}\right).$$

Proof: Let the set $X_0 = \{x_+, x_-, x_1, \ldots, x_d\} \subseteq X$ be shattered by C, where $d = \text{VCD}(C) - 2$. In the target distributions we construct, the points in X_0 will be the only points with nonzero probability; therefore we assume without loss of generality that $X = X_0$ and $\sigma(C) = 2^{X_0}$ (recall that σ maps each representation to the concept it represents). We will further restrict our attention to representations $c \in C$ such that $x_+ \in pos(c)$ and $x_- \in neg(c)$; we let C_0 denote the set of all such representations in C. Note that $\{x_1, \ldots, x_d\}$ is shattered by C_0. For a target representation $c \in C_0$, suppose $pos(c) = \{x_+, x_{i_1}, \ldots, x_{i_l}\}$ and $neg(c) = \{x_-, x_{i_{l+1}}, \ldots, x_{i_d}\}$, where $x_{i_j} \in \{x_1, \ldots, x_d\}$ for $1 \leq j \leq d$. We then define the following target distributions for c:

$$\begin{aligned} D_c^+(x_+) &= 1 - 32\epsilon \\ D_c^+(x_{i_j}) &= \frac{32\epsilon}{l}, 1 \leq j \leq l \end{aligned}$$

and

$$\begin{aligned} D_c^-(x_-) &= 1 - 32\epsilon \\ D_c^-(x_{i_j}) &= \frac{32\epsilon}{d-l}, l+1 \leq j \leq d. \end{aligned}$$

If $pos(c) = \{x_+\}$ (respectively, if $neg(c) = \{x_-\}$), then we define $D_c^+(x_+) = 1$ (respectively, $D_c^-(x_-) = 1$).

We assume without loss of generality that randomized learning algorithms have a separate input tape for receiving a random bit string r. Let A be a learning algorithm for C_0 using sample size $S_A(\epsilon, \delta)$ and requiring $R_A(\epsilon, \delta)$ random bits. We set $m = S_A(\epsilon, \delta)$ and $k = R_A(\epsilon, \delta)$.

For a sample S, let $C_0(S)$ denote the set of representations in C_0 that are consistent with S. Fix $\epsilon \leq 1/32$ and $\delta \leq 1/1000$. Then once the sample S received by A and the random string r are fixed, the hypothesis h_A is determined. Let $A(S, r)$ denote the hypothesis h_A output by A on inputs S and r. For $c \in C_0$, we define the variable $e_A(S, r, c)$ by $e_A(S, r, c) = e_c^+(A(S, r)) + e_c^-(A(S, r))$ if $c \in C_0(S)$, and $e_A(S, r, c) = 8\epsilon$ if $c \notin C_0(S)$. Similarly, we define $w_A(S, r, c)$ to be the number of points in $\{x_1, \ldots, x_d\}$ on which $A(S, r)$ and c disagree if $c \in C_0(S)$, and $w_A(S, r, c) = d/4$ if $c \notin C_0(S)$.

Let S be a sample consisting of m examples of some $c \in C_0$. If S contains fewer than $d/2$ distinct points from $\{x_1, \ldots, x_d\}$, we call S a *bad* sample. Let

B_m denote the set of all bad samples of size m, and let $B_m(c)$ denote the set of all samples S such that $S \in B_m$ and $c \in C_0(S)$. We will now show that there is some target representation $c_0 \in C_0$ such that if A receives a sample $S \in B_m(c_0)$, then with probability at least $1/11$, the hypothesis $h_A = A(S,r)$ is ϵ-bad with respect to the target distributions $D^+_{c_0}$ and $D^-_{c_0}$.

Lemma 6.7 *For some $c_0 \in C_0$,*

$$\mathbf{Pr}_{S \in B_m(c_0), r \in \{0,1\}^k}[e^+_c(A(S,r)) + e^-_c(A(S,r)) \geq 2\epsilon] > \frac{1}{11}.$$

Proof: Let r be a fixed random input string of length k for A, and fix $S \in B_m$. Let l be the number of distinct elements of $\{x_1, \ldots, x_d\}$ appearing in S. Note that $l \leq d/2$ since $S \in B_m$. Thus there are exactly 2^{d-l} representations in $C_0(S)$, since $\{x_1, \ldots, x_d\}$ is shattered by C_0. Let $x \in \{x_1, \ldots, x_d\}$ be one of the $d - l$ points that does not appear in S. Then half of the representations in $C_0(S)$ will agree with c_0 on x, and half will disagree with c_0 on x, since S is shattered by c_0. Thus, for fixed r and fixed $S \in B_m$, we have shown

$$\mathbf{E}_{c \in C_0(S)}[w_A(S,r,c)] \geq \frac{d-l}{2} \geq \frac{d}{4}.$$

The first inequality comes from the fact that the hypothesis $h_A = A(S,r)$ may also be incorrect on points that did appear in S, and the second inequality from the fact that $l \leq d/2$. Since $w_A(S,r,c) = d/4$ for $c \notin C_0(S)$ by definition of $w_A(S,r,c)$, we have in fact shown, again for fixed r and fixed $S \in B_m$,

$$\mathbf{E}_{c \in C_0}[w_A(S,r,c)] \geq \frac{d}{4}. \tag{6.1}$$

Since we have restricted our attention to the class C_0, we may without loss of generality assume that A is always correct on the points x_+ and x_-, so for $c \in C_0(S)$ we have

$$e_A(S,r,c) \geq \frac{32\epsilon}{d} w_A(S,r,c) \tag{6.2}$$

since each point in $\{x_1, \ldots, x_d\}$ has probability at least $32\epsilon/d$ in either D^+_c or D^-_c. Also, for $c \notin C_0(S)$ we have

$$e_A(S,r,c) = 8\epsilon$$

$$\begin{aligned} &= \frac{32\epsilon}{d}\frac{d}{4} \\ &= \frac{32\epsilon}{d} w_A(S,r,c) \end{aligned}$$

since $w_A(S,r,c) = d/4$ for $c \notin C_0(S)$. Thus Equation 6.2 in fact holds for any $c \in C_0$. Together with Equation 6.1 this implies

$$\begin{aligned} \mathbf{E}_{c\in C_0}[e_A(S,r,c)] &= \mathbf{E}_{c\in C_0}\left[\frac{32\epsilon}{d} w_A(S,r,c)\right] \\ &= \frac{32\epsilon}{d}\mathbf{E}_{c\in C_0}[w_A(S,r,c)] \\ &\geq \frac{32\epsilon}{d}\frac{d}{4} \\ &= 8\epsilon. \end{aligned} \tag{6.3}$$

Thus, the experiment we are considering can be summarized as follows: fix a bad sample $S \in B_m$, and fix the random string r. This determines $h_A = A(S,r)$. Then Equation 6.3 says that if we now draw c from C_0 at random, then the expected value of $e_A(S,r,c)$ is at least 8ϵ. Since Equation 6.3 holds for *any* fixed bad sample S and random string r, it must still hold if we choose S at random from among all bad samples, and choose r at random from $\{0,1\}^k$, giving

$$\mathbf{E}_{S\in B_m, r\in\{0,1\}^k, c\in C_0}[e_A(S,r,c)] \geq 8\epsilon.$$

We now wish to change the order of this experiment. We first choose $c \in C_0$ at random, then choose a sample S at random from among all bad samples. We then choose the random string r, giving

$$\mathbf{E}_{c\in C_0, S\in B_m, r\in\{0,1\}^k}[e_A(S,r,c)] \geq 8\epsilon.$$

This is justified by the fact that the random choices of c, S and r are independent. Thus, there must be some fixed $c_0 \in C_0$ satisfying

$$\mathbf{E}_{S\in B_m, r\in\{0,1\}^k}[e_A(S,r,c_0)] \geq 8\epsilon. \tag{6.4}$$

From Equation 6.4 and the fact that

$$\mathbf{E}_{S\in B_m - B_m(c_0), r\in\{0,1\}^k}[e_A(S,r,c_0)] = 8\epsilon$$

by definition of $e_A(S,r,c_0)$ for $S \in B_m - B_m(c_0)$, it follows that

$$\mathbf{E}_{S\in B_m(c_0), r\in\{0,1\}^k}[e_A(S,r,c_0)] \geq 8\epsilon. \tag{6.5}$$

On the other hand, we have that for $S \in B_m(c_0)$

$$e_A(S, r, c_0) \leq 64\epsilon. \tag{6.6}$$

From Equations 6.5 and 6.6 we can show

$$\mathbf{Pr}_{S \in B_m(c_0), r \in \{0,1\}^k}[e_A(S, r, c_0) \geq 2\epsilon] > \frac{1}{11}. \tag{6.7}$$

To see this, let ψ be a random variable whose expectation $\mathbf{E}[\psi]$ according to some distribution is at least 8ϵ, but whose absolute value is bounded above by 64ϵ (as is the case with the random variable $e_A(S, r, c_0)$). Then if p is the probability that ψ is larger than 2ϵ, we have

$$8\epsilon \leq \mathbf{E}[\psi] < p64\epsilon + (1-p)2\epsilon.$$

Solving for p, we obtain $p > 3/31 > 1/11$ as claimed.

From Equation 6.7 and the definition of $e_A(S, r, c_0)$ for some $c_0 \in C_0(S)$, we obtain

$$\mathbf{Pr}_{S \in B_m(c_0), r \in \{0,1\}^k}[e_c^+(A(S, r)) + e_c^-(A(S, r)) \geq 2\epsilon] > \frac{1}{11}.$$

□(Lemma 6.7)

Lemma 6.8 *For $c \in C_0$, let S be a sample of size $m = S_A(\epsilon, \delta)$ drawn according to D_c^+ and D_c^-. If $m \leq d/128\epsilon$, then the probability that $S \in B_m(c)$ is at least 11δ.*

Proof: On each draw from either D_c^+ or D_c^-, the probability that a point in $\{x_1, \ldots, x_d\}$ is received is at most 32ϵ. Thus the probability that $d/2$ such points are drawn in m trails is bounded above by $GE(32\epsilon, m, d/2)$. Setting $m = d/128\epsilon$, we have by Fact CB2

$$\begin{aligned} & GE\left(32\epsilon, \frac{d}{128\epsilon}, \frac{d}{2}\right) \\ = \;& GE\left(32\epsilon, \frac{d}{128\epsilon}, 2\frac{d}{128\epsilon}32\epsilon\right) \\ \leq \;& e^{-d/12} \\ \leq \;& e^{-1/12}. \end{aligned}$$

But $e^{-1/12} < 1 - 11\delta$ for $\delta < 1/1000$. Thus the probability that $S \in B_m(c)$ must be at least 11δ, as claimed. □(Lemma 6.8)

Thus, by Lemma 6.7 we know that there exists some hard representation c_0 such that if the sample S received by A is a bad sample, then with probability at least $1/11$, either $e_c^+(h_A) \geq \epsilon$ or $e_c^-(h_A) \geq \epsilon$. But by Lemma 6.8, the probability that S is a bad sample is at least 11δ, so the probability that A fails to learn c_0 is at least $(1/11)11\delta = \delta$, completing the proof. □

No attempt has been made to optimize the constants in Theorem 6.6. A slightly modified version of the proof of Theorem 6.6 can be used to show that when $\text{VCD}(C) \geq 2$ and the sample size is $O(\text{VCD}(C)/\epsilon)$, then for any learning algorithm there is a target representation and target distributions such that the *expected* error of the hypothesis produced by the learning algorithm is at least ϵ [52]. Theorem 6.6 holds for any fixed $\epsilon \leq 1/32$ and $\delta \leq 1/1000$.

We emphasize that the lower bound of Theorem 6.6 holds regardless of the *computational* complexity of the learning algorithm — that is, even algorithms allowed infinite computational resources must use $\Omega(\text{VCD}(C)/\epsilon)$ examples. Theorem 6.6 also makes no assumptions on the hypothesis class of the learning algorithm.

We now state the previous best lower bound on the sample size, and combine it with Theorem 6.6 to obtain the general lower bound. We say that the representation class C is *trivial* if C consists of one representation, or two disjoint representations whose union (of positive examples) is the entire domain X; otherwise C is *nontrivial.*

Theorem 6.9 *(Blumer et al. [25]) Let C be a nontrivial representation class. Then any learning algorithm A for C must use sample size*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \text{VCD}(C)\right).$$

Thus our Theorem 6.6 improves separate lower bounds proportional to $\text{VCD}(C)$ and $1/\epsilon$ to a single lower bound that is proportional to their product. Using Theorems 6.6 and 6.9, we obtain our final lower bound:

Theorem 6.10 *Let C be a nontrivial representation class. Then any learning algorithm A for C must use sample size*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{\text{VCD}(C)}{\epsilon}\right).$$

6.3.1 Applications of the general lower bound

In this section, we apply Theorem 6.10 to obtain lower bounds on the number of examples needed for learning for many specific representation classes. Note that from a non-computational viewpoint, these bounds are tight for any finite representation class. This is because there is always a (possibly super-polynomial time) algorithm that finds an hypothesis consistent with a given input sample that is thus a learning algorithm with optimal sample size by Theorem 3.1. However, we will also see that these bounds prove that many of the existing polynomial-time learning algorithms have optimal or near-optimal sample size.

Before discussing lower bounds for specific representation classes, we point out the following general principle: if $\ln|C| = O(\text{VCD}(C))$ and there is a polynomial-time algorithm outputting a consistent hypothesis in C, then C is polynomially learnable (by the results of Blumer et al. [25]) with provably optimal sample complexity to within a constant factor (by Theorem 6.10).

Corollary 6.11 *Let M_n be the class of monomials over $x_1, \ldots, x_n$, and let A be a learning algorithm for M_n. Then*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{n}{\epsilon}\right).$$

Proof of this corollary follows from the Vapnik-Chervonenkis dimension of monomials and Theorem 6.10. Corollary 6.11 proves that the polynomial-time algorithm of Valiant [93] has optimal sample complexity.

Corollary 6.12 *For fixed k, let $k\text{DNF}_n$ be the class of kDNF formulae over $x_1, \ldots, x_n$, and let A be a learning algorithm for $k\text{DNF}_n$. Then*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{n^k}{\epsilon}\right).$$

By duality we obtain the same bound for kCNF. Proof of Corollary 6.12 follows from Theorem 6.10 and the Vapnik-Chervonenkis dimension. This also shows that the polynomial-time algorithms for kDNF and kCNF of Valiant [93] have optimal sample complexity.

Corollary 6.13 *Let* SF_n *be the class of symmetric functions over* $x_1, \ldots, x_n$, *and let* A *be a learning algorithm for* SF_n. *Then*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{n}{\epsilon}\right).$$

Proof follows from Theorem 6.10 and the Vapnik-Chervonenkis dimension. This also shows that the error-tolerant algorithm of Theorem 5.11 in Section 5.4 has optimal sample complexity.

Corollary 6.14 *For fixed* k, *let* k-TERM-DNF_n *be the class of* k*-term DNF formulae over* $x_1, \ldots, x_n$, *and let* A *be a learning algorithm for* k-TERM-DNF_n. *Then*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{n}{\epsilon}\right).$$

By duality the same bound holds for k-CLAUSE-CNF. Proof of Corollaries 6.14 follows from Theorem 6.10 and the Vapnik-Chervonenkis dimension. The best known polynomial-time learning algorithm for k-TERM-DNF (k-TERM-DNF, respectively) is the algorithm for kCNF (kDNF, respectively) of Valiant [93]. This algorithm uses $O(1/\epsilon \ln 1/\delta + n^k/\epsilon)$ examples. Thus, for this class there is a significant gap ($\Theta(n^{k-1})$) between the information-theoretic lower bound and the smallest sample size used by a known polynomial-time learning algorithm.

Corollary 6.15 *For fixed* k, *let* $k\mathrm{DNF}_n^s$ *be the class of* kDNF *formulae over* $x_1, \ldots, x_n$ *with at most* s *terms, and let* A *be a learning algorithm for* $k\mathrm{DNF}_n^s$. *Then*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{s\ln\frac{n}{s}}{\epsilon}\right).$$

By duality we obtain the same bound for $k\text{CNF}^s$. Proof of Corollary 6.15 follows from Theorem 6.10 and the calculation of the Vapnik-Chervonenkis dimension given by Littlestone [73]. Results by Littlestone [73] and Haussler et al. [52] can be combined to give a polynomial-time learning algorithm for $k\text{DNF}^s$ using sample size $O\left((s\ln(n/s)/\epsilon)\ln 1/\delta\right)$. Corollary 6.15 shows that this sample size exceeds the optimal by at most a factor of $O(\ln 1/\delta)$. Dual results can be stated for $k\text{CNF}$ with at most s clauses.

