Efficient Algorithms in Computational Learning Theory

A thesis presented
by

Rocco Anthony Servedio

to

The Division of Engineering and Applied Sciences
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Computer Science

Harvard University
Cambridge, Massachusetts

May 2001

Abstract

Title: Efficient Algorithms in Computational Learning Theory
Author: Rocco Anthony Servedio
Advisor: Leslie Valiant

A major open problem in the theory of machine learning is to develop an efficient algorithm for learning Boolean formulas in Disjunctive Normal Form (DNF). This thesis reports progress towards such an algorithm. We derive a new algebraic characterization of DNF formulas and use this characterization to obtain a DNF learning algorithm which is substantially faster than the best previous approaches. The running time of our algorithm is exponential in the cube root of the number of variables in the DNF formula, as opposed to previous bounds which were exponential in the square root of the number of variables.

The two most important resources for a learning algorithm are computation time and input data; an ideal learning algorithm would simultaneously optimize its use of both resources. Using techniques from cryptography, we show that even very simple learning problems can exhibit strong inherent tradeoffs between these two resources. We describe a class of decision lists and prove that these structures can be learned in polynomial time from a large data set but cannot be learned in polynomial time using a minimal (constant size) data set. These results establish that for some natural learning problems "ideal" learning algorithms simultaneously optimizing both resources cannot exist.

Perceptron and Winnow are two classical algorithms for learning a linear separator which have some remarkable properties. We demonstrate a close connection between boosting, a learning technique which has gained wide use in recent years, and these algorithms for learning linear classifiers. Using boosting we construct a new family of

learning algorithms for linear classifiers which closely match the sample complexity and noise tolerance of algorithms such as Perceptron and Winnow. We thus help unify the seemingly disparate topics of boosting and these classical learning algorithms.

We also present new algorithms which give quantitative performance improvements for a range of well-studied problems in learning theory. We present a new boosting algorithm which is guaranteed to use intermediate probability distributions which are optimally smooth. We use this smooth boosting algorithm to give the fastest known algorithm for learning DNF under the uniform distribution using membership queries, and also to improve on known hard-core set constructions in complexity theory. We also describe a fast, noise tolerant algorithm for learning linear classifiers under symmetric distributions. Finally, we give the fastest known algorithm for learning monotone DNF under the uniform distribution without membership queries.

Acknowledgements

"My friend," said I, "what a charming morning! How sweet the country looks! Pray, did you hear that extraordinary cock-crow this morning? Take a glass of my stout!"

"*Yours*? First pay your debts before you offer folks *your* stout!"

– Herman Melville, "Cock-A-Doodle-Doo! OR
The Crowing of the Noble Cock Beneventano" [1]

I am very grateful to Les Valiant, my advisor, whose course on computational learning theory inspired me to do research in this field. Les has always been generous with his time, advice, encouragement and support. His suggestions and insights have been invaluable to me in my work, and his research has been an inspiration. I thank Les for making my graduate experience so rewarding and enjoyable.

I thank my wife Jenny for all of her love, support, and patience over the years, without which this thesis never would have been written. I thank my son Nicholas, whose recent arrival made this past year infinitely more interesting and enjoyable. I thank my mother, Margaret Servedio, and my sister, Maria Servedio, for their love and support over the years.

Special thanks go to Adam Klivans, who collaborated with me on the research described in Chapters 3 and 8. It has been a pleasure to work with Adam and to know him as a friend.

Many colleagues and friends have helped make my time in graduate school productive, memorable and enjoyable. There are too many to name, but among them I would like to single out Adam Deaton (for the lunches), Amos Beimel (for the coffee), Kostas Magoutis (for the volleyball) and Steven Gortler (for the baseball). I also thank Carol Harlow, Ronda Scott, Eleni Drinea, Lillian Lee, Michael Bender, Roni Khardon, Dan Roth, Wheeler Ruml, Luke Hunsberger, Rebecca Hwa, Yan Zong Ding, John Dunagan, Venkatesan Guruswami, Jeffrey Jackson, Richard Beigel, Avrim Blum, Santosh Vempala, and many others. Thanks to Michael Mitzenmacher, Michael Rabin, and Salil Vadhan for serving on my thesis committee.

This thesis is dedicated to the memory of my father, Frank Servedio.

This research was supported by a National Science Foundation Graduate Research Fellowship, by ONR grant N00014-96-1-0550, by NSF grant CCR-95-04436, and by NSF grant CCR-98-77049.

[1] "COCK-A-DOODLE-DOO!–OO!–OO!–OO!–OO!"