Corollary 6.16 *For fixed k, let $k\text{DL}_n$ be the class of k-decision lists over $x_1, \ldots, x_n$, and let A be a learning algorithm for $k\text{DL}_n$. Then*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{n^k}{\epsilon}\right).$$

Proof of Corollary 6.16 follows from Theorem 6.10 and the calculation of the Vapnik-Chervonenkis dimension given by Ehrenfeucht et al. [36]. Rivest [84] shows that $k\text{DL}$ properly includes $k\text{CNF}$ and $k\text{DNF}$, and a polynomial-time consistent algorithm for learning $k\text{DL}$ is given that uses sample size $O(1/\epsilon\ln 1/\delta + n^k/\epsilon\ln n)$. The analysis of this algorithm uses Theorem 3.1. Thus, the sample size of the algorithm of Rivest [84] is at most $O(\ln n)$ above the optimal by Corollary 6.16. Furthermore, the upper bound on $\text{VCD}(k\text{DL})$ given in Ehrenfeucht et al. [36] yields an alternative analysis of this algorithm: by applying the results of Blumer et al. [25], we see that in fact a sample of size $O(1/\epsilon\ln 1/\delta + n^k/\epsilon\ln 1/\epsilon)$ also suffices. If it is decided at run time which log factor is smaller, then we have shown that the sample complexity of the algorithm of Rivest [84] is in fact $O(1/\epsilon\ln 1/\delta + n^k/\epsilon\min(\ln 1/\epsilon, \ln n))$, a factor of $\min(\ln 1/\epsilon, \ln n)$ worse than the optimal.

Corollary 6.17 *Let LS_n denote the class of linear separators in n dimensions, and let A be a learning algorithm for LS_n. Then*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{n}{\epsilon}\right).$$

Proof of Corollary 6.17 follows from the Vapnik-Chervonenkis dimension (see e.g. Wencour and Dudley [100] or Haussler and Welzl [54]) and Theorem 6.10. A polynomial-time algorithm for learning LS by LS can be implemented using linear programming (see Karmarkar [58], Khachiyan [65], and

Blumer et al. [25] for details) in the uniform cost model of arithmetic computation. By the results of Blumer et al. [25] this algorithm requires sample size $O(1/\epsilon \ln 1/\delta + n/\epsilon \ln 1/\epsilon)$, which is within a factor of $O(\ln 1/\epsilon)$ of optimal by Corollary 6.17.

Corollary 6.18 *Let* APR_n *denote the class of axis-parallel rectangles in* n *dimensions, and let* A *be a learning algorithm for* APR_n. *Then*

$$S_A(\epsilon, \delta) = \Omega\left(\frac{1}{\epsilon} \ln \frac{1}{\delta} + \frac{n}{\epsilon}\right).$$

Proof of Corollary 6.18 follows from the Vapnik-Chervonenkis dimension (see Wencour and Dudley [100] and Blumer et al [25]) and Theorem 6.10. An efficient algorithm for learning APR by APR is given by Blumer et al. [25]; this algorithm uses sample size $O(n/\epsilon \ln n/\delta)$. By Corollary 6.18, this bound is off from optimal by a factor of at most $O(\ln n/\delta)$. Since the algorithm of Blumer et al. [25] also outputs a consistent hypothesis in APR, we can obtain a different upper bound on its sample size, namely $O(1/\epsilon \ln 1/\delta + n/\epsilon \ln 1/\epsilon)$. This bound is off from optimal by a factor of at most $O(\ln 1/\epsilon)$. As in the case of k-decision lists, we may again choose the smaller sample size at run time.

6.4 Expected sample complexity

We conclude this chapter with a brief discussion of how the sample complexity lower bounds provided here and elsewhere can be used to obtain lower bounds on the *expected* number of examples for algorithms whose sample size may depend on coin flips and the actual sequence of examples received. We first define what we mean by the expected number of examples. Let A be a (randomized) learning algorithm for a class C, let $\tilde{r}$ be an infinite sequence of bits (interpreted as the random coin tosses for A), and let $\tilde{w}$ be an infinite sequence of alternating positive and negative examples of some $c \in C$. Then we define $S_A(\epsilon, \delta, \tilde{r}, \tilde{w})$ to be the number of examples read by A (where each request for an example results in either the next positive or next negative example being read from $\tilde{w}$) on inputs ϵ, δ, $\tilde{r}$ and $\tilde{w}$. The *expected sample complexity* of A is then the maximum over all $c \in C$ and all target distributions D^+ and D^- for c of the expectation $\mathbf{E}[S_A(\epsilon, \delta, \tilde{r}, \tilde{w})]$, where the infinite bit sequence $\tilde{r}$ is drawn

uniformly at random and the infinite example sequence $\tilde{w}$ is drawn randomly according to the target distributions D^+ and D^-.

The basic format of the proofs of the lower bounds in this chapter (as well as those of Blumer et al. [25]) is to give specific distributions such that a random sample of size at most B has probability at least p of causing any learning algorithm to fail to output an ϵ-good hypothesis. To obtain a lower bound on the expected sample complexity, let A be any learning algorithm, and let q be the probability that A draws fewer than B examples when run on the same distributions that were given to prove the deterministic lower bound. Then the probability that algorithm A fails to output an ϵ-good hypothesis is bounded below by pq. Since A is a learning algorithm we must have $pq \leq \delta$, so $q \leq \delta/p$. This gives a lower bound of $(1 - \delta/p)B$ on the expected sample complexity. For example, since the value of p proved in Theorem 6.1 is $1/16$, we immediately obtain an asymptotic lower bound of $\Omega(n^k)$ for any constant k on the expected sample complexity of any negative-only learning algorithm for monomials. Similarly, since the value of p in the proof of Theorem 6.6 is $1/1000$, the expected sample complexity of any algorithm learning a representation class C is $\Omega(\text{VCD}(C)/\epsilon)$.

7

Cryptographic Limitations on Polynomial-time Learning

7.1 Introduction

Recall that in the discussion of hardness results for learning in Section 3.1, we argued that for practical purposes the polynomial learnability of a representation class must be considered unresolved until a polynomial-time learning algorithm is discovered or until a representation-independent hardness result is proved. This is because a representation-based result stating that C is not polynomially learnable by H (modulo some complexity-theoretic assumption such as $RP \neq NP$) still leaves open the possibility that C is polynomially learnable by a different hypothesis class H'. Indeed, as pointed out in Chapter 3, this possibility has been realized for natural target classes such as k-TERM-DNF and Boolean threshold functions.

Our goal in this chapter is that of proving representation-independent hardness results for a number of classes whose polynomial learnability has thus far been in question. The only previous representation-independent hardness results follow from the elegant work of Goldreich, Goldwasser and Micali [45] on constructing random functions. As mentioned in Chapter 3, their functions have many properties stronger than those mentioned here, but for our purposes we may state their result formally as follows: let $\mathrm{CKT}_n^{p(n)}$ denote the class of Boolean circuits over n inputs with at most $p(n)$ gates, and let $\mathrm{CKT}^{p(n)} = \cup_{n \geq 1} \mathrm{CKT}_n^{p(n)}$. Then it is shown by Goldreich et al. [45] that if

there exists a one-way function, then for some polynomial $p(n)$, $\mathrm{CKT}^{p(n)}$ is not polynomially learnable (by *any* polynomially evaluatable representation class). Pitt and Warmuth [79] then used this result to construct other hard-to-learn representation classes. For definitions and a discussion of one-way functions we refer the reader to Yao [102], Blum and Micali [23], Levin [68], and Goldreich et al. [45].

Note that the most powerful (polynomially evaluatable) hypothesis that can be output by a polynomial-time learning algorithm is a hypothesis that is itself a polynomial-time algorithm, or equivalently, a polynomial-size circuit. For this reason we do not expect to find polynomial-time learning algorithms for representations that do not have small circuits, since even a learning algorithm that managed to infer the *exact* target representation would not have time to write this representation down. More formally, Schapire [90] has shown that any representation class that is not polynomially evaluatable cannot be learned in polynomial time.

Thus, we may informally interpret the result of Goldreich, Goldwasser and Micali as stating that not everything with a small representation is efficiently learnable (assuming there is a one-way function). However, there is a large gap in computational power between the class of polynomial-size circuits and the classes that have been the subject of intense scrutiny within the computational learning theory community of late (e.g., DNF, decision trees, Boolean formulae, classes based on finite automata, restricted classes of circuits). In other words, at this point the boundary of what is efficiently learnable has some known limits but is still quite unclear. Can we prove hardness results similar to those of Goldreich, Goldwasser and Micali but for less powerful representation classes, thus clarifying the limits of efficient learnability?

In this chapter we prove representation-independent hardness results for learning several simple representation classes, including small Boolean formulae, acyclic deterministic finite automata, and constant-depth threshold circuits (which may be regarded as a form of simplified "neural networks"). These hardness results are based on assumptions regarding the intractability of specific number-theoretic problems of interest in cryptography, namely factoring Blum integers, inverting the RSA function, and recognizing quadratic residues. Thus, a polynomial-time learning algorithm for any of the named representation classes using *any* polynomially evaluatable hypothesis represen-

tation would immediately yield a polynomial-time algorithm for all of these cryptographic problems, which have denied efficient solution for decades, and are widely believed to be intractable.

The intuition behind the approach taken to obtain these results is contained in the following analogy. Consider a computer system with two users, Alice and Bob. Alice and Bob wish to communicate via an insecure channel, and it is assumed that Eve the eavesdropper is listening to this channel. We make no assumptions about Eve's behavior other than a polynomial bound on her computing resources. In this cryptographic setting, Alice and Bob wish to communicate *privately* in spite of Eve's nosey presence.

A classic solution to Alice and Bob's problem is the *one-time pad*. Here Alice and Bob would physically meet in a secure room (away from Eve) and compile a large common table of random bits. Then after separating, Bob, to send a bit b to Alice, chooses the next random bit c from the common list and sends the bit $b \oplus c$ to Alice. It is easily verified that if the bit c is uniformly distributed then the encoded bit $b \oplus c$ is also uniformly distributed, regardless of the value of the cleartext message bit b. Thus Eve, regardless of computation time, is provably unable to gain any information about the cleartext messages from listening to the channel between Alice and Bob. Alice, however, also knows the random bit c, and so may decode by computing $(b \oplus c) \oplus c = b$.

There are some obvious practical problems with the one-time pad. Foremost among these is the need for Alice and Bob to meet in person and compile the table of random bits; in a network of thousands of computers, having every pair of users meet clearly defeats the point of using computers in the first place. In response to complaints such as these and also more subtle security concerns, the field of *public-key cryptography* was initiated by Diffie and Hellman [32].

Public-key cryptography solves the problem of Alice and Bob via the use of *trapdoor functions*. Informally, a trapdoor function is one that can be computed in polynomial time (i.e., it is easy to compute $f(x)$ on input x) but cannot be inverted in polynomial time (i.e., it is hard to compute x on input $f(x)$) — unless one is the "creator" of the function, in which case one possesses a piece of "trapdoor" information that makes inversion possible in polynomial time. Now rather than meeting with Bob in person, Alice "creates" a trapdoor function f and *publishes* a program for computing f (which reveals no information about f^{-1}) in a directory that is available to everyone — Bob and

Eve included. To send the message x to Alice, Bob simply computes $f(x)$ and sends it to Alice. Eve, seeing only $f(x)$ on the channel and not possessing the trapdoor, is unable to recover the message x in polynomial time. Alice, being the creator of f and thus having the trapdoor, can efficiently invert Bob's ciphertext and recover x.

Our approach is based on viewing Eve as a learning algorithm. Note that since a program for f is available to Eve, she may create as many pairs of the form $(f(x), x)$ that she likes simply by choosing x and then computing $f(x)$. If we set $y = f(x)$, we see that such pairs have the form $(y, f^{-1}(y))$, and can thus be regarded as "examples" of the inverse function f^{-1}. Thus, from the learning perspective, public-key cryptography assumes the existence of functions that are not learnable from examples, since if Eve could learn f^{-1} efficiently from examples of its input-output behavior, she could then decode messages sent from Bob to Alice! Furthermore, note that the inverse function f^{-1} is "simple" in the sense that it does have a small circuit (determined by the trapdoor, which Alice has access to and uses for decoding); thus from an information-theoretic standpoint the learning problem is "fair", as opposed to the one-time pad, where there is no small circuit underlying the communication between Alice and Bob, just a large random bit table.

Thus we see that recent developments in the theory of cryptography provide us with simple functions that are difficult to learn. Our approach in this chapter is based on refining the functions provided by cryptography in an attempt to find the *simplest* functions that are difficult to learn.

An outline of the chapter is as follows: we give the necessary background and references from cryptography in Section 7.2. We develop and motivate our techniques for proving representation-independent hardness results in Section 7.3 and apply these to specific classes of interest in Section 7.4. In Section 7.5 we generalize our methods to give a general technique for proving hardness results based on the weaker assumption that there exists a trapdoor function [102]. The classes for which one can prove hardness results will be determined by properties of the trapdoor function chosen.

In Section 7.6 we embark on a brief digression and apply our learning results to prove hardness results for approximating combinatorial optimization problems. In particular, we define a problem that generalizes the graph coloring problem and prove that approximating the optimal solution by an algorithm

using as many as $(opt)^{\alpha}|I|^{\beta}$ colors (for any $\alpha \geq 1$ and $\beta < 1$), where *opt* is the value of the optimal solution for instance I and $|I|$ is the instance size, is as hard as the number-theoretic problems mentioned above. This illustrates that cryptographic assumptions may suffice to give negative results for combinatorial optimization in some cases where no *NP*-hardness results are known, and seem difficult to obtain.

7.2 Background from cryptography

Some basic number theory. For an introduction to number theory that is relevant to cryptography, we refer the reader to the work of Angluin [6] and Kranakis [66]. For N a natural number, Z_N will denote the ring of integers modulo N, and Z_N^* will denote the multiplicative group modulo N. Thus $Z_N = \{x : 0 \leq x \leq N-1\}$ and $Z_N^* = \{x : 1 \leq x \leq N-1 \text{ and } gcd(x,N) = 1\}$, where $gcd(x,N)$ denotes the greatest common divisor of x and N. The *Euler totient function* φ is defined by $\varphi(N) = |Z_N^*|$. For $x \in Z_N^*$, we say that x is a *quadratic residue* modulo N if there is an $a \in Z_N^*$ such that $x = a^2 \ mod \ N$. We denote by QR_N the set of all quadratic residues in Z_N^*. For a prime p and $x \in Z_p^*$, we define the *Legendre symbol* of x with respect to p by $L(x,p) = 1$ if x is a quadratic residue modulo p, and $L(x,p) = -1$ otherwise. For $N = p \cdot q$, where p and q are prime, we define the *Jacobi symbol* of $x \in Z_N^*$ with respect to N by $J(x,N) = L(x,p) \cdot L(x,q)$. Since x is a quadratic residue modulo N if and only if it is a quadratic residue modulo p and modulo q, it follows that $J(x,N) = -1$ implies that x is not a quadratic residue modulo N. However, $J(x,N) = 1$ does not necessarily imply that x is a quadratic residue mod N. For any integer N, we define the sets $Z_N^*(+1) = \{x \in Z_N^* : J(x,N) = 1\}$ and $QR_N(+1) = \{x \in Z_N^* : J(x,N) = 1 \ and \ x \in QR_N\}$. A *Blum integer* is an integer of the form $p \cdot q$, where p and q are primes both congruent to 3 modulo 4.

We will make use of the following facts from number theory.

Fact NT1. On inputs x and N, $gcd(x,N)$ can be computed in polynomial time.

Fact NT2. For p a prime and $x \in Z_p^*$, $L(x,p) = x^{(p-1)/2} \ mod \ p$.

Fact NT3. On inputs x and N, $J(x, N)$ can be computed in polynomial time.

Fact NT4. For $N = p \cdot q$ where p and q are prime, $|Z_N^*(+1)| = |Z_N^*|/2$ and $|QR_N(+1)| = |Z_N^*|/4$.

Fact NT5. For any $x \in Z_N^*$, $x^{\varphi(N)} = 1 \ mod \ N$.

The RSA encryption function. Let p and q be primes of length l, and let $N = p \cdot q$. Let e be an *encrypting exponent* such that $gcd(e, \varphi(N)) = 1$ and d a *decrypting exponent* such that $d \cdot e = 1 \ mod \ \varphi(N)$. The existence of such a d is guaranteed by the existence of multiplicative inverses modulo $\varphi(N)$. The *RSA encryption function* [88] is then defined by

$$RSA(x, N, e) = x^e \ mod \ N.$$

Note that decryption can be accomplished by exponentiation mod N:

$$(x^e)^d = x^{e \cdot d} \ mod \ N = x^{1+i \cdot \varphi(N)} \ mod \ N = x \ mod \ N$$

for some natural number i by Fact NT5 because $e \cdot d = 1 \ mod \ \varphi(N)$.

Thus, following the informal intuition of Section 7.1, we think of Alice as generating the product $N = p \cdot q$; since she also knows p and q, she can generate both e (which she publishes along with N, thus yielding an encryption program) and d (the "trapdoor", which she keeps private).

There is currently no known polynomial-time algorithm for *inverting* the RSA encryption function — that is, the problem of computing x on inputs $RSA(x, N, e), N$ and e. Furthermore, the following result from Alexi et al. [5] indicates that determining the least significant bit of x is as hard as inverting RSA (which amounts to determining *all* the bits of x).

Theorem 7.1 *(Alexi et al. [5]) Let x, N and e be as above. Then with respect to probabilistic polynomial-time reducibility, the following problems are equivalent:*

(1) On input $RSA(x, N, e), N$ and e, output x.

(2) On input $RSA(x, N, e), N$ and e, output $LSB(x)$ with probability exceeding $1/2 + 1/p(l)$, where p is any fixed polynomial, $l = \log N$ is the length of N, and $LSB(x)$ denotes the least significant bit of x. The probability is taken over x chosen uniformly from Z_N and any coin tosses of A.

The Rabin and modified Rabin encryption functions. The *Rabin encryption function* [82] is specified by two primes p and q of length l. For $N = p \cdot q$ and $x \in Z_N^*$, we define

$$R(x, N) = x^2 \ mod \ N.$$

In this scheme the trapdoor is the factorization of N, which allows Alice to compute square roots modulo N, and thus to decrypt. Known results regarding the security of the Rabin function include the following:

Theorem 7.2 *(Rabin [82]) Let x and N be as above. Then with respect to probabilistic polynomial-time reducibility, the following problems are equivalent:*

(1) On input N, output a nontrivial factor of N.

(2) On input N and $R(x, N)$, output x.

Furthermore, this reduction still holds when N is restricted to be a Blum integer in both problems. The *modified Rabin encryption function* [5] is specified by two primes p and q of length l, both congruent to 3 modulo 4. Let $N = p \cdot q$ (thus N is a Blum integer). We define a subset M_N of Z_N^* by

$$M_N = \{x : 0 \leq x \leq \frac{N}{2} \text{ and } x \in Z_N^*(+1)\}.$$

For $x \in M_N$, the modified Rabin encryption function is then

$$MR(x, N) = x^2 \ mod \ N \text{ if } x^2 \ mod \ N \in M_N$$

$$MR(x, N) = (N - x^2) \ mod \ N \text{ otherwise.}$$

This defines a 1-1 map from M_N onto M_N.

Theorem 7.3 *(Alexi et al. [5]) Let x and N be as above. Then with respect to probabilistic polynomial-time reducibility, the following problems are equivalent:*

(1) On input $MR(x, N)$ and N, output x.

(2) On input $MR(x, N)$ and N, output $LSB(x)$ with probability exceeding $1/2 + 1/p(l)$, where p is any fixed polynomial and $l = \log N$ is the length of N. The probability is taken over x chosen uniformly from M_N and any coin tosses of A.

For Blum integers, $R(x, N)$ is a 1-1 mapping of QR_N. Hence if $MR(x, N)$ is invertible then we can invert $R(x, N)$ by attempting to invert MR for both the values $R(x, N)$ and $N - R(x, N)$, and succeeding for just the right one of these. Hence Theorems 7.2 and 7.3 together imply that Problem (2) in Theorem 7.3 is equivalent to factoring Blum integers (with respect to probabilistic polynomial-time reducibility), a problem for which no polynomial-time algorithm is known.

The Quadratic Residue Assumption. Let $N = p \cdot q$, where p and q are primes of length l. For each $x \in Z_N^*(+1)$, define $QR(x, N) = 1$ if x is a quadratic residue mod N and $QR(x, N) = 0$ otherwise. Then the *Quadratic Residue Assumption* states that if A is any probabilistic polynomial-time algorithm that takes N and x as input, then for infinitely many N we have

$$\mathbf{Pr}[A(N, x) = QR(x, N)] \quad < \quad \frac{1}{2} + \frac{1}{p(l)}$$

where p is any fixed polynomial. The probability is taken over x chosen uniformly from the set $Z_N^*(+1)$ and any coin tosses of A. As in the Rabin scheme, knowledge of the factors of N allows Alice to compute square roots modulo N and thus to determine if an element is a quadratic residue.

7.3 Hard learning problems based on cryptographic functions

In this section we construct hard learning problems based on the number-theoretic encryption functions described above. For each such function, we first define a representation class based on the function. For each possible target representation in this class, we then describe the *relevant examples* for this

representation. These are the only examples with non-zero probability in the hard target distributions we define. We then proceed to prove the difficulty of even *weakly* learning the representation class under the chosen distributions, based on some standard cryptographic assumption on the security of the underlying encryption function. Finally, we show the ease of *evaluating* the representation class: more precisely, we show that each representation in the class can be computed by an NC^1 circuit (a polynomial-size, log-depth circuit of standard fan-in 2 Boolean gates). In Section 7.4 we apply these results to prove that weakly learning Boolean formulae, finite automata, constant-depth threshold circuits and a number of other representation classes is hard under cryptographic assumptions.

We adopt the following notation: if $a_1, \ldots, a_m$ are natural numbers, we denote by $binary(a_1, \ldots, a_m)$ the binary representation of the sequence $a_1, \ldots, a_m$ in some fixed encoding scheme. The relevant examples we construct will be of the form

$$< binary(a_1, \ldots, a_m), b >$$

where b is a bit indicating whether the example is positive or negative. We denote by $powers(z, N)$ the sequence of natural numbers

$$z \ mod \ N, z^2 \ mod \ N, z^4 \ mod \ N, \ldots, z^{2^{\lceil \log N \rceil}} \ mod \ N$$

which are the first $\lceil \log N \rceil + 1$ successive square powers of z modulo N.

In the following subsections, we will define representation classes C_n based on the number-theoretic function families described above. Representations in C_n will be over the domain $\{0,1\}^n$; relevant examples with length less than n will implicitly be assumed to be padded to length n. Since only the relevant examples will have non-zero probability, we assume that on all non-relevant examples are negative examples of the target representation.

7.3.1 A learning problem based on RSA

The representation class C_n: Let l be the largest natural number satisfying $4l^2 + 6l \leq n$. Each representation in C_n is defined by a triple (p, q, e) and this representation will be denoted $r_{(p,q,e)}$. Here p and q are l-bit primes and $e \in Z^*_{\varphi(N)}$, where $N = p \cdot q$ (thus, $gcd(e, \varphi(N)) = 1$).

Relevant examples for $r_{(p,q,e)} \in C_n$: A relevant example of $r_{(p,q,e)} \in C_n$ is of the form

$$< binary(powers(RSA(x, N, e), N), N, e), LSB(x) >$$

where $x \in Z_N$. Note that since the length of N is $2l$, the length of such an example in bits is $(2l + 1)2l + 2l + 2l = 4l^2 + 6l \leq n$. The target distribution D^+ for $r_{(p,q,e)}$ is uniform over the relevant positive examples of $r_{(p,q,e)}$ (i.e., those for which $LSB(x) = 1$) and the target distribution D^- is uniform over the relevant negative examples (i.e., those for which $LSB(x) = 0$).

Difficulty of weakly learning $C = \cup_{n \geq 1} C_n$: Suppose that A is a polynomial-time weak learning algorithm for C. We now describe how we can use algorithm A to invert the RSA encryption function. Let N be the product of two unknown l-bit primes p and q, and let $e \in Z^*_{\varphi(N)}$. Then given only N and e, we run algorithm A. Each time A requests a positive example of $r_{(p,q,e)}$, we uniformly choose an $x \in Z_N$ such that $LSB(x) = 1$ and give the example

$$< binary(powers(RSA(x, N, e), N), N, e), 1 >$$

to A. Note that we can generate such an example in polynomial time on input N and e. This simulation generates the target distribution D^+. Each time that A requests a negative example of $r_{(p,q,e)}$, we uniformly choose an $x \in Z_N$ such that $LSB(x) = 0$ and give the example

$$< binary(powers(RSA(x, N, e), N), N, e), 0 >$$

to A. Again, we can generate such an example in polynomial time, and this simulation generates the target distribution D^-. Let h_A be the hypothesis output by algorithm A following this simulation. Then given $r = RSA(x, N, e)$ for some unknown x chosen uniformly from Z_N, $h_A(binary(powers(r, N), N, e)) = LSB(x)$ with probability at least $1/2 + 1/p(l)$ for some polynomial p by the definition of weak learning. Thus we have a polynomial advantage for inverting the least significant bit of RSA. This allows us to invert RSA by the results of Alexi et al. [5] given as Theorem 7.1.

Ease of evaluating $r_{(p,q,e)} \in C_n$: For each $r_{(p,q,e)} \in C_n$, we show that $r_{(p,q,e)}$ has an equivalent NC^1 circuit. More precisely, we give a circuit that has

depth $O(\log n)$ and size polynomial in n, and outputs the value of $r_{(p,q,e)}$ on inputs of the form

$$binary(powers(r, N), N, e)$$

where $N = p \cdot q$ and $r = RSA(x, N, e)$ for some $x \in Z_N$. Thus, the representation class $C = \cup_{n \geq 1} C_n$ is contained in (nonuniform) NC^1.

Since $e \in Z^*_{\varphi(N)}$, there is a $d \in Z^*_{\varphi(N)}$ such that $e \cdot d = 1 \ mod \ \varphi(N)$ (d is just the decrypting exponent for e). Thus, $r^d \ mod \ N = x^{e \cdot d} \ mod \ N = x \ mod \ N$. Hence the circuit for $r_{(p,q,e)}$ simply multiplies together the appropriate powers of r (which are always explicitly provided in the input) to compute $r^d \ mod \ N$, and outputs the least significant bit of the resulting product. This is an NC^1 step by the iterated product circuits of Beame, Cook and Hoover [15].

7.3.2 A learning problem based on quadratic residues

The representation class C_n: Let l be the largest natural number satisfying $4l^2 + 4l \leq n$. Each representation in C_n is defined by a pair of l-bit primes (p, q) and this representation will be denoted $r_{(p,q)}$.

Relevant examples for $r_{(p,q)} \in C_n$: For a representation $r_{(p,q)} \in C_n$, let $N = p \cdot q$. We consider only points $x \in Z^*_N(+1)$. A relevant example of $r_{(p,q)}$ is then of the following form:

$$< binary(powers(x, N), N), QR(x, N) > .$$

Note that the length of such an example in bits is $(2l + 1)2l + 2l = 4l^2 + 4l \leq n$. The target distribution D^+ for $r_{(p,q)}$ is uniform over the relevant positive examples of $r_{(p,q)}$ (i.e., those for which $QR(x, N) = 1$) and the target distribution D^- is uniform over the relevant negative examples (i.e., those for which $QR(x, N) = 0$).

Difficulty of weakly learning $C = \cup_{n \geq 1} C_n$: Suppose that A is a polynomial-time weak learning algorithm for C. We now describe how we can use algorithm A to recognize quadratic residues. Let N be the product of

two unknown l-bit primes p and q. Given only N as input, we run algorithm A. Every time A requests a positive example of $r_{(p,q)}$, we uniformly choose $y \in Z_N^*$ and give the example

$$< binary(powers(y^2 \ mod \ N, N), N), 1 >$$

to A. Note that such an example can be generated in polynomial time on input N. This simulation generates the target distribution D^+.

In order to generate the negative examples for our simulation of A, we uniformly choose $u \in Z_N^*$ until $J(u, N) = 1$. By Fact NT4, this can be done with high probability in polynomial time. The probability is $1/2$ that such a u is a non-residue modulo N. Assuming we have obtained a non-residue u, every time A requests a negative example of $r_{(p,q)}$, we uniformly choose $y \in Z_N^*$ and give to A the example

$$< binary(powers(uy^2 \ mod \ N, N), N), 0 >$$

which can be generated in polynomial time. Note that if u actually is a non-residue then this simulation generates the target distribution D^-, and this run of A will with high probability produce an hypothesis h_A with accuracy at least $1/2 + 1/p(l)$ with respect to D^+ and D^-, for some polynomial p (call such a run a *good* run). On the other hand, if u is actually a residue then A has been trained improperly (that is, A has been given positive examples when it requested negative examples), and no performance guarantees can be assumed. The probability of a good run of A is at least $1/2(1 - \delta)$.

We thus simulate A as described above many times, testing each hypothesis to determine if the run was a good run. To test if a good run has occurred, we first determine if h_A has accuracy at least $1/2 + 1/2p(l)$ with respect to D^+. This can be determined with high probability by generating D^+ as above and estimating the accuracy of h_A using Fact CB1 and Fact CB2. Assuming h_A passes this test, we now would like to test h_A against the simulated distribution D^-; however, we do not have direct access to D^- since this requires a non-residue *mod* N. Thus we instead estimate the probability that h_A classifies an example as positive when this example is drawn from the uniform distribution over *all* relevant examples (both positive and negative). This can be done by simply choosing $x \in Z_N^*$ uniformly and computing $h_A(binary(powers(x, N), N))$. The

probability that h_A classifies such examples as positive is near $1/2$ if and only if h_A has nearly equal accuracy on D^+ and D^-. Thus by estimating the accuracy of h_A on D^+, we can estimate the accuracy of h_A on D^- as well, without direct access to a simulation of D^-.

We continue to run A and test until a good run of A is obtained with high probability. Then given x chosen randomly from Z_N^*,

$$h_A(binary(powers(x, N), N)) = QR(x, N)$$

with probability at least $1/2+1/p(l)$, contradicting the Quadratic Residue Assumption.

Ease of evaluating $r_{(p,q)} \in C_n$: For each $r_{(p,q)} \in C_n$, we give an NC^1 circuit for evaluating the concept represented by $r_{(p,q)}$ on an input of the form

$$binary(powers(x, N), N)$$

where $N = p \cdot q$ and $x \in Z_N^*$. This circuit has four phases.

Phase I. Compute the powers

$$x \bmod p, x^2 \bmod p, x^4 \bmod p, \ldots, x^{2^{2l}} \bmod p$$

and the powers

$$x \bmod q, x^2 \bmod q, x^4 \bmod q, \ldots, x^{2^{2l}} \bmod q.$$

Note that the length of N is $2l$. Since for any $a \in Z_N^*$ we have that $a \bmod p = (a \bmod N) \bmod p$, these powers can be computed from the input $binary(powers(x, N), N)$ by parallel $\bmod p$ and $\bmod q$ circuits. Each such circuit involves only a division step followed by a multiplication and a subtraction. The results of Beame et al. [15] imply that these steps can be carried out by an NC^1 circuit.

Phase II. Compute $x^{(p-1)/2} \bmod p$ and $x^{(q-1)/2} \bmod q$. These can be computed by multiplying the appropriate powers mod p and mod q computed in Phase I. Since the iterated product of l numbers each of length l bits can be computed in NC^1 by the results of Beame et al. [15], this is also an NC^1 step.

Phase III. Determine if $x^{(p-1)/2} = 1 \ mod \ p$ or $x^{(p-1)/2} = -1 \ mod \ p$, and if $x^{(q-1)/2} = 1 \ mod \ q$ or $x^{(q-1)/2} = -1 \ mod \ q$. That these are the only cases follows from Fact NT2; furthermore, this computation determines whether x is a residue mod p and mod q. Given the outputs of Phase II, this is clearly an NC^1 step.

Phase IV. If the results of Phase III were $x^{(p-1)/2} = 1 \ mod \ p$ and $x^{(q-1)/2} = 1 \ mod \ q$, then output 1, otherwise output 0. This is again an NC^1 step.

7.3.3 A learning problem based on factoring Blum integers

The representation class C_n: Let l be the largest natural number satisfying $4l^2 + 4l \leq n$. Each representation in C_n is defined by a pair of l-bit primes (p, q), both congruent to 3 modulo 4, and this representation will be denoted $r_{(p,q)}$. Thus the product $N = p \cdot q$ is a Blum integer.

Relevant examples for $r_{(p,q)} \in C_n$: We consider points $x \in M_N$. A relevant example of $r_{(p,q)} \in C_n$ is then of the form

$$< binary(powers(MR(x, N), N), N), LSB(x) > .$$

The length of this example in bits is $(2l + 1)2l + 2l = 4l^2 + 4l \leq n$. The target distribution D^+ for $r_{(p,q)}$ is uniform over the relevant positive examples (i.e., those for which $LSB(x) = 1$) and the target distribution D^- is uniform over the relevant negative examples (i.e., those for which $LSB(x) = 0$).

Difficulty of weakly learning $C = \cup_{n \geq 1} C_n$: Suppose that A is a polynomial-time weak learning algorithm for C. We now describe how to use A to factor Blum integers. Let N be a Blum integer. Given only N as input, we run algorithm A. Every time A requests a positive example, we choose $x \in M_N$ uniformly such that $LSB(x) = 1$, and give the example

$$< binary(powers(MR(x, N), N), N), 1 >$$

to A. Such an example can be generated in polynomial time on input N. This simulation generates the distribution D^+. Every time A requests a

negative example, we choose $x \in M_n$ uniformly such that $LSB(x) = 0$, and give the example

$$< binary(powers(MR(x, N), N), N), 0 >$$

to A. Again, this example can be generated in polynomial time. This simulation generates the distribution D^-. When algorithm A has halted, $h_A(binary(powers(r, N), N)) = LSB(x)$ with probability $1/2 + 1/p(l)$ for $r = MR(x, N)$ and x chosen uniformly from M_N. This implies that we can factor Blum integers by the results of Rabin [82] and Alexi et al. [5] given in Theorems 7.2 and 7.3.

Ease of evaluating $r_{(p,q)} \in C_n$: For each $r_{(p,q)} \in C_n$, we give an NC^1 circuit for evaluating the concept represented by $r_{(p,q)}$ on an input of the form

$$binary(powers(r, N), N)$$

where $N = p \cdot q$ and $r = MR(x, N)$ for some $x \in M_N$. This is accomplished by giving an NC^1 implementation of the first three steps of the root-finding algorithm of Adleman, Manders and Miller [2] as it is described by Angluin [6]. Note that if we let $a = x^2 \ mod \ N$, then either $r = a$ or $r = (N - a) \ mod \ N$ according to the definition of the modified Rabin function. The circuit has four phases.