Bibliographic Note

Most of the research presented in this thesis has appeared elsewhere in some form.

Chapter 3 is based on the paper "Learning DNF in Time $2^{\tilde{O}(n^{1/3})}$" which is joint work with Adam Klivans and appeared in the Proceedings of the 2001 Symposium on Theory of Computing (Klivans and Servedio, 2001).

Chapter 4 is based on the paper "Computational Sample Complexity and Attribute-Efficient Learning" which appeared in the *Journal of Computer and System Sciences* (Servedio, 2000a); a preliminary version of this paper appeared in the Proceedings of the 1999 Symposium on Theory of Computing (Servedio, 1999a).

Chapters 5 and 6 are based on the paper "On PAC Learning using Winnow, Perceptron, and a Perceptron-like Algorithm" which appeared in the Proceedings of the 1999 Conference on Computational Learning Theory (Servedio, 1999b).

Chapter 7 is based on the paper "PAC Analogues of Perceptron and Winnow via Boosting the Margin" which will appear in *Machine Learning* (Servedio, 2001b); a preliminary version of this paper appeared in the Proceedings of the 2000 Conference on Computational Learning Theory (Servedio, 2000b).

Chapter 8 is based on the paper "Boosting and Hard-Core Sets" which is joint work with Adam Klivans and appeared in the 1999 Symposium on Foundations of Computer Science (Klivans and Servedio, 1999).

Chapter 9 is based on the paper "Smooth Boosting and Learning with Malicious Noise" which appeared in the Proceedings of the 2001 Conference on Computational Learning Theory (Servedio, 2001c).

Chapter 10 is based on the paper "On Learning Monotone DNF under Product Distributions" which appeared in the Proceedings of the 2001 Conference on Computational Learning Theory (Servedio, 2001a).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

- A voice-activated domestic robot performs a wide range of household chores.
- A real-time system fluently translates between English and Italian.
- A powerful computer interprets radiographic images and makes accurate diagnoses.

While automated systems such as these would be indisputably useful, the task of writing explicit computer programs to solve such "messy" real-world problems seems to be prohibitively difficult. For these and many other useful real-world tasks, any successful system must be able to handle a vast number of potential inputs (English sentences, radiographic images, etc.); even if the system's designer has extensive knowledge of the specific problem domain of interest, it may be nearly impossible to explicitly construct a program which exhibits the desired behavior. The field of *machine learning* attempts to circumvent this difficulty by building automated systems which can learn from data, thus obviating the need to program an explicit algorithm. But what principles should guide the design of learning-based systems? And what are the inherent capabilities and limitations of computers that learn?

Research in the theory of machine learning attempts to provide a rigorous mathematical foundation for answering such questions. The broad aim of most work in theoretical machine learning is to define and analyze different learning frameworks, learning problems, and learning algorithms. Towards this end a range of different approaches have been taken by researchers in fields such as pattern recognition, estimation theory, and theoretical computer science. In this thesis we consider learning from the perspective of *computational learning theory,* the branch of theoretical computer science which studies the abilities and limitations of computationally efficient learning algorithms.

Several compelling arguments support the strong emphasis which computational learning theory places on algorithmic efficiency issues. Since computationally efficient learning algorithms are indeed the only algorithms which can be used in practice for large data sets, it is beyond dispute that algorithmic efficiency is a *sine qua non* in real-world large-scale learning situations. In addition to this pragmatic motivation, a strong case can be made that computational efficiency should be taken into account in order to achieve an adequately rich and nuanced mathematical theory of learning. It is well known that in many learning models, if the computational complexity of learning is ignored then learnability can be completely characterized by simple combinatorial parameters such as the Vapnik-Chervonenkis dimension; such a coarse categorization sheds little light on the actual similarities and differences between learning problems. Moreover, from this perspective of information-theoretic (as opposed to computational) learnability, many learning problems can be "solved" by exhaustive search. Of course such solutions gives little if any insight into the structure of the learning problem, and are completely impractical to boot. On the other hand, by taking the computational complexity of learning into account a broad and fascinating spectrum of different behaviors are found to exist among learning problems. In this thesis, for example, we consider learning problems which are provably easy (solvable by polynomial-time algorithms); learning problems which are certifiably hard (at least as difficult as cryptographic problems widely believed to require exponential time); and learning problems whose complexity is as yet tantalizingly unresolved.