Phase I. Determine if the input r is a quadratic residue mod N. This can be done using the given powers of r and $r_{(p,q)}$ using the NC^1 circuit described in quadratic residue-based scheme of Section 7.3.2. Note that since p and q are both congruent to 3 *mod* 4, $(N-a) \ mod \ N$ is never a quadratic residue mod N (see Angluin [6]). If it is decided that $r = (N - a) \ mod \ N$, generate the intermediate output $a \ mod \ N$. This can clearly be done in NC^1. Also, notice that for any z, $z^{2i} = (N - z)^{2i} \ mod \ N$ for $i \geq 1$. Hence these powers of r are identical in the two cases. Finally, recall that the NC^1 circuit for quadratic residues produced the powers of r mod p and the powers of r mod q as intermediate outputs, so we may assume that the powers

$$a, a^2 \ mod \ p, a^4 \ mod \ p, \ldots, a^{2^{2l}} \ mod \ p$$

and

$$a, a^2 \ mod \ q, a^4 \ mod \ q, \ldots, a^{2^{2l}} \ mod \ q$$

are also available.

Phase II. Let l_p (respectively, l_q) be the largest positive integer such that $2^{l_p}|(p-1)$ (respectively, $2^{l_q}|(q-1)$). Let $Q_p = (p-1)/2^{l_p}$ (respectively, $Q_q = (q-1)/2^{l_q}$). Using the appropriate powers of x^2 mod p and mod q, compute $u = a^{(Q_p+1)/2} \ mod \ p$ and $v = a^{(Q_q+1)/2} \ mod \ q$ with NC^1 iterated product circuits. Since p and q are both congruent to 3 *mod* 4, u and $p-u$ are square roots of a mod q, and v and $q-v$ are square roots of a mod q by the results of Adleman et al. [2] (see also Angluin [6]).

Phase III. Using Chinese remaindering, combine $u, p-u, v$ and $q-v$ to compute the four square roots of a mod N (see e.g. Kranakis [66]). Given p and q, this requires only a constant number of multiplication and addition steps, and so is computed in NC^1.

Phase IV. Find the root from Phase III that is in M_N, and output its least significant bit.

7.4 Learning small Boolean formulae, finite automata and threshold circuits is hard

The results of Section 7.3 show that for some fixed polynomial $q(n)$, learning NC^1 circuits of size at most $q(n)$ is computationally as difficult as the problems of inverting RSA, recognizing quadratic residues, and factoring Blum integers. However, there is a polynomial $p(n)$ such that any NC^1 circuit of size at most $q(n)$ can be represented by a Boolean formulae of size at most $p(n)$. Thus we have proved the following:

Theorem 7.4 *Let* $\mathrm{BF}_n^{p(n)}$ *denote the class of Boolean formulae over* n *variables of size at most* $p(n)$*, and let* $\mathrm{BF}^{p(n)} = \cup_{n \geq 1} \mathrm{BF}_n^{p(n)}$*. Then for some polynomial* $p(n)$*, the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to weakly learning* $\mathrm{BF}^{p(n)}$*.*

In fact, we can apply the substitution arguments of Section 4.3 to show that Theorem 7.4 holds even for the class of monotone Boolean formulae in which each variable appears at most once.

Pitt and Warmuth [79] show that if the class ADFA is polynomially weakly learnable, then the class BF is polynomially weakly learnable. Combining this with Theorem 7.4, we have:

Theorem 7.5 *Let* $\mathrm{ADFA}_n^{p(n)}$ *denote the class of deterministic finite automata of size at most* $p(n)$ *that only accept strings of length* n*, and let* $\mathrm{ADFA}^{p(n)} = \cup_{n\geq 1}\mathrm{ADFA}_n^{p(n)}$*. Then for some polynomial* $p(n)$*, the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to weakly learning* $\mathrm{ADFA}^{p(n)}$*.*

Using results of Chandra, Stockmeyer and Vishkin [27], Beame et al. [15] and Reif [83], it can be shown that the representations described in Section 7.3 can each be computed by a polynomial-size, constant-depth threshold circuit. Thus we have:

Theorem 7.6 *For some fixed constant natural number* d*, let* $d\mathrm{TC}_n^{p(n)}$ *denote the class of threshold circuits over* n *variables with depth at most* d *and size at most* $p(n)$*, and let* $d\mathrm{TC}^{p(n)} = \cup_{n\geq 1} d\mathrm{TC}_n^{p(n)}$*. Then for some polynomial* $p(n)$*, the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to weakly learning* $d\mathrm{TC}^{p(n)}$*.*

It is important to reiterate that these hardness results hold regardless of the hypothesis representation class of the learning algorithm; that is, Boolean formulae, DFA's and constant-depth threshold circuits are not weakly learnable by *any* polynomially evaluatable representation class (under standard cryptographic assumptions). We note that no NP-hardness results are known for these classes even if we restrict the hypothesis class to be the same as the target class and insist on strong learnability rather than weak learnability. It is also possible to give reductions showing that many other interesting classes (e.g., CFG's and NFA's) are not weakly learnable, under the same cryptographic assumptions. In general, any representation class whose computational power subsumes that of NC^1 is not weakly learnable; however, more subtle reductions are also possible. In particular, our results resolve a problem posed by Pitt and Warmuth [79] by showing that under cryptographic assumptions,

the class of all languages accepted by logspace Turing machines is not weakly learnable.

Pitt and Warmuth [79] introduce a general notion of reduction between learning problems, and a number of learning problems are shown to have equivalent computational difficulty (with respect to probabilistic polynomial-time reducibility). Learning problems are then classified according to the complexity of their *evaluation problem*, the problem of evaluating a representation on an input example. In Pitt and Warmuth [79] the evaluation problem is treated as a uniform problem (i.e., one algorithm for evaluating all representations in the class); by treating the evaluation problem nonuniformly (e.g., a separate circuit for each representation) we were able to show that NC^1 contains a number of presumably hard-to-learn classes of Boolean functions. By giving reductions from NC^1 to other classes of representations, we thus clarify the boundary of what is efficiently learnable.

7.5 A generalized construction based on any trapdoor function

Let us now give a brief summary of the techniques that were used in Sections 7.3 and 7.4 to obtain hardness results for learning based on cryptographic assumptions. In each construction (RSA, quadratic residue and factoring Blum integers), we began with a candidate *trapdoor function* family, informally a family of functions each of whose members f is easy to compute (that is, given x, it is easy to compute $f(x)$), hard to invert (that is, given only $f(x)$, it is difficult to compute x), but easy to invert given a secret "key" to the function [102] (the *trapdoor*). We then constructed a learning problem in which the complexity of inverting the function *given* the trapdoor key corresponds to the complexity of the representations being learned, and learning from random examples corresponds to inverting the function *without* the trapdoor key. Thus, the learning algorithm is essentially required to learn the inverse of a trapdoor function, and the small representation for this inverse is simply the secret trapdoor information.

To prove hardness results for the simplest possible representation classes, we then eased the computation of the inverse given the trapdoor key by pro-

viding the powers of the original input in each example. This additional information provably does not compromise the security of the original function. A key property of trapdoor functions exploited by our constructions is the ability to generate random examples of the target representation without the trapdoor key; this corresponds to the ability to generate encrypted messages given only the public key in a public-key cryptosystem.

By assuming that specific functions such as RSA are trapdoor functions, we were able to find modified trapdoor functions whose inverse computation given the trapdoor could be performed by very simple circuits. This allowed us to prove hardness results for specific representation classes that are of interest in computational learning theory. Such specific intractability assumptions appear necessary since the weaker and more general assumption that there exists a trapdoor family that can be computed (in the forward direction) in polynomial time does not allow us to say anything about the hard-to-learn representation class other than it having polynomial-size circuits.

However, the summary above suggests a general method for proving hardness results for learning: to show that a representation class C is not learnable, find a trapdoor function whose inverse can be computed by C given the trapdoor key. In this section we formalize these ideas and prove a theorem demonstrating that this is indeed a viable approach.

We use the following definition for a family of trapdoor functions, which can be derived from Yao [102]: let $P = \{P_n\}$ be a family of probability distributions, where for $n \geq 1$ the distribution P_n is over pairs $(k, k') \in \{0,1\}^n \times \{0,1\}^n$. We think of k as the n-bit *public key* and k' as the associated n-bit *private key*. Let $Q = \{Q_k\}$ be a family of probability distributions parameterized by the public key k, where if $|k| = n$ then Q_k is a distribution over $\{0,1\}^n$. We think of Q as a distribution family over the *message space*. The function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ maps an n-bit public key k and an n-bit *cleartext message* x to the *ciphertext* $f(k,x)$. We call the triple (P, Q, f) an *α-strong trapdoor scheme* if it has the following properties:

(i) There is probabilistic polynomial-time algorithm G (the *key generator*) that on input 1^n outputs a pair (k, k') according to the distribution P_n. Thus, pairs of public and private keys are easily generated.

(ii) There is a probabilistic polynomial-time algorithm M (the *message gen-*

erator) that on input k outputs x according to the distribution Q_k. Thus, messages are easily generated given the public key k.

(*iii*) There is a polynomial-time algorithm E that on input k and x outputs $f(k,x)$. Thus, encryption is easy.

(*iv*) Let A be any probabilistic polynomial-time algorithm. Perform the following experiment: draw a pair (k,k') according to P_n, and draw x according to Q_k. Give the inputs k and $f(k,x)$ to A. Then the probability that $A(k,f(k,x)) \neq x$ is at least α. Thus, decryption from only the public key and the ciphertext is hard.

(*v*) There is a polynomial-time algorithm D that on input k,k' and $f(k,x)$ outputs x. Thus, decryption given the private key (or *trapdoor*) is easy.

As an example, consider the RSA cryptosystem [88]. Here the distribution P_n is uniform over all (k,k') where $k' = (p,q)$ for n-bit primes p and q and $k = (p \cdot q, e)$ with $e \in Z^*_{\varphi(p \cdot q)}$. The distribution Q_k is uniform over $Z_{p \cdot q}$, and $f(k,x) = f((p \cdot q, e), x) = x^e \ mod \ p \cdot q$.

We now formalize the notion of the inverse of a trapdoor function being computed in a representation class. Let $C = \cup_{n \geq 1} C_n$ be a parameterized Boolean representation class. We say that a trapdoor scheme (P,Q,f) is *invertible in C given the trapdoor* if for any $n \geq 1$, for any pair of keys $(k,k') \in \{0,1\}^n \times \{0,1\}^n$, and for any $1 \leq i \leq n$, there is a representation $c^i_{(k,k')} \in C_n$ that on input $f(k,x)$ (for any $x \in \{0,1\}^n$) outputs the ith bit of x.

Theorem 7.7 *Let p be any polynomial, and let $\alpha(n) \geq 1/p(n)$. Let (P,Q,f) be an $\alpha(n)$-strong trapdoor scheme, and let C be a parameterized Boolean representation class. Then if (P,Q,f) is invertible in C given the trapdoor, C is not polynomially learnable.*

Proof: Let A be any polynomial-time learning algorithm for C. We use algorithm A as a subroutine in a polynomial-time algorithm A' that with high probability outputs x on input k and $f(k,x)$, thus contradicting condition (*iv*) in the definition of a trapdoor scheme.

Let (k,k') be n-bit public and private keys generated by the distribution P_n. Let x be an n-bit message generated according to the distribution Q_k.

Then on input k and $f(k,x)$, algorithm A' behaves as follows: for $1 \leq i \leq n$, algorithm A' simulates algorithm A, choosing accuracy parameter $\epsilon = \alpha(n)/n$. For the ith run of A, each time A requests a positive example, A' generates random values x' from the distribution Q_k (this can be done in polynomial time by condition (ii) in the definition of trapdoor scheme) and computes $f(k,x')$ (this can be done in polynomial time by condition (iii) in the definition of trapdoor scheme). If the ith bit of $f(k,x')$ is 1, then A' gives x' as a positive example to A; similarly, A' generates negative examples for the ith run of A by drawing x' such that the ith bit of $f(k,x')$ is 0. If after $O(1/\epsilon \ln n/\delta)$ draws from Q_k, A' is unable to obtain a positive (respectively, negative) example for A, then A' assumes that with high probability a random x' results in the ith bit of $f(k,x')$ being 0 (respectively, 1), and terminates this run by setting h_k^i to the hypothesis that is always 0 (respectively, 1). The probability that A' terminates the run incorrectly can be shown to be smaller than δ/n by application of Fact CB1 and Fact CB2.

Note that all of the examples given to the ith run of A are consistent with a representation in C_n, since the ith bit of $f(k,\cdot)$ is computed by the representation $c^i_{(k,k')}$. Thus with high probability A outputs an ϵ-good hypothesis h_k^i. To invert the original input $f(k,x)$, A' simply outputs the bit sequence $h_k^1(f(k,x)) \cdots h_k^n(f(k,x))$. The probability that any bit of this string differs from the corresponding bit of x is at most $n\epsilon < \alpha(n)$, contradicting the assumption that (P,Q,f) is an $\alpha(n)$-strong trapdoor scheme. □

7.6 Application: hardness results for approximation algorithms

In this section, we digress from learning briefly and apply the results of Section 7.4 to prove that under cryptographic assumptions, certain combinatorial optimization problems, including a natural generalization of graph coloring, cannot be efficiently approximated even in a very weak sense. These results show that for these problems, it is difficult to find a solution that approximates the optimal solution even within a factor that grows rapidly with the input size. Such results are infrequent in complexity theory, and seem difficult to obtain for natural problems using presumably weaker assumptions such as

$P \neq NP$.

Let C and H be polynomially evaluatable parameterized Boolean representation classes, and define the *Consistency Problem* $Con(C, H)$ as follows:

The Consistency Problem $Con(C, H)$**:**

Input: A labeled sample S of some $c \in C_n$.

Output: $h \in H_n$ such that h is consistent with S and $|h|$ is minimized.

We use $opt_{Con}(S)$ to denote the size of the smallest hypothesis in H that is consistent with the sample S, and $|S|$ to denote the number of bits in S. Using the results of Section 7.4 and Theorem 3.1 of Section 3.2, we immediately obtain proofs of the following theorems.

Theorem 7.8 *Let* BF_n *denote the class of Boolean formulae over* n *variables, and let* $\mathrm{BF} = \cup_{n \geq 1} \mathrm{BF}_n$. *Let* H *be any polynomially evaluatable parameterized Boolean representation class. Then the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to the problem of approximating the optimal solution of an instance* S *of* $Con(\mathrm{BF}, H)$ *by an hypothesis* h *satisfying*

$$|h| \leq (opt_{Con}(S))^{\alpha} |S|^{\beta}$$

for any $\alpha \geq 1$ *and* $0 \leq \beta < 1$.

Theorem 7.9 *Let* ADFA_n *denote the class of deterministic finite automata accepting only strings of length* n, *and let* $\mathrm{ADFA} = \cup_{n \geq 1} \mathrm{ADFA}_n$. *Let* H *be any polynomially evaluatable parameterized Boolean representation class. Then inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to approximating the optimal solution of an instance* S *of* $Con(\mathrm{ADFA}, H)$ *by an hypothesis* h *satisfying*

$$|h| \leq (opt_{Con}(S))^{\alpha} |S|^{\beta}$$

for any $\alpha \geq 1$ *and* $0 \leq \beta < 1$.

Theorem 7.10 *Let $d\mathrm{TC}_n$ denote the class of threshold circuits over n variables with depth at most d, and let $d\mathrm{TC} = \cup_{n \geq 1} d\mathrm{TC}_n$. Let H be any polynomially evaluatable parameterized Boolean representation class. Then for some constant $d \geq 1$, the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to the problem of approximating the optimal solution of an instance S of $Con(d\mathrm{TC}, H)$ by an hypothesis h satisfying*

$$|h| \leq (opt_{Con}(S))^{\alpha} |S|^{\beta}$$

for any $\alpha \geq 1$ and $0 \leq \beta < 1$.

These theorems demonstrate that the results of Section 7.4 are in some sense not dependent upon the particular models of learnability that we study, since we are able to restate the hardness of learning in terms of standard combinatorial optimization problems. Using a generalization of Theorem 3.1, we can in fact prove Theorems 7.8, 7.9 and 7.10 for the *Relaxed Consistency Problem*, where the hypothesis found must agree with only a fraction $1/2 + 1/p(opt_{Con}(S), n)$ for any fixed polynomial p. Using the results of Goldreich et al. [45], it is also possible to show similar hardness results for the Boolean circuit consistency problem $Con(\mathrm{CKT}, \mathrm{CKT})$ using the weaker assumption that there exists a one-way function.

Note that Theorem 7.10 addresses the optimization problem $Con(d\mathrm{TC}, \mathrm{TC})$ as a special case. This problem is essentially that of finding a set of weights in a neural network that yields the desired input-output behavior, sometimes referred to as the *loading problem*. Theorem 7.10 states that even if we allow a much larger net than is actually required, finding these weights is computationally intractable, even for only a constant number of "hidden layers". This result should be contrasted with those of Judd [57] and Blum and Rivest [22], which rely on the weaker assumption $P \neq NP$ but do not prove hardness for relaxed consistency and do not allow the hypothesis network to be substantially larger than the smallest consistent network. We also make no assumptions on the topology of the output circuit.

Theorems 7.8, 7.9 and 7.10 are interesting for at least two reasons. First, they suggest that it is possible to obtain stronger hardness results for combinatorial optimization approximation algorithms by using stronger complexity-theoretic assumptions. Such results seem difficult to obtain using only the

assumption $P \neq NP$. Second, these results provide us with natural examples of optimization problems for which it is hard to approximate the optimal solution even within a multiplicative factor that grows as a function of the input size. Several well-studied problems apparently have this property, but little has been proven in this direction. Perhaps the best example is graph coloring, where the best polynomial-time algorithms require approximately $n^{1-1/(k-1)}$ colors on k-colorable n-vertex graphs (see Wigderson [101] and Blum [19]) but coloring has been proven NP-hard only for $(2-\epsilon)k$ colors for any $\epsilon > 0$ (see Garey and Johnson [39]). Thus for 3-colorable graphs we only know that 5-coloring is hard, but the best algorithm requires roughly $O(n^{0.4})$ colors on n-vertex graphs! This leads us to look for approximation-preserving reductions from our provably hard optimization problems to other natural problems.

We now define a class of optimization problems that we call *formula coloring* problems. Here we have variables $y_1, \ldots, y_m$ assuming natural number values, or *colors*. We regard an assignment of colors to the y_i (called a *coloring*) as a partition P of the variable set into equivalence classes; thus two variables have the same color if and only if they are in the same equivalence class. We consider Boolean formulae that are formed using the standard basis over atomic elements of the form $(y_i = y_j)$ and $(y_i \neq y_j)$, where the predicate $(y_i = y_j)$ is satisfied if and only if y_i and y_j are assigned the same color.

A *model* for such a formula $F(y_1, \ldots, y_m)$ is a coloring of the variables $y_1, \ldots, y_m$ such that F is satisfied. A *minimum model* for the F is a model using the fewest colors. For example, the formula

$$(y_1 = y_2) \vee ((y_1 \neq y_2) \wedge (y_3 \neq y_4))$$

has as a model the two-color partition $\{y_1, y_3\}, \{y_2, y_4\}$ and has as a minimum model the one-color partition $\{y_1, y_2, y_3, y_4\}$.

We will be interested in the problem of finding minimum models for certain restricted classes of formulae. For $F(y_1, \ldots, y_m)$ a formula as described above, and P a model of F, we let $|P|$ denote the number of colors in P and $opt_{FC}(F)$ the number of colors in a minimum model of F.

We first show how graph coloring can be exactly represented as a formula coloring problem. If G is a graph, then for each edge (v_i, v_j) in G, we conjunct the expression $(y_i \neq y_j)$ to the formula $F(G)$. Then $opt_{FC}(F(G))$ is exactly the number of colors required to color G. Similarly, by conjuncting expressions

of the form

$$((y_1 \neq y_2) \vee (y_1 \neq y_3) \vee (y_2 \neq y_3))$$

we can also exactly represent the *3-hypergraph coloring* problem (where each hyperedge contains 3 vertices) as a formula coloring problem.

To prove our hardness results, we consider a generalization of the graph coloring problem:

The Formula Coloring Problem FC**:**

Input: A formula $F(y_1, \ldots, y_m)$ which is a conjunction only of expressions of the form $(y_i \neq y_j)$ (as in the graph coloring problem) *or* of the form $((y_i \neq y_j) \vee (y_k = y_l))$.

Output: A minimum model for F.

Theorem 7.11 *There is a polynomial-time algorithm A that on input an instance S of the problem Con*(ADFA, ADFA) *outputs an instance $F(S)$ of the formula coloring problem such that S has a k-state consistent hypothesis $M \in$ ADFA if and only if $F(S)$ has a model of k colors.*

Proof: Let S contain the labeled examples

$$< w_1, b_1 >, < w_2, b_2 >, \ldots, < w_m, b_m >$$

where each $w_i \in \{0,1\}^n$ and $b_i \in \{0,1\}$. Let w_i^j denote the jth bit of w_i. We create a variable z_i^j for each $1 \leq i \leq n$ and $0 \leq j \leq m$. Let M be a smallest DFA consistent with S. Then we interpret z_i^j as representing the state that M is in immediately after reading the bit w_i^j on input w_i. The formula $F(S)$ will be over the z_i^j and is constructed as follows: for each i_1, i_2 and j_1, j_2 such that $0 \leq j_1, j_2 < n$ and $w_{i_1}^{j_1+1} = w_{i_2}^{j_2+1}$ we conjunct the predicate

$$((z_{i_1}^{j_1} = z_{i_2}^{j_2}) \rightarrow (z_{i_1}^{j_1+1} = z_{i_2}^{j_2+1}))$$

to $F(S)$. Note that this predicate is equivalent to

$$((z_{i_1}^{j_1} \neq z_{i_2}^{j_2}) \vee (z_{i_1}^{j_1+1} = z_{i_2}^{j_2+2}))$$

and thus has the required form. These formulae are designed to encode the constraint that if M is in the same state in two different computations on input strings from S, and the next input symbol is the same in both strings, then the next state in each computation must be the same.

For each i_1, i_2 such that $b_{i_1} \neq b_{i_2}$ we conjunct the predicate $(z_{i_1}^n \neq z_{i_2}^n)$. These predicates are designed to encode the constraint that the input strings in S that are accepted by M must result in different final states than those strings in S that are rejected by M.

We first prove that if M has k states, then $opt_{FC}(F(S)) \leq k$. In particular, let P be the k-color partition that assigns $z_{i_1}^{j_1}$ and $z_{i_2}^{j_2}$ the same color if and only if M is in the same state after reading $w_{i_1}^{j_1}$ on input w_{i_1} and after reading $w_{i_2}^{j_2}$ on input w_{i_2}. We show that P is a model of $F(S)$. A conjunct

$$((z_{i_1}^{j_1} = z_{i_2}^{j_2}) \rightarrow (z_{i_1}^{j_1+1} = z_{i_2}^{j_2+1}))$$

of $F(S)$ cannot be violated by P since this conjunct appears only if $w_{i_1}^{j_1+1} = w_{i_2}^{j_2+1}$; thus if state $z_{i_1}^{j_1}$ is equivalent to state $z_{i_2}^{j_2}$ then state $z_{i_1}^{j_1+1}$ must be equivalent to state $z_{i_2}^{j_2+1}$ since M is deterministic. A conjunct

$$(z_{i_1}^n \neq z_{i_2}^n)$$

of $F(S)$ cannot be violated by P since this conjunct appears only if $b_{i_1} \neq b_{i_2}$, and if state $z_{i_1}^n$ is equivalent to state $z_{i_2}^n$ then w_{i_1} and w_{i_2} are either both accepted or both rejected by M, which contradicts M being consistent with S.

For the other direction, we show that if $opt_{FC}(F(S)) \leq k$ then there is a k-state DFA M' that is consistent with S. M' is constructed as follows: the k states of M' are labeled with the k equivalence classes (colors) $X_1, \ldots X_k$ of the variables z_i^j in a minimum model P' for $F(S)$. There is a transition from state X_p to state X_q if and only if there are i, j such that $z_i^j \in X_p$ and $z_i^{j+1} \in X_q$; this transition is labeled with the symbol w_i^{j+1}. We label X_p an *accepting* (respectively, *rejecting*) state if for some variable $z_i^n \in X_p$ we have $b_i = 1$ (respectively, $b_i = 0$).

We first argue that no state X_p of M' can be labeled both an accepting and rejecting state. For if $b_i = 1$ and $b_j = 0$ then the conjunct $(z_i^n \neq z_j^n)$ appears in $F(S)$, hence z_i^n and z_j^n must have different colors in P'.

Next we show that M is in fact deterministic. For suppose that some state X_p has transitions to X_q and X_r, and that both transitions are labeled with the same symbol. Then there exist i_1, i_2 and j_1, j_2 such that $z_{i_1}^{j_1} \in X_p$ and $z_{i_1}^{j_1+1} \in X_q$, and $z_{i_2}^{j_2} \in X_p$ and $z_{i_2}^{j_2+1} \in X_r$. Furthermore we must have $w_{i_1}^{j_1+1} = w_{i_2}^{j_2+1}$ since both transition have the same label. But then the conjunct

$$((z_{i_1}^{j_1} = z_{i_2}^{j_2}) \rightarrow (z_{i_1}^{j_1+1} = z_{i_2}^{j_2+1}))$$

must appear in $F(S)$, and this conjunct is violated P', a contradiction. Thus M' is deterministic.

These arguments prove that M' is a well-defined DFA. To see that M' is consistent with S, consider the computation of M' on any w_i in S. The sequence of states visited on this computation is just $EC_{P'}(z_i^1), \ldots, EC_{P'}(z_i^n)$, where $EC_{P'}(z_i^j)$ denotes the equivalence class of the variable z_i^j in the coloring P'. The final state $EC_{P'}(z_i^n)$ is by definition of M' either an accept state or a reject state according to whether $w_i^n = 1$ or $w_i^n = 0$. □

Note that if $|S|$ is the number of bits in the sample S and $|F(S)|$ denotes the number of bits in the formula $F(S)$, then in Theorem 7.11 we have $|F(S)| = \Theta(|S|^2 \log |S|) = O(|S|^{2+\gamma})$ for any $\gamma > 0$. Thus by Theorems 7.9 and 7.11 we have:

Theorem 7.12 *The problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are polynomial-time reducible to approximating the optimal solution to an instance F of the formula coloring problem by a model P of F satisfying*

$$|P| \leq opt_{FC}(F)^{\alpha} |F|^{\beta}$$

for any $\alpha \geq 1$ and $0 \leq \beta < 1/2$.

Figure 7.1 summarizes hardness results for coloring a formula F using at most $f(opt_{FC}(F))g(|F|)$ colors for various functions f and g, where an entry "NP-hard" indicates that such an approximation is NP-hard, "Factoring" indicates that such an approximation is as hard as factoring Blum integers (or recognizing quadratic residues or inverting the RSA function), and "P" indicates there is a polynomial-time algorithm achieving this approximation factor. The NP-hardness results follow from Garey and Johnson [39] and Pitt and Warmuth [80].

Difficulty of coloring F using $A \cdot B$ colors	$A = 1$	$A = \|F\|^{1/29}$	$A = \|F\|^{0.499\ldots}$	$A = \|F\|$
$B = opt_{FC}(F)$	NP-hard	NP-hard	Factoring	P
$B = 1.99\ldots opt_{FC}(F)$	NP-hard	Factoring	Factoring	P
$B = (opt_{FC}(F))^{\alpha}$ any fixed $\alpha \geq 0$	NP-hard	Factoring	Factoring	P

Figure 7.1: Difficulty of approximating the formula coloring problem using at most $A \cdot B$ colors on input formula F.

8

Distribution-specific Learning in Polynomial Time

8.1 Introduction

We have seen that for several natural representation classes, the learning problem is computationally intractable (modulo various complexity-theoretic assumptions), in some cases even if we allow arbitrary polynomially evaluatable hypothesis representations. In other cases, perhaps most notably the class of polynomial-size DNF formulae, researchers have been unable to provide firm evidence for either polynomial learnability or for the intractability of learning. Given this state of affairs, we seek to obtain partial positive results by weakening our demands on a learning algorithm, thus making either the computational problem easier (in cases such as Boolean formulae, where we already have strong evidence for the intractability of learning) or the mathematical problem easier (in cases such as DNF, where essentially nothing is currently known). This approach has been pursued in at least two directions: by providing learning algorithms with *additional information* about the target concept in the form of queries, and by relaxing the demand for performance against arbitrary target distributions to that of performance against *specific* natural distributions. In this section we describe results in the latter direction. Other recent distribution-specific learning algorithms include those of Linial, Mansour and Nisan [71] and Kearns and Pitt [62].

We describe polynomial-time algorithms for learning under uniform distributions representation classes for which the learning problem under arbitrary

distributions is either intractable or unresolved. We begin with an algorithm for weakly learning the class of *all* monotone Boolean functions under uniform target distributions in polynomial time; note that here we make no restrictions on the "size" of the function in any particular representation or encoding scheme. We will argue below that in some sense this result is the best possible positive result for learning the class of all monotone functions.

8.2 A polynomial-time weak learning algorithm for all monotone Boolean functions under uniform distributions

We begin with some preliminary definitions and a needed lemma.

For $T \subseteq \{0,1\}^n$ and $\vec{u}, \vec{v} \in \{0,1\}^n$ define

$$\vec{u} \oplus \vec{v} = (u_1 \oplus v_1, \ldots, u_n \oplus v_n)$$

and $T \oplus \vec{v} = \{\vec{u} \oplus \vec{v} : \vec{u} \in T\}$. For $1 \leq i \leq n$ let $\vec{e_i}$ be the vector with the ith bit set to 1 and all other bits set to 0.

The following lemma is due to Aldous [4].

Lemma 8.1 *(Aldous [4]) Let $T \subset \{0,1\}^n$ be such that $|T| \leq 2^n/2$. Then for some $1 \leq i \leq n$,*

$$|T \oplus \vec{e_i} - T| \geq \frac{|T|}{2n}.$$

Armed with this lemma, we can prove the following theorem:

Theorem 8.2 *The class of all monotone Boolean functions is polynomially weakly learnable under uniform D^+ and uniform D^-.*

Proof: Let f be any monotone Boolean function on $\{0,1\}^n$. First assume that $|pos(f)| \leq 2^n/2$. For $\vec{v} \in \{0,1\}^n$ and $1 \leq i \leq n$, let $\vec{v}[i = b]$ denote $\vec{v}$ with the ith bit set to $b \in \{0,1\}$.

Now suppose that $\vec{v} \in \{0,1\}^n$ is such that $\vec{v} \in neg(f)$ and $v_j = 1$ for some $1 \leq j \leq n$. Then $\vec{v}[j = 0] \in neg(f)$ by monotonicity of f. Thus for any $1 \leq j \leq n$ we must have

$$\mathbf{Pr}_{\vec{v} \in D^-}[v_j = 1] \leq \frac{1}{2} \tag{8.1}$$

since D^- is uniform over $neg(f)$.

Let $\vec{e}_i$ be the vector satisfying $|pos(f) \oplus \vec{e}_i - pos(f)| \geq |pos(f)|/2n$ in Lemma 8.1 above. Let $\vec{v} \in \{0,1\}^n$ be such that $\vec{v} \in pos(f)$ and $v_i = 0$. Then $\vec{v}[i = 1] \in pos(f)$ by monotonicity of f. However, by Lemma 8.1, the number of $\vec{v} \in pos(f)$ such that $v_i = 1$ and $\vec{v}[i = 0] \in neg(f)$ is at least $|pos(f)|/2n$. Thus, we have

$$\mathbf{Pr}_{\vec{v} \in D^+}[v_i = 1] \geq \frac{1}{2} + \frac{1}{4n}. \tag{8.2}$$

Similarly, if $|neg(f)| \leq 2^n/2$, then for any $1 \leq j \leq n$ we must have

$$\mathbf{Pr}_{\vec{v} \in D^+}[v_j = 0] \leq \frac{1}{2} \tag{8.3}$$

and for some $1 \leq i \leq n$,

$$\mathbf{Pr}_{\vec{v} \in D^-}[v_i = 0] \geq \frac{1}{2} + \frac{1}{4n}. \tag{8.4}$$

Note that either $|pos(f)| \leq 2^n/2$ or $|neg(f)| \leq 2^n/2$.

We use these differences in probabilities to construct a polynomial-time weak learning algorithm A. A first assumes $|pos(f)| \leq 2^n/2$; if this is the case, then Equations 8.1 and 8.2 must hold. A then finds an index $1 \leq k \leq n$ satisfying

$$\mathbf{Pr}_{\vec{v} \in D^+}[v_k = 1] \geq \frac{1}{2} + \frac{1}{8n} \tag{8.5}$$

The existence of such a k is guaranteed by Equation 8.2. A finds such a k with high probability by sampling POS enough times according to Fact CB1 and Fact CB2 to obtain an estimate $\hat{p}$ of $\mathbf{Pr}_{\vec{v} \in D^+}[v_k = 1]$ satisfying

$$\mathbf{Pr}_{\vec{v} \in D^+}[v_k = 1] - \frac{1}{8n} < \hat{p} < \mathbf{Pr}_{\vec{v} \in D^+}[v_k = 1] + \frac{1}{8n}.$$

If A successfully identifies an index k satisfying Equation 8.5, then the hypothesis h_A is defined as follows: given an unlabeled input vector $\vec{v}$, h_A flips

a biased coin and with probability $1/16n$ classifies $\vec{v}$ as negative; this is to "spread" some of the advantage obtained to the distribution D^-. With probability $1-1/16n$, h_A classifies $\vec{v}$ as positive if $v_i = 1$ and as negative if $v_i = 0$. It is easy to verify by Equations 8.1 and 8.5 that this is a randomized hypothesis meeting the conditions of weak learnability.