In computational learning theory the object being learned is typically modeled as a Boolean function over some domain, and the task of the learning algorithm is to infer this unknown target function on the basis of labeled examples. The intuition is that the unknown function represents a *concept* which the learning algorithm is trying to master; e.g. in a situation where a robot is learning to do inventory control in a library, the concept to be learned might be "book written in English." In this setting the unknown Boolean function would label each object in the robot's universe as either a positive or negative instance of an English-language book, and the robot's task would be to infer an accurate classification rule after seeing some sample objects and their corresponding binary labels. The standard paradigm in learning theory is that the unknown "target concept" is *a priori* known to belong to some fixed class of possible functions, usually referred to as the *concept class* which is being learned. In our library application the concept class might consist of concepts such as "book written in language X," "book published in country Y," "book about topic Z" and so on. In computational learning theory research the most frequently studied concept classes are syntactic classes of Boolean formulae; two such classes which have been intensively studied (and are among the main topics of this thesis) are the class of Disjunctive Normal Form formulae and the class of linear threshold functions.

It is intuitively clear that simple concepts should be easier to learn than complex ones; for instance, the class of simple library concepts described above should be easier to learn than a class which includes concepts such as "gripping suspense thriller" and "poststructuralist critique of dialectical materialism." A broad goal of computational learning theory research, and of this thesis, is to discover how the complexity of learning scales with the complexity of the concepts which are being learned.

1.2 Summary of Thesis Contributions

While the research in this thesis spans several different topics, the common thread which unites our results is a central emphasis on *computationally efficient learning.* We consider the following results, described in more detail in Section 1.3, to be the primary contributions of the thesis:

- We give the fastest known algorithm for learning an arbitrary Disjunctive Normal Form (DNF) formula in the Probably Approximately Correct (PAC) model of learning with respect to an arbitrary probability distribution on examples. The problem of PAC learning DNF is widely viewed as one of the most important problems in computational learning theory.

- We demonstrate that even for very simple learning problems, a strong tradeoff can exist between the running time and the number of examples required of any successful learning algorithm. As an aspect of this tradeoff we give the first example of a simple learning problem for which *attribute-efficient* learning (a type of learning from few examples) is computationally hard.

- We construct a new family of efficient noise-tolerant linear threshold learning algorithms. These algorithms, which are based on hypothesis boosting, closely match the performance of the Perceptron and Winnow algorithms, two of the best-known algorithms in machine learning.

- We establish an equivalence between boosting algorithms in learning theory and hard-core set constructions in complexity theory. This equivalence yields improved complexity theoretic hard-core set constructions as well as more efficient algorithms in learning theory.

- We give a new boosting algorithm which generates optimally smooth distributions and show that this boosting algorithm can be used to construct PAC learning algorithms which tolerate high levels of noisy data.

The following results are secondary contributions of the thesis:

- We give an efficient algorithm which learns $2^{O(\sqrt{\log n})}$-term monotone DNF under the uniform distribution. This is the first polynomial-time algorithm which learns monotone DNF with more than a polylogarithmic number of terms in any model of learning from random examples only.

- We give the fastest known version of the celebrated Harmonic Sieve algorithm for learning DNF formulae under the uniform distribution using membership queries.

- We give new analyses of the Perceptron and Winnow algorithms in the PAC learning model, including the first proof that Winnow is not an efficient PAC learning algorithm for the class of linear threshold functions.

- We give a fast, simple algorithm for learning origin-centered linear threshold functions under the uniform distribution which is resistant to several types of noisy data.

With the exception of Chapter 2, which contains preliminary background material, each chapter is self contained and can be read separately from the rest. Below we give an outline of the thesis chapters.

1.3 Outline of the Thesis

Models and Background

In this chapter we define the PAC learning model and the online mistake bound model, the two main learning models which we use, and present some mathematical background material.

Faster Learning of DNF via Polynomial Threshold Functions

The problem of efficiently learning arbitrary DNF formulae is one of the outstanding open questions in computational learning theory. In this chapter we present the fastest known algorithm for this well-studied problem.

Our algorithm is based on a new conversion from DNF to *polynomial threshold functions*. A polynomial threshold function of degree d over n variables is the Boolean function obtained by thresholding a real-valued n-variable polynomial of degree d. In their 1968 book *Perceptrons*, Minsky and Papert proved a lower bound of $\Omega(n^{1/3})$ on the degree of polynomial threshold functions which represent certain polynomial-size DNF. We give an upper bound which matches this lower bound to within a logarithmic factor by showing that any s-term DNF formula on n variables can be expressed as a polynomial threshold function of degree $O(n^{1/3} \log s)$. This upper bound enables us to give an algorithm which learns s-term DNF formulae over n variables in time $2^{O(n^{1/3} \log n \log s)}$, thus substantially improving the $2^{\tilde{O}(\sqrt{n})}$ time bounds of the best previous algorithms. We also give improved upper bounds on the degree of polynomial threshold functions for read-once DNF and for read-once constant depth formulae. In each case our upper bound leads to the fastest known algorithm for the corresponding learning problem.