If A is unable to identify an index k satisfying Equation 8.5, then A assumes that $|neg(f)| \leq 2^n/2$, and in a similar fashion proceeds to form a hypothesis h_A based on the differences in probability of Equations 8.3 and 8.4. □

It can be shown using Theorem 6.10 that the class of monotone Boolean functions is not polynomially weakly learnable under arbitrary target distributions, since the Vapnik-Chervonenkis dimension of this class is exponential in n. It can also be shown that the class of monotone Boolean functions is not polynomially (strongly) learnable under uniform target distributions. Theorem 6.10 can also be used to show that the class of all Boolean functions is not polynomially weakly learnable under uniform target distributions. Thus, Theorem 8.2 is optimal in the sense that generalization in any direction — uniform distributions to arbitrary distributions, weak learning to strong learning, or monotone functions to arbitrary functions — results in intractability.

Another interesting interpretation of Theorem 8.2 is that functions that are cryptographically secure with respect to the uniform distribution (e.g., trapdoor functions used in secure message exchange such as RSA and quadratic residues) *must* be non-monotone. It would be interesting to find other such general properties that are prerequisites for cryptographic security.

8.3 A polynomial-time learning algorithm for μDNF under uniform distributions

We next give a polynomial-time algorithm for learning DNF in which each variable occurs at most once (μDNF, sometimes also called *read-once* DNF) under uniform target distributions. Recall that in the distribution-free setting, this learning problem is as hard as the general DNF learning problem by Corollary 4.11. Recently Linial, Mansour and Nisan have given a sub-exponential time algorithm for learning general DNF under the uniform distribution [71];

see also the paper of Verbeurgt [98].

Theorem 8.3 μDNF *is polynomially learnable by* μDNF *under uniform* D^+ *and uniform* D^-.

Proof: Let $f = T_1 + \cdots T_s$ be the target μDNF formula over n variables, where each T_i is a monomial. Note that $s \leq n$ since no variable appears twice in f. Let d be such that $n^d = 1/\epsilon$. We say that a monomial T appearing in f is *significant* if $\mathbf{Pr}_{\vec{v} \in D^+}[\vec{v} \in pos(m)] \geq \epsilon/4n = 1/4n^{d+1}$. Thus the error on D^+ incurred by ignoring all monomials that are not significant is at most $\epsilon/4$. We now give an outline of the learning algorithm and then show how each step can be implemented and prove its correctness. For simplicity, our algorithm assumes that the target formula is monotone; this restriction is easily removed, because since each variable appears at most once we may simply regard any occurrence of $\overline{x}_i$ as an unnegated occurrence of a new variable y_i.

Algorithm A:

Step 1. Assume that every significant monomial in f has at least $r \log n$ literals for $r = 2d$. This step will learn an approximation for f using only positive examples if this assumption is correct. If this assumption is not correct, then we will discover this in Step 2, and learn correctly in Step 3 (using only negative examples). The substeps of Step 1 are outlined as follows:

Substep 1.1. For each i, use positive examples to determine whether the variable x_i appears in one of the significant monomials of f.

Substep 1.2. For each i, j such that variables x_i and x_j were determined in Substep 1.1 to appear in some significant monomial of f, use positive examples to decide whether they appear in the same significant monomial.

Substep 1.3. Form a μDNF hypothesis h_A in the obvious way.

Step 2. Decide whether h_A is an ϵ-good hypothesis by testing it on a polynomial number of positive and negative examples. If it is decided that h_A is ϵ-good, stop and output h_A. Otherwise, guess that the assumption of Step 1 is not correct and go to Step 3.

Step 3. Assuming that some significant monomial in f is shorter than $r \log n$, we can also assume that all the monomials are shorter than $2r \log n$, since the longer ones are not significant. We use only negative examples in this step. The substeps are:

Substep 3.1. For each i, use negative examples to determine whether variable x_i appears in some significant monomial of f.

Substep 3.2. For each i, j such that variables x_i and x_j were determined in Substep 3.1 to appear in some significant monomial, use negative examples to decide if they appear in the same significant monomial.

Substep 3.3. Form a μDNF hypothesis h_A in the obvious way and stop.

Throughout the following analysis, we will make use of the following fact: let E_1 and E_2 be events over a probability space, and let $\mathbf{Pr}[E_1 \cup E_2] = 1$ with respect to this probability space. Then for any event E, we have

$$\begin{aligned}
\mathbf{Pr}[E] &= \mathbf{Pr}[E|E_1]\mathbf{Pr}[E_1] + \mathbf{Pr}[E|E_2]\mathbf{Pr}[E_2] \\
&\quad -\mathbf{Pr}[E|E_1 \cap E_2]\mathbf{Pr}[E_1 \cap E_2] \\
&= \mathbf{Pr}[E|E_1]\mathbf{Pr}[E_1] + \mathbf{Pr}[E|E_2](1 - \mathbf{Pr}[E_1] + \mathbf{Pr}[E_1 \cap E_2]) \\
&\quad -\mathbf{Pr}[E|E_1 \cap E_2]\mathbf{Pr}[E_1 \cap E_2] \\
&= \mathbf{Pr}[E|E_1]\mathbf{Pr}[E_1] + \mathbf{Pr}[E|E_2](1 - \mathbf{Pr}[E_1]) \pm O(\mathbf{Pr}[E_1 \cap E_2]) \quad (8.6)
\end{aligned}$$

where here we are assuming that $\mathbf{Pr}[E_1 \cap E_2]$ will depend on the number of variables n.

In Step 1, we draw only positive examples. Since there are at most n (disjoint) monomials in f, and we assume that the size of each monomial is at least $r \log n$, the probability that a positive example of f drawn at random from the uniform D^+ satisfies 2 or more monomials of f is at most $n/2^{r \log n} = 1/n^{r-1} << \epsilon$. Therefore, in the following analysis, we restrict our attention to positive examples of f which satisfy precisely one monomial of f.

Analysis of Substep 1.1. For each i, if the variable x_i is not in any monomial of f,

$$\mathbf{Pr}_{\vec{v} \in D^+}[v_i = 0] = \mathbf{Pr}_{\vec{v} \in D^+}[v_i = 1] = \frac{1}{2}$$

since D^+ is uniform. Now suppose that variable x_i appears in a significant monomial m of f. Then we have

$$\begin{aligned}\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1] &= \mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1|\vec{v} \in pos(m)]\mathbf{Pr}_{\vec{v}\in D^+}[\vec{v} \in pos(m)] \\ &\quad + \mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1|\vec{v} \in neg(m)]\mathbf{Pr}_{\vec{v}\in D^+}[\vec{v} \in neg(m)] \\ &= \mathbf{Pr}_{\vec{v}\in D^+}[\vec{v} \in pos(m)] + (\frac{1}{2} - \frac{1}{n^r})(1 - \mathbf{Pr}_{\vec{v}\in D^+}[\vec{v} \in pos(m)]) \\ &\geq \frac{1}{2} + \frac{1}{2n^{d+1}}. \end{aligned} \tag{8.7}$$

Thus there is a difference of $\Omega(1/n^{d+1})$ between the probability that a variable appearing in a significant monomial is set to 1 and the probability that a variable not appearing in f is set to 1. Notice that if x_i appears in a monomial that is not significant then we simply think that x_i appears in no monomial of f. Using Facts CB1 and CB2, we can determine with high probability if x_i appears in a significant monomial of f by drawing a polynomial number of examples from POS.

Analysis of Substep 1.2. For each pair of variables x_i and x_j that appear in some monomial of f (as decided in Substep 1.1), we now decide whether they appear in the same monomial of f.

Lemma 8.4 *If variables x_i and x_j appear in the same monomial of f, then*

$$\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \text{ or } v_j = 1] = \frac{3}{4} + \frac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1] - \frac{1}{2}) \pm O(\frac{1}{n^{r-1}}).$$

Proof: Since x_i and x_j appear in the same monomial of f and appear only once in f, we have $\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1] = \mathbf{Pr}_{\vec{v}\in D^+}[v_j = 1]$ since D^+ is uniform. Let m be the monomial of f in which x_i and x_j appear, and let E_1 be the event that m is satisfied. Let E_2 be the event that at least one monomial of f besides (but possibly in addition to) m is satisfied. Note that $\mathbf{Pr}_{\vec{v}\in D^+}[E_1 \cup E_2] = 1$. Using the facts that since D^+ is uniform, $\mathbf{Pr}_{\vec{v}\in D^+}[E_1 \cap E_2] \leq 1/n^{r-1}$ (because given that a positive example already satisfies a monomial of f, the remaining variables are independent and uniformly distributed) and $\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1] = \mathbf{Pr}_{\vec{v}\in D^+}[E_1] + 1/2(1 - \mathbf{Pr}_{\vec{v}\in D^+}[E_1])$ and by Equation 8.6, we have

$$\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \text{ or } v_j = 1] = \mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \text{ or } v_j = 1|E_1]\mathbf{Pr}_{\vec{v}\in D^+}[E_1]$$

$$
\begin{aligned}
&+\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \text{ or } v_j = 1|E_2]\mathbf{Pr}_{\vec{v}\in D^+}[E_2] - O(\frac{1}{n^{r-1}}) \\
= \;& \mathbf{Pr}_{\vec{v}\in D^+}[E_1] + \frac{3}{4}(1 - \mathbf{Pr}_{\vec{v}\in D^+}[E_1]) \pm O(\frac{1}{n^{r-1}}) \\
= \;& \frac{3}{4} + \frac{1}{4}\mathbf{Pr}_{\vec{v}\in D^+}[E_1] \pm O(\frac{1}{n^{r-1}}) \\
= \;& \frac{3}{4} + \frac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1] - \frac{1}{2}) \pm O(\frac{1}{n^{r-1}}).
\end{aligned}
$$

□(Lemma 8.4)

Lemma 8.5 *If variables x_i and x_j appear in different monomials of f, then*

$$
\begin{aligned}
\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \text{ or } v_j = 1] = \frac{3}{4} + \frac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1] - \frac{1}{2}) \\
+\frac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}[v_j = 1] - \frac{1}{2}) \pm O(\frac{1}{n^{r-1}}).
\end{aligned}
$$

Proof: Let E_1 be the event that the monomial T_1 of f containing x_i is satisfied, and E_2 the event that the monomial T_2 containing x_j is satisfied. Let E_3 be the event that some monomial other than (but possibly in addition to) T_1 and T_2 is satisfied. Note that $\mathbf{Pr}_{\vec{v}\in D^+}[E_1 \cup E_2 \cup E_3] = 1$. Then similar to the proof of Lemma 8.4, we have

$$
\begin{aligned}
&\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \text{ or } v_j = 1] \\
= \;& \mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \text{ or } v_j = 1|E_1]\mathbf{Pr}_{\vec{v}\in D^+}[E_1] \\
&+\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \text{ or } v_j = 1|E_2]\mathbf{Pr}_{\vec{v}\in D^+}[E_2] \\
&+\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \text{ or } v_j = 1|E_3]\mathbf{Pr}_{\vec{v}\in D^+}[E_3] - O(\frac{1}{n^{r-1}}) \\
= \;& \mathbf{Pr}_{\vec{v}\in D^+}[E_1] + \mathbf{Pr}_{\vec{v}\in D^+}[E_2] + \frac{3}{4}\mathbf{Pr}_{\vec{v}\in D^+}[E_3] - O(\frac{1}{n^{r-1}}) \\
= \;& \mathbf{Pr}_{\vec{v}\in D^+}[E_1] + \mathbf{Pr}_{\vec{v}\in D^+}[E_2] \\
&+\frac{3}{4}(1 - \mathbf{Pr}_{\vec{v}\in D^+}[E_1] - \mathbf{Pr}_{\vec{v}\in D^+}[E_2]) \pm O(\frac{1}{n^{r-1}}) \\
= \;& \frac{3}{4} + \frac{1}{4}(\mathbf{Pr}_{\vec{v}\in D^+}[E_1] + \mathbf{Pr}_{\vec{v}\in D^+}[E_2]) \pm O(\frac{1}{n^{r-1}}) \\
= \;& \frac{3}{4} + \frac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1] - \frac{1}{2}) \\
&+\frac{1}{2}(\mathbf{Pr}_{\vec{v}\in D^+}[v_j = 1] - \frac{1}{2}) \pm O(\frac{1}{n^{r-1}}).
\end{aligned}
$$

□(Lemma 8.5)

From Equation 8.7, Lemma 8.4, Lemma 8.5 and the fact that if x_i and x_j appear in the same monomial of f, then $\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1] = \mathbf{Pr}_{\vec{v}\in D^+}[v_j = 1]$, we have that there is a difference $\Omega(1/n^{d+1} - 1/n^{r-1})$ between the value of $\mathbf{Pr}_{\vec{v}\in D^+}[v_i = 1 \; or \; v_j = 1]$ in the two cases addressed by Lemmas 8.4 and 8.5. Thus we can determine whether x_i and x_j appear in the same significant monomial by drawing a polynomial number of examples from *POS* using Facts CB1 and CB2.

In Step 2, we draw a polynomial number of examples from both *POS* and *NEG* to test if the hypothesis h_A produced in Step 1 is ϵ-good, again using Facts CB1 and CB2. If it is determined that h_A is not ϵ-good, then A guesses the assumption made in Step 1 is not correct, and therefore that there is a monomial in f which is of length at most $r \log n$. This implies that all the monomials of length larger than $2r \log n$ are not significant. Therefore in Step 3 we assume that all the monomials in f are shorter than $2r \log n$. We use only the negative examples.

Analysis of Substep 3.1. If variable x_i does not appear in any monomial of f, then

$$\mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0] = \frac{1}{2} \tag{8.8}$$

since D^+ is uniform.

Lemma 8.6 *If variable x_i appears in a significant monomial of f, then*

$$\mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0] \geq \frac{1}{2} + \frac{1}{2(n^{2r} - 1)}.$$

Proof: Let l be the number of literals in the monomial m of f that variable x_i appears in. Then in a vector $\vec{v}$ drawn at random from D^-, if some bit of $\vec{v}$ is set such that m is already not satisfied, the remaining bits are independent and uniformly distributed. Thus

$$\mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0] = \frac{2^{l-1}}{2^l - 1} = \frac{1}{2} + \frac{1}{2(2^l - 1)}.$$

Since $l \leq 2r \log n$, the claim follows. □(Lemma 8.6)

By Equation 8.8 and Lemma 8.6, there is a difference of $\Omega(1/n^{2r})$ between the probability that a variable in a significant monomial of f is set to 0 and the probability that a variable not appearing in f is set to 0. Thus we can draw a polynomial number of examples from NEG, and decide if variable x_i appears in some significant monomial of f, using Facts CB1 and CB2.

Analysis of Substep 3.2. We have to decide whether variables x_i and x_j appear in the same monomial of f, given that each appear in some monomial of f.

Lemma 8.7 *If variables x_i and x_j are not in the same monomial of f, then*

$$\mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0 \text{ and } v_j = 0] = \mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0]\mathbf{Pr}_{\vec{v}\in D^-}[v_j = 0].$$

Proof: If x_i and x_j do not appear in the same monomial, then they are independent of each other with respect to D^- since each variable appears only once in f. □(Lemma 8.7)

Lemma 8.8 *If variables x_i and x_j appear in the same monomial of f, then*

$$\mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0 \text{ and } v_j = 0] = \frac{1}{2}\mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0].$$

Proof:

$$\mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0 \text{ and } v_j = 0] = \mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0]\mathbf{Pr}_{\vec{v}\in D^-}[v_j = 0|v_i = 0]$$

But $\mathbf{Pr}_{\vec{v}\in D^-}[v_j = 0|v_i = 0] = 1/2$. □(Lemma 8.8)

By Lemmas 8.6, 8.7 and 8.8 we have that there is a difference of $\Omega(1/n^{2r})$ in the value of $\mathbf{Pr}_{\vec{v}\in D^-}[v_i = 0 \text{ and } v_j = 0]$ in the two cases addressed by Lemmas 8.7 and 8.8. Thus we can test if x_i and x_j appear in the same monomial of f by drawing a polynomial number of examples from NEG using Facts CB1 and CB2. This completes the proof of Theorem 8.3. □

The results of Pitt and Valiant [78] show that k-term μDNF is not learnable by k-term μDNF unless $NP = RP$. However, the algorithm of Theorem 8.3 outputs an hypothesis with the same number of terms as the target formula;

thus k-term μDNF is learnable by k-term μDNF under uniform target distributions. This is an example of a class for which learning under arbitrary target distributions is NP-hard, but learning under uniform target distributions is tractable, and is analogous to results from complexity theory giving polynomial-time algorithms for NP-complete problems when the inputs are generated from specific probability distributions.

9

Equivalence of Weak Learning and Group Learning

9.1 Introduction

In this chapter we prove the equivalence of the model of weak learning with another model which we call *group learning*. Informally, in group learning we ask that the learning algorithm output an hypothesis that is $1 - \epsilon$ accurate in classifying a polynomial-size *group* of examples that are either all positive or all negative. Thus, the basic model of (strong) learnability is a special case of group learning where the group size is 1. The question we wish to address here is whether learning becomes easier in some cases if the group size is allowed to be larger.

Recently it has been shown by Schapire [90] that in the distribution-free setting, polynomial-time weak learning is in fact equivalent to polynomial-time strong learning. His proof gives a recursive technique for taking an algorithm outputting hypotheses with accuracy slightly above 1/2 and constructing hypotheses of accuracy $1 - \epsilon$. This result combined with ours shows that group learning is in fact equivalent to strong learning. Thus, allowing the hypothesis to accurately classify only larger groups of (all positive or all negative) examples does not increase what is polynomially learnable. These results also demonstrate the robustness of our underlying model of learnability, since it is invariant under these apparently significant but reasonable modifications. Related equivalences are given by Haussler et al. [51].