Computational Sample Complexity and Attribute Efficient Learning

Two resources required by all learning algorithms are data and computation time. In this chapter we show that even for simple learning problems a strong inherent tradeoff can exist between these two resources.

In Section 4.3 we use error-correcting codes and length-preserving one-way permutations to construct a class of very simple Boolean functions (1-decision lists). We prove that while a computationally unbounded learner can learn any function in this class from $O(1)$ examples, under a general cryptographic hardness assumption any polynomial-time learner must use almost $\Omega(n)$ examples.

Using a different cryptographic construction based on pseudorandom generators, in Section 4.4 we present a more general class of k-decision lists which exhibits a similar but stronger gap ($O(1)$ versus $\Omega(n^k)$) in the number of examples required for computationally unbounded versus polynomial-time learning. We show that this construction comes within a logarithmic factor of the largest possible gap for classes of k-decision lists.

Attribute efficient learning is a type of learning from few examples which has been widely studied. In Section 4.5 we apply our techniques to construct a concept class of decision lists which can be learned attribute-efficiently and can be learned in polynomial time but cannot be learned attribute-efficiently in polynomial time. This is the first demonstration that attribute-efficient learning can be computationally hard.

PAC Learning with Perceptron and Winnow

Chapter 5 contains detailed analyses of the PAC learning abilities of the Perceptron and Winnow algorithms. While these algorithms have been studied in the online mistake-bound model by many researchers, little was previously known about their performance in the PAC learning model.

In Section 5.3 we give an exponential lower bound on the running time of the Winnow algorithm when used as a PAC learning algorithm for the class of positive linear threshold functions over Boolean examples. This answers an open question of Schmitt and is the first negative result for Winnow in the PAC learning model.

In Section 5.4 we prove a strong negative result for the Perceptron algorithm by showing that Perceptron cannot efficiently PAC learn linear threshold functions even under the uniform distribution on Boolean examples. Our proof simplifies and strengthens an earlier negative result due to Schmitt on the PAC-model performance of the Perceptron algorithm.

On the positive side, we show in Section 5.5 that Perceptron is a polynomial-time PAC learning algorithm for the class of nested functions under the uniform distribution on Boolean examples. This class is known to be hard for Perceptron in the general PAC model where the distribution can be arbitrary.

Learning Origin-Centered Halfspaces under Uniform Distributions

In this chapter we consider a natural learning problem which has been studied by several researchers: learning an unknown origin-centered halfspace (linear threshold function) from random examples drawn uniformly from the origin-centered unit sphere in $\Re^n$. We describe a new algorithm based on a geometric averaging technique which can learn efficiently even in the presence of monotonic noise (a generalization of the well-studied classification noise model). We show that our new algorithm is both simpler and faster than previous algorithms which have been proposed for this problem.

PAC Analogues of Perceptron and Winnow via Boosting the Margin

In this chapter we describe and analyze a new family of boosting-based PAC model algorithms for learning linear threshold functions. While our new algorithms are conceptually and algorithmically quite different from Perceptron and Winnow, we prove performance bounds for the new algorithms which are remarkably similar to those of Perceptron and Winnow, thus suggesting that these well-studied online algorithms in some sense correspond to instances of boosting.

In Section 7.3 we describe a parameterized version of the geometric averaging algorithm from Chapter 6 and show that this algorithm produces hypotheses whose error rate is bounded away from 1/2. In Section 7.4 we use a boosting algorithm to efficiently convert this geometric weak learning algorithm into a strong learning algorithm which can generate arbitrarily accurate hypotheses.

By using techniques from the theory of large margin classification to carefully bound the number of examples required by the new algorithms, we show that the new algorithms can be viewed as natural PAC analogues of the online p-norm algorithms which have recently been studied by Grove, Littlestone, and Schuurmans (Grove *et al.*, 1997) and Gentile and Littlestone (Gentile and Littlestone, 1999). As special cases of the new algorithms, by taking $p = 2$ and $p = \infty$ we obtain natural boosting-based PAC analogues of Perceptron and Winnow respectively. The $p = \infty$ case of our algorithm can also be viewed as a generalization (with an improved sample complexity bound) of Jackson and Craven's PAC-model boosting-based algorithm for learning "sparse perceptrons."