Our equivalence proof also holds in both directions under *fixed* target distributions: thus, C is polynomially group learnable under a restricted class of distributions if and only if C is polynomially weakly learnable under these same distributions. As an immediate corollary, we have by the results of Section 8.2 that the class of all monotone Boolean functions is group learnable in polynomial time under uniform distributions. Furthermore, since it was argued in Section 8.2 that the class of all monotone Boolean functions cannot be strongly learned in polynomial time under uniform distributions, there cannot be an efficient distribution-preserving reduction of the strong learning model to the weak learning model. Thus, the problem of learning monotone functions under uniform distributions exhibits a trade-off: we may either have accuracy slightly better than guessing on single examples, or high accuracy on polynomial-size groups of examples, but not high accuracy on single examples.

Our formal definitions are as follows: for p any fixed polynomial in $1/\epsilon, 1/\delta$ (and n), the hypothesis h_A of learning algorithm A is now defined over the space $X^{p(1/\epsilon,1/\delta,n)}$. We ask that if $p(1/\epsilon, 1/\delta, n)$ examples all drawn from D^+ (respectively, D^-) are given to h_A, then h_A classifies this *group* as positive (respectively, negative) with probability at least $1-\epsilon$. If A runs in polynomial time, we say that C is *polynomially group learnable.*

9.2 The equivalence

Theorem 9.1 *Let C be a polynomially evaluatable parameterized Boolean representation class. Then C is polynomially group learnable if and only if C is polynomially weakly learnable.*

Proof: (If) Let A be a polynomial-time weak learning algorithm for C. We construct a polynomial-time group learning algorithm A' for C as follows: A' first simulates algorithm A and obtains an hypothesis h_A that with probability $1-\delta$ has accuracy $1/2 + 1/p(|c|, n)$ for some polynomial p. Now given m examples that are either all drawn from D^+ or all drawn from D^-, A' evaluates h_A on each example. Suppose all m points are drawn from D^+. Assuming that h_A in fact has accuracy $1/2 + 1/p(|c|, n)$, the probability that h_A evaluates as positive on fewer than $(1/2 + 1/(2p(|c|, n)))m$ of the examples can be made smaller than $\epsilon/2$ by choosing m to be a large enough polynomial in $1/\epsilon$ and

$1/p(|c|,n)$ using Fact CB1. On the other hand, if all m examples are drawn from D^- then the probability that h_A evaluates as positive on more than $(1/2+1/(2p(|c|,n)))m$ can be made smaller than $\epsilon/2$ for m large enough by Fact CB2. Thus, if h_A evaluates as positive on more than $(1/2+1/(2p(|c|,n)))m$ of the m examples, A' guesses that the sample is positive; otherwise, A' guesses that the sample is negative. The probability of misclassifying the sample is then at most ϵ, so A' is a group learning algorithm.

(Only if) Let A be a polynomial-time group learning algorithm for C. We use A as a subroutine in a polynomial-time weak learning algorithm A'. Suppose algorithm A is run to obtain with high probability an $\epsilon/2$-good hypothesis h_A for groups of size $l = p(2/\epsilon, 1/\delta, |c|, n)$ all drawn from D^+ or all drawn from D^- for some polynomial p. Note that although h_A is guaranteed to produce an output only when given l positive examples or l negative examples, the *probability* that h_A produces an output when given a mixture of positive and negative examples is well-defined. Thus for $0 \leq i \leq l$ let q_i denote the probability that h_A evaluates as positive when given as input a group whose first i examples are drawn from D^+ and whose last $l-i$ examples are drawn from D^-. Then since h_A is an $\epsilon/2$-good hypothesis we have $q_0 \leq \epsilon/2$ and $q_l \geq 1-\epsilon/2$. Thus,

$$(q_l - q_0) \geq (1 - \frac{\epsilon}{2}) - \frac{\epsilon}{2} = 1 - \epsilon.$$

Then

$$1 - \epsilon \leq (q_l - q_0) = (q_l - q_{l-1}) + (q_{l-1} - q_{l-2}) + \cdots + (q_1 - q_0).$$

This implies that for some $1 \leq j \leq l$, $(q_j - q_{j-1}) \geq (1-\epsilon)/l$.

Algorithm A' first runs algorithm A with accuracy parameter $\epsilon/2$. A' next obtains an estimate $\hat{q}_i$ of q_i for each $0 \leq i \leq l$ that is accurate within an additive factor of $(1-\epsilon)/16l$, that is

$$q_i - \frac{1-\epsilon}{16l} \leq \hat{q}_i \leq q_i + \frac{1-\epsilon}{16l}. \tag{9.1}$$

This is done by repeatedly evaluating h_A on groups of l examples in which the first i examples are drawn from D^+ and the rest are drawn from D^-, and computing the fraction of runs for which h_A evaluates as positive. These estimates can be obtained in time polynomial in l, $1/\epsilon$ and $1/\delta$ with high probability using Facts CB1 and CB2. Now for the j such that $(q_j - q_{j-1}) \geq$

$(1-\epsilon)/l$, the estimates $\hat{q}_j$ and $\hat{q}_{j-1}$ will have a difference of at least $(1-\epsilon)/2l$ with high probability. Furthermore, for any i if $(q_i - q_{i-1}) \leq (1-\epsilon)/4l$ then with high probability the estimates satisfy $(\hat{q}_i - \hat{q}_{i-1}) \leq (1-\epsilon)/2l$. Thus, if k is the index such that $(q_k - q_{k-1})$ is largest, assume without loss of generality that $1/2 \geq q_k > q_{k-1}$.

The intermediate hypothesis of $h_{A'}$ of A' is now defined as follows: given an example whose classification is unknown, $h_{A'}$ constructs l input examples for h_A consisting of $k-1$ examples drawn from D^+, the unknown example, and $l-k$ examples drawn from D^-. The prediction of $h_{A'}$ is then the same as the prediction of h_A on this constructed group. The probability that $h_{A'}$ predicts positive when the unknown example is drawn from D^+ is then q_k and the probability that $h_{A'}$ predicts positive when the unknown example is drawn from D^- is q_{k-1}.

One problem with the $h_{A'}$ defined at this point is that new examples need to be drawn from D^+ and D^- each time an unknown point is classified. This sampling is eliminated as follows: for U a fixed sequence of $k-1$ positive examples of the target representation and V a fixed sequence of $l-k$ negative examples, define $h_{A'}(U,V)$ to be the one-input intermediate hypothesis described above using the *fixed* constructed sample consisting of U and V. Let $p^+(U,V)$ be the probability that $h_{A'}(U,V)$ classifies a random example drawn from D^+ as positive, and let $p^-(U,V)$ be the probability that $h_{A'}(U,V)$ classifies a random example drawn from D^- as positive. Then for U drawn randomly according to D^+ and V drawn randomly according to D^-, define the random variable

$$R(U,V) = p^+(U,V) - p^-(U,V).$$

Then the expectation of R obeys

$$\mathbf{E}[R(U,V)] \geq 2\alpha$$

where $\alpha = (1-\epsilon)/4l$. However, it is always true that

$$R(U,V) \leq 1.$$

Thus, let r be the probability that

$$R(U,V) \geq \alpha. \tag{9.2}$$

Then we have

$$r + (1 - r)(\alpha) \geq 2\alpha.$$

Solving, we obtain $r \geq \alpha = (1 - \epsilon)/4l$.

Thus, A' repeatedly draws U from D^+ and V from D^- until Equation 9.2 is satisfied; by the above argument, this takes only $8l/(1 - \epsilon)$ tries with high probability. Note that A' can test whether Equation 9.2 is satisfied in polynomial time. The (almost) final hypothesis $h_{A'}(U_0, V_0)$ simply "hard-wires" the successful U_0 and V_0 as the constructed sample, leaving one input free for the example whose classification is to be predicted by $h_{A'}$.

Lastly, we need to "center" the bias of the hypothesis $h_{A'}(U_0, V_0)$. Let b be a value such that

$$b + (1 - b)q_k \geq \frac{1}{2} + \frac{q_k - q_{k-1}}{4}$$

and

$$b + (1 - b)q_{k-1} \leq \frac{1}{2} - \frac{q_k - q_{k-1}}{4}.$$

Note that A' can compute an accurate estimate $\hat{b}$ of b from accurate estimates of q_k and q_{k-1}. The final hypothesis $h_{A'}$ of A' is now defined as follows: given an example whose classification is unknown, $h_{A'}$ flips a coin of bias $\hat{b}$. If the outcome is heads, $h_{A'}$ predicts that the input example is positive. If the outcome is tails, $h_{A'}$ predicts with the classification given by $h_{A'}(U_0, V_0)$. Then we have

$$1 - e^+(h_{A'}) \geq \hat{b} + (1 - \hat{b})q_k \geq \frac{1}{2} + \frac{1 - \epsilon}{c_0 l}$$

for an appropriate constant $c_0 > 1$, and

$$1 - e^-(h_{A'}) \geq 1 - \hat{b} + (1 - \hat{b})q_{k-1} \geq \frac{1}{2} + \frac{1 - \epsilon}{c_0 l}.$$

□

10

Conclusions and Open Problems

In the introduction we stated the hypothesis that many natural learning problems can be well-formalized, and that the tools of the theory of efficient computation can be applied and adapted to these problems. We feel that the results presented here and elsewhere in computational learning theory bear out this hypothesis to a partial but promising extent.

We wish to emphasize that the recent progress in this area represents only the beginning of a theory of efficient machine learning. This is both a cautionary and an optimistic statement. It is a cautionary statement because the unsolved problems far outnumber the solved ones, and the models often fall short of our notions of "reality". It is an optimistic statement because we do have the beginnings of a *theory*, with general techniques being developed and natural structure being uncovered, not simply isolated and unrelated problems being solved by ad-hoc methods.

As humans we have the first-hand experience of being rapid and expert learners. This experience is undoubtedly part of what makes the subject of machine learning fascinating to many researchers. It also tends to make us critical of the models for learning that we propose: as good learners, we have strong opinions, however imprecise, on what demands a good learning algorithm should meet; as poor graph-colorers, for instance, we may be less vehement in our criticism of the best known heuristics. Hopefully researchers will build on the progress made so far and formulate mathematical models of learning that more closely match our first-hand experience.

We hope that the coming years will see significant progress on the issues raised here and elsewhere in computational learning theory, and on issues yet to be raised. To entice the interested reader in this direction, we conclude with a short list of selected open problems and areas for further research.

Efficient learning of DNF. Several of the results in this book focus attention on what is perhaps the most important open problem in the distribution-free model, the polynomial learnability of DNF formulae. More precisely: is the class of polynomial-size DNF learnable in polynomial time by some polynomially evaluatable representation class? Chapter 8 gave positive results only for the case of uniform distributions, and only when the DNF are "read-once". Recently, Linial, Mansour and Nisan [71] gave a sub-exponential but super-polynomial time algorithm for learning DNF against uniform target distributions. Neither of these results seems likely to generalize to the distribution-free setting. On the other hand, the DNF question also seems beyond the cryptographic techniques for proving hardness results of Chapter 7, since those methods require that the hard class of circuits at least be able to perform multiplication. Several apparently computationally easier subproblems also remain unsolved, such as: Is monotone DNF polynomially learnable under uniform distributions? Is DNF polynomially learnable with the basic natural queries (e.g., membership queries) allowed? Are decision trees polynomially learnable by some polynomially evaluatable representation class? Some progress has recently been made by Angluin, Frazier and Pitt [10], who show that the class of DNF in which each term has at most one negated literal can be efficiently learned from random examples and membership queries.

Improved time and sample complexity for learning k-TERM-DNF. Short of solving the polynomial learnability of general DNF, can one improve on the $O(n^k)$ time and sample complexity for learning the class k-TERM-DNF provided by the algorithm of Valiant [93]? Note that we still may have exponential dependence on k (otherwise we have solved the general DNF problem); however, it is entirely plausible that there is, say, an $O(n^{k/2})$ solution using a different hypothesis space than kCNF.

Shift DNF. (Suggested by Petros Maragos and Les Valiant) A potentially manageable subproblem of the general DNF problem is motivated by

machine vision. In the "shift" DNF problem, the target representation over $\{0,1\}^n$ is of the form $T_1 + \cdots + T_n$, where the literal l_i is in the term T_j if and only if literal $l_{i-j+1 \bmod n}$ is in term T_1. We think of the variables $x_1, \ldots, x_n$ as representing pixels in a visual field, and T_1 is a template for some simple object (such as the letter "a"). Then the shift DNF represents the presence of the object somewhere in the visual field. More generally, notice that shift DNF is a class of DNF that is invariant under a certain set of cyclic permutations of the inputs; at the extreme we have that symmetric functions, which are invariant under *all* permutations of the inputs, are efficiently learnable by Theorem 5.11. It would be interesting to investigate the smallest class of permutation invariants that suffice for polynomial learnability.

Improved error rates. Can the polynomial-time malicious error rates given in Chapter 5 be improved? Of particular interest is the class of monomials, the simplest class where the tolerable error rate apparently diminishes as the number of variables increases. Is it possible to prove "representation-independent" bounds on the polynomial-time malicious error rate? Recall that the result relating the error rate for monomials and set cover approximations requires that the learning algorithm outputs a monomial as its hypothesis. It would be nice to remove this restriction, or give an algorithm that tolerates a larger error rate by using a more powerful hypothesis representation.

Cryptography from non-learnability. Results in Chapter 7 demonstrate that the existence of various cryptographically secure functions implies that some simple representation classes are not learnable in polynomial time. It would be quite interesting to prove some sort of partial converse to this. Namely, if a parameterized Boolean representation class C cannot be learned in polynomial time, is it possible to develop protocols for some cryptographic primitives based on C? Note that such results could actually have practical importance for cryptography — one might be able to construct very efficient protocols based on, for instance, the difficulty of learning DNF formulae. Intuitively, the "simpler" the hard class C, the more efficient the protocol, since in Chapter 7 the complexity of the hard class was directly related to the complexity of decryption.

Weakened assumptions for non-learnability results. In the opposite direction of the preceding problem, all known representation-independent

hardness results for learning rely on the existence of one-way or trapdoor functions. It would be interesting to find a polynomially evaluatable representation class that is not polynomially learnable based on ostensibly weaker assumptions such as $RP \neq NP$. Recently Board and Pitt [26] suggested an approach to this problem, but it remains open.

A theory of learning with background information. One frequent complaint about the distribution-free model is its *tabula rasa* approach to learning, in the sense that the learning algorithm has no prior information or experience on which to draw. This is in contrast to human learning, where people often learn hierarchically by building on previously learned concepts, by utilizing provided "background information" about the particular domain of interest, or by relying on a possibly "hard-wired" biological predisposition towards certain abilities. It would be interesting to formulate good models of *efficient* learning in the presence of these valuable and natural sources of information, and to compare the difficulty of learning with and without such sources. Note that the demand for efficiency forces any good model to carefully consider how such background information is represented and processed by a learning algorithm, and one might expect to see trade-offs between the computational expense of processing provided background information and the computational expense of learning using this information. For example, extensive knowledge of abstract algebra almost certainly eases the task of learning basic integer arithmetic, but the effort required to gain this knowledge is not worthwhile if basic arithmetic is the ultimate goal.

Learning with few mistakes. Many of the efficient learning algorithms in the distribution-free model actually have the stronger property of having a small *absolute mistake bound*; that is, they misclassify only a polynomial number of examples, even if an adversary chooses the presentation sequence (see Littlestone [73] for details). On the other hand, Blum [21] has recently demonstrated a representation class that can be efficiently learned in the distribution-free model, but which cannot be learned with a polynomial absolute mistake bound (assuming the existence of a one-way function). Are there still reasonably general conditions under which distribution-free learning implies learning with a small absolute mistake bound? Note that if we relax our demands to that of only having a small *expected* mistake bound, then we obtain equivalence to the distribution-

free model within polynomial factors (see Haussler, Littlestone and Warmuth [52].

Learning more expressive representations. In this book we concentrated exclusively on *concept* learning — that is, learning representations of sets. It is also important to consider efficient learning of more complicated representations than simple $\{0,1\}$-valued functions. Here we would like to avoid simply reducing learning multi-valued functions to learning concepts (for example, by learning each bit of the output separately as a concept). Rather, we would like to explicitly use the more expressive representations to allow simple modeling of more real-world learning scenarios than is possible in the basic concept learning framework. The sample complexity of such generalized learning was recently investigated in great detail by Haussler [50], and efficient learning of real-valued functions whose output is interpreted as the conditional probability that the input is a positive example has been studied by Kearns and Schapire [63]. It would be interesting to examine other settings in which more expressive functions provide more realistic modeling and still permit efficient learning.

Cooperating learning algorithms. (Suggested by Nick Littlestone) Humans often seem able to greatly reduce the time required to learn by communicating and working together in groups. It would be interesting to define a formal model of learning algorithms that are allowed to communicate their hypotheses and/or other information in an attempt to converge on the target more rapidly.

Agnostic learning. A typical feature of pattern recognition and empirical machine learning research is to make very few or no assumptions on how the sample data is generated; thus there is no "target concept", and the goal of a learning algorithm might be to choose the "best" hypothesis from a given hypothesis class (even if this hypothesis is a relatively poor description of the data). We might call this type of learning *agnostic learning*, to emphasize the fact that the learning algorithm has no a priori beliefs regarding the structure of the sample data. Agnostic learning has been studied in a general but non-computational setting by Haussler [50]; it would be interesting to study the possibilities for efficient agnostic learning in the distribution-free model.

Bibliography

[1] N. Abe.
Polynomial learnability of semilinear sets.
Proceedings of the 1989 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1989, pp. 25-40.

[2] L. Adleman, K. Manders, G. Miller.
On taking roots in finite fields.
Proceedings of the 18th I.E.E.E. Symposium on Foundations of Computer Science, 1977, pp. 175-178.

[3] A. Aho, J. Hopcroft, J. Ullman.
The design and analysis of computer algorithms.
Addison-Wesley, 1974.

[4] D. Aldous.
On the Markov chain simulation method for uniform combinatorial distributions and simulated annealing.
University of California at Berkeley Statistics Department, technical report number 60, 1986.

[5] W. Alexi, B. Chor, O. Goldreich, C.P. Schnorr.
RSA and Rabin functions: certain parts are as hard as the whole.
S.I.A.M. Journal on Computing, 17(2), 1988, pp. 194-209.

[6] D. Angluin.
Lecture notes on the complexity of some problems in number theory.
Yale University Computer Science Department, technical report number TR-243, 1982.

[7] D. Angluin.
Learning regular sets from queries and counterexamples.
Information and Computation, 75, 1987, pp. 87-106.

[8] D. Angluin.
Queries and concept learning.
Machine Learning, 2(4), 1988, pp. 319-342.

[9] D. Angluin.
Learning with hints.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 167-181.

[10] D. Angluin, M. Frazier, L. Pitt.
Learning conjunctions of horn clauses.
To appear, *Proceedings of the 31st I.E.E.E. Symposium on the Foundations of Computer Science*, 1990.

[11] D. Angluin, L. Hellerstein, M. Karpinski.
Learning read-once formulas with queries.
University of California at Berkeley Computer Science Department, technical report number 89/528, 1989. Also International Computer Science Institute, technical report number TR-89-05099.

[12] D. Angluin, P. Laird.
Learning from noisy examples.
Machine Learning, 2(4), 1988, pp. 343-370.

[13] D. Angluin, C. Smith.
Inductive inference: theory and methods.
A.C.M. Computing Surveys, 15, 1983, pp. 237-269.

[14] D. Angluin, L.G. Valiant.
Fast probabilistic algorithms for Hamiltonian circuits and matchings.
Journal of Computer and Systems Sciences, 18, 1979, pp. 155-193.

[15] P.W. Beame, S.A. Cook, H.J. Hoover.
Log depth circuits for division and related problems.
S.I.A.M. Journal on Computing, 15(4), 1986, pp. 994-1003.

[16] G.M. Benedek, A. Itai.
Learnability by fixed distributions.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 80-90.

[17] B. Berger, P. Shor, J. Rompel.
Efficient NC algorithms for set cover with applications to learning and geometry.
Proceedings of the 30th I.E.E.E. Symposium on the Foundations of Computer Science, 1989, pp. 54-59.

[18] P. Berman, R. Roos.
Learning one-counter languages in polynomial time.
Proceedings of the 28th I.E.E.E. Symposium on Foundations of Computer Science, 1987, pp. 61-77.

[19] A. Blum.
An $\tilde{O}(n^{0.4})$-approximation algorithm for 3-coloring.
Proceedings of the 21st A.C.M. Symposium on the Theory of Computing, 1989, pp. 535-542.

[20] A. Blum.
Learning in an infinite attribute space.
Proceedings of the 22nd A.C.M. Symposium on the Theory of Computing, 1990, pp. 64-72.

[21] A. Blum.
Separating PAC and mistake-bound learning models over the Boolean domain.
To appear, *Proceedings of the 31st I.E.E.E. Symposium on the Foundations of Computer Science*, 1990.

[22] A. Blum, R.L. Rivest.
Training a 3-node neural network is NP-complete.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 9-18.

[23] M. Blum, S. Micali.
How to generate cryptographically strong sequences of pseudo-random bits.
S.I.A.M. Journal on Computing, 13(4), 1984, pp. 850-864.

[24] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth.
Occam's razor.
Information Processing Letters, 24, 1987, pp. 377-380.

[25] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth.
Learnability and the Vapnik-Chervonenkis dimension.
Journal of the A.C.M., 36(4), 1989, pp. 929-965.

[26] R. Board, L. Pitt.
On the necessity of Occam algorithms.
Proceedings of the 22nd A.C.M. Symposium on the Theory of Computing, 1990, pp. 54-63.

[27] A.K. Chandra, L.J. Stockmeyer, U. Vishkin.
Constant depth reducibility.
S.I.A.M. Journal on Computing, 13(2), 1984, pp. 423-432.

[28] H. Chernoff.
A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations.
Annals of Mathematical Statistics, 23, 1952, pp. 493-509.

[29] V. Chvatal.
A greedy heuristic for the set covering problem.
Mathematics of Operations Research, 4(3), 1979, pp. 233-235.

[30] T.H. Cormen, C.E. Leiserson, R.L. Rivest.
Introduction to algorithms.
The MIT Press, 1990.

[31] L. Devroye.
Automatic pattern recognition: a study of the probability of error.
I.E.E.E. Transactions on Pattern Analysis and Machine Intelligence, 10(4), 1988, pp. 530-543.

[32] W. Diffie, M. Hellman.
New directions in cryptography.
I.E.E.E. Transactions on Information Theory, 22, 1976, pp. 644-654.

[33] R. Duda, P. Hart.
Pattern classification and scene analysis.
John Wiley and Sons, 1973.

[34] R.M. Dudley.
A course on empirical processes.
Lecture Notes in Mathematics, 1097:2-142, 1984.

[35] A. Ehrenfeucht, D. Haussler.
Learning decision trees from random examples.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 182-194.

[36] A. Ehrenfeucht, D. Haussler, M. Kearns. L.G. Valiant.
A general lower bound on the number of examples needed for learning.
Information and Computation, 82(3), 1989, pp. 247-261.

[37] S. Floyd.
On space-bounded learning and the Vapnik-Chervonenkis Dimension.
International Computer Science Institute, technical report number TR-89-061, 1989. See also *Proceedings of the 1988 Workshop on Computational Learning Theory*, Morgan Kaufmann Publishers, 1988, pp. 413-414.

[38] M. Fulk, J. Case, editors.
Proceedings of the 1990 Workshop on Computational Learning Theory.
Morgan Kaufmann Publishers, 1990.

[39] M. Garey, D. Johnson.
Computers and intractability: a guide to the theory of NP-completeness.
Freeman, 1979.

[40] M. Gereb-Graus.
Complexity of learning from one-sided examples.
Harvard University, unpublished manuscript, 1989.

[41] E.M. Gold.
Complexity of automaton identification from given data.
Information and Control, 37, 1978, pp. 302-320.

[42] S. Goldman, M. Kearns, R. Schapire.
Exact identification of circuits using fixed points of amplification functions.
To appear, *Proceedings of the 31st I.E.E.E. Symposium on the Foundations of Computer Science*, 1990.

[43] S. Goldman, M. Kearns, R. Schapire.
On the sample complexity of weak learning.
To appear, *Proceedings of the 1990 Workshop on Computational Learning Theory*, Morgan Kaufmann Publishers, 1990.

[44] S. Goldman, R. Rivest, R. Schapire.
Learning binary relations and total orders.
Proceedings of the 30th I.E.E.E. Symposium on the Foundations of Computer Science, 1989, pp. 46-51.

[45] O. Goldreich, S. Goldwasser, S. Micali.
How to construct random functions.
Journal of the A.C.M., 33(4), 1986, pp. 792-807.

[46] T. Hancock.
On the difficulty of finding small consistent decision trees.
Harvard University, unpublished manuscript, 1989.

[47] T. Hancock.
Learning μ-formula decision trees with queries.
To appear, *Proceedings of the 1990 Workshop on Computational Learning Theory*, Morgan Kaufmann Publishers, 1990.

[48] D. Haussler.
Quantifying inductive bias: AI learning algorithms and Valiant's model.
Artificial Intelligence, 36(2), 1988, pp. 177-221.

[49] D. Haussler.
Space-bounded learning.
University of California at Santa Cruz Information Sciences Department, technical report number UCSC-CRL-88-2, 1988.

[50] D. Haussler.
Generalizing the PAC model: sample size bounds from metric dimension-based uniform convergence results.
Proceedings of the 30th I.E.E.E. Symposium on the Foundations of Computer Science, 1989, pp. 40-45.

[51] D. Haussler, M. Kearns, N. Littlestone, M. Warmuth.
Equivalence of models for polynomial learnability.

Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 42-55, and University of California at Santa Cruz Information Sciences Department, technical report number UCSC-CRL-88-06, 1988.

[52] D. Haussler, N. Littlestone, M. Warmuth.
Predicting 0,1-functions on randomly drawn points.
Proceedings of the 29th I.E.E.E. Symposium on the Foundations of Computer Science, 1988, pp. 100-109.

[53] D. Haussler, L. Pitt, editors.
Proceedings of the 1988 Workshop on Computational Learning Theory.
Morgan Kaufmann Publishers, 1988.

[54] D. Haussler, E. Welzl.
Epsilon-nets and simplex range queries.
Discrete Computational Geometry, 2, 1987, pp. 127-151.

[55] D. Helmbold, R. Sloan, M. Warmuth.
Learning nested differences of intersection-closed concept classes.
Proceedings of the 1989 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1989, pp. 41-56.

[56] D. Johnson.
Approximation algorithms for combinatorial problems.
Journal of Computer and Systems Sciences, 9, 1974, pp. 256-276.

[57] S. Judd.
Learning in neural networks.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 2-8.

[58] N. Karmarkar.
A new polynomial-time algorithm for linear programming.
Combinatorica, 4, 1984, pp. 373-395.

[59] M. Kearns, M. Li.
Learning in the presence of malicious errors.
Proceedings of the 20th A.C.M. Symposium on the Theory of Computing, 1988, pp. 267-280.

[60] M. Kearns, M. Li, L. Pitt, L.G. Valiant.
On the learnability of Boolean formulae.
Proceedings of the 19th A.C.M. Symposium on the Theory of Computing, 1987, pp. 285-295.

[61] M. Kearns, M. Li, L. Pitt, L.G. Valiant.
Recent results on Boolean concept learning.
Proceedings of the 4th International Workshop on Machine Learning, Morgan Kaufmann Publishers, 1987, pp. 337-352.

[62] M. Kearns, L. Pitt.
A polynomial-time algorithm for learning k-variable pattern languages from examples.
Proceedings of the 1989 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1989, pp. 57-71.

[63] M. Kearns, R. Schapire.
Efficient distribution-free learning of probabilistic concepts.
To appear, *Proceedings of the 31st I.E.E.E. Symposium on the Foundations of Computer Science*, 1990.

[64] M. Kearns, L.G. Valiant.
Cryptographic limitations on learning Boolean formulae and finite automata.
Proceedings of the 21st A.C.M. Symposium on the Theory of Computing, 1989, pp. 433-444.

[65] L.G. Khachiyan.
A polynomial algorithm for linear programming.
Doklady Akademiia Nauk SSSR, 244:S, 1979, pp. 191-194.

[66] E. Kranakis.
Primality and cryptography.
John Wiley and Sons, 1986.

[67] P. Laird.
Learning from good and bad data.
Kluwer Academic Publishers, 1988.

[68] L. Levin.
One-way functions and psuedorandom generators.

Proceedings of the 17th A.C.M. Symposium on the Theory of Computing, 1985, pp. 363-365.

[69] M. Li, U. Vazirani.
On the learnability of finite automata.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 359-370.

[70] M. Li, P. Vitanyi.
A theory of learning simple concepts under simple distributions and average case complexity for the universal distribution.
Proceedings of the 30th I.E.E.E. Symposium on the Foundations of Computer Science, 1989, pp. 34-39.

[71] N. Linial, Y. Mansour, N. Nisan.
Constant depth circuits, Fourier transform and learnability.
Proceedings of the 30th I.E.E.E. Symposium on the Foundations of Computer Science, 1989, pp. 574-579.

[72] N. Linial, Y. Mansour, R.L. Rivest.
Results on learnability and the Vapnik-Chervonenkis dimension.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 56-68, and *Proceedings of the 29th I.E.E.E. Symposium on the Foundations of Computer Science*, 1988, pp. 120-129.

[73] N. Littlestone.
Learning quickly when irrelevant attributes abound: a new linear threshold algorithm.
Machine Learning, 2(4), 1988, pp. 245-318, and *Proceedings of the 28th I.E.E.E. Symposium on the Foundations of Computer Science*, 1987, pp. 68-77.

[74] N. Littlestone, M.K. Warmuth.
The weighted majority algorithm.
Proceedings of the 30th I.E.E.E. Symposium on the Foundations of Computer Science, 1989, pp. 256-261.

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

[76] B.K. Natarajan.
On learning Boolean functions.
Proceedings of the 19th A.C.M. Symposium on the Theory of Computing, 1987, pp. 296-304.

[77] R. Nigmatullin.
The fastest descent method for covering problems.
Proceedings of a Symposium on Questions of Precision and Efficiency of Computer Algorithms, Kiev, 1969 (in Russian).

[78] L. Pitt, L.G. Valiant.
Computational limitations on learning from examples.
Journal of the A.C.M., 35(4), 1988, pp. 965-984.

[79] L. Pitt, M.K. Warmuth.
Reductions among prediction problems: on the difficulty of predicting automata.
Proceedings of the 3rd I.E.E.E. Conference on Structure in Complexity Theory, 1988, pp. 60-69.

[80] L. Pitt, M.K. Warmuth.
The minimum consistent DFA problem cannot be approximated within any polynomial.
Proceedings of the 21st A.C.M. Symposium on the Theory of Computing, 1989, pp. 421-432.

[81] D. Pollard.
Convergence of stochastic processes.
Springer Verlag, 1984.

[82] M.O. Rabin.
Digital signatures and public key functions as intractable as factoring.
M.I.T. Laboratory for Computer Science, technical report number TM-212, 1979.

[83] J. Reif.
On threshold circuits and polynomial computations.
Proceedings of the 2nd Structure in Complexity Theory Conference, 1987, pp. 118-125.

[84] R. Rivest.
Learning decision lists.
Machine Learning, 2(3), 1987, pp. 229-246.

[85] R. Rivest, D. Haussler, M. K. Warmuth, editors.
Proceedings of the 1989 Workshop on Computational Learning Theory.
Morgan Kaufmann Publishers, 1989.

[86] R.L. Rivest, R. Schapire.
Diversity-based inference of finite automata.
Proceedings of the 28th I.E.E.E. Symposium on the Foundations of Computer Science, 1987, pp. 78-88.

[87] R.L. Rivest, R. Schapire.
Inference of finite automata using homing sequences.
Proceedings of the 21st A.C.M. Symposium on the Theory of Computing, 1989, pp. 411-420.

[88] R. Rivest, A. Shamir, L. Adleman.
A method for obtaining digital signatures and public key cryptosystems.
Communications of the A.C.M., 21(2), 1978, pp. 120-126.

[89] R.L. Rivest, R. Sloan.
Learning complicated concepts reliably and usefully.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 69-79.

[90] R. Schapire.
On the strength of weak learnability.
Proceedings of the 30th I.E.E.E. Symposium on the Foundations of Computer Science, 1989, pp. 28-33.

[91] G. Shackelford, D. Volper.
Learning k-DNF with noise in the attributes.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 97-105.

[92] R. Sloan.
Types of noise in data for concept learning.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 91-96.

[93] L.G. Valiant.
A theory of the learnable.
Communications of the A.C.M., 27(11), 1984, pp. 1134-1142.

[94] L.G. Valiant.
Learning disjunctions of conjunctions.
Proceedings of the 9th International Joint Conference on Artificial Intelligence, 1985, pp. 560-566.

[95] L.G. Valiant.
Functionality in neural nets.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 28-39.

[96] V.N. Vapnik.
Estimation of dependences based on empirical data.
Springer Verlag, 1982.

[97] V.N. Vapnik, A.Ya. Chervonenkis.
On the uniform convergence of relative frequencies of events to their probabilities.
Theory of Probability and its Applications, 16(2), 1971, pp. 264-280.

[98] K. Verbeurgt.
Learning DNF under the uniform distribution in quasi-polynomial time.
To appear, *Proceedings of the 1990 Workshop on Computational Learning Theory*, Morgan Kaufmann Publishers, 1990.

[99] J.S. Vitter, J. Lin.
Learning in parallel.
Proceedings of the 1988 Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, 1988, pp. 106-124.

[100] R.S. Wencour, R.M. Dudley.
Some special Vapnik-Chervonenkis classes.
Discrete Mathematics, 33, 1981, pp. 313-318.

[101] A. Wigderson.
A new approximate graph coloring algorithm.
Proceedings of the 14th A.C.M. Symposium on the Theory of Computing, 1982, pp. 325-329.

[102] A.C. Yao.
Theory and application of trapdoor functions.
Proceedings of the 23rd I.E.E.E. Symposium on the Foundations of Computer Science, 1982, pp. 80-91.

Index