Boosting and Hard-Core Set Construction

In this chapter we show that a close connection exists between hard-core set construction, a type of hardness amplification from computational complexity, and boosting algorithms in learning theory. We exploit this connection both to improve previous learning theory results which are based on boosting and to obtain improved hard-core set constructions in complexity theory.

In Section 8.4 we show that the hard-core set construction of Impagliazzo (Impagliazzo, 1995), which establishes the existence of distributions under which Boolean functions are highly inapproximable, may be viewed as a boosting algorithm. More generally, we prove that any boosting algorithm yields a corresponding hard-core set construction. Using this connection we apply known boosting algorithms to obtain improved bounds for hard-core set construction which match known lower bounds from boosting and thus are optimal within this class of techniques.

The boosting algorithm which corresponds to Impagliazzo's hard-core set construction has the property that it generates only "smooth" distributions which do not assign too much weight to any single example. In Section 8.5 we show that smooth boosting algorithms can be used to give a new version of Jackson's celebrated Harmonic Sieve algorithm for learning DNF formulae under the uniform distribution using membership queries. Our new version of the Sieve has a significant asymptotic improvement in running time. By combining our techniques with recent improvements to other aspects of the Sieve, we obtain an $\tilde{O}(ns^4/\epsilon^2)$-time algorithm for learning s-term n-variable DNF under uniform with membership queries.

Smooth Boosting and Learning with Malicious Noise

This chapter considers the problem of learning linear threshold functions in the presence of *malicious noise*, a more demanding noise model than either the classification noise or monotonic noise models which we considered in Chapter 6. By combining ideas from the previous two chapters, we derive a new family of malicious noise tolerant algorithms for learning linear threshold functions.

In Section 9.2 we describe a new boosting algorithm which, like Impagliazzo's algorithm from Chapter 8, generates only smooth distributions. Our new algorithm is significantly faster than Impagliazzo's algorithm and has several other desirable properties such as the ability to generate large-margin hypotheses and to use real-valued weak hypotheses.

In Section 9.3 we show that our new boosting algorithm can be used in conjunction with the parameterized averaging algorithm from Section 7.3 to obtain malicious noise tolerant versions of the PAC model linear threshold learning algorithms described in Section 7.4. The bounds on malicious noise tolerance and sample complexity of these new PAC algorithms closely correspond to known bounds for the online p-norm algorithms which include the Perceptron and Winnow algorithms as special cases.

Our analysis reveals an interesting connection between the smoothness of boosting algorithms and the level of malicious noise tolerance which can be achieved in the PAC model. In Section 9.4 we use this connection to prove that the distributions generated by our new boosting algorithm are optimally smooth.

Learning Monotone DNF under Product Distributions

Finally, in Chapter 10 we return to the problem of learning DNF. Through a careful analysis of the Fourier spectrum of monotone DNF, we give a polynomial time uniform distribution PAC learning algorithm for the class of monotone $2^{O(\sqrt{\log n})}$-term DNF formulae on n variables. This is an exponential improvement over previous polynomial-time algorithms in this model, which could learn monotone $o(\log^2 n)$-term DNF, and is the first efficient algorithm for monotone $(\log n)^{\omega(1)}$-term DNF in any model of learning from random examples only. We show that our result can be extended to learning various classes of small constant-depth circuits which compute monotone functions on few variables and to learning DNF (or circuits as described above) under any constant-bounded product distribution.

Future Directions

We close with a brief discussion of future research directions.

Chapter 2

Models and Background

In each of the learning models which we consider the learning algorithm has some type of access to the input-output behavior of an unknown Boolean function, and the learning algorithm's goal is to construct a representation (exact or approximate) of this unknown function. The various models which we describe below differ in the details of how they instantiate this general paradigm.

2.1 Concepts and Representations

Let X be a set which is the domain for the unknown function to be learned; we refer to X as the *instance domain*. Typically X will be either the Boolean cube $\{0,1\}^n$ or n-dimensional Euclidean space $\Re^n$. A *concept* is a Boolean function $c : X \rightarrow \{-1, 1\}$. Equivalently we will sometimes view a concept c as a subset of X, namely $\{x \in X : c(x) = 1\}$. A *concept class* is a set $\mathcal{C}$ of concepts. In each of the learning scenarios we consider there is an unknown concept $c \in \mathcal{C}$ called the *target concept* which the algorithm is trying to learn from labeled examples. The intuition is that the class $\mathcal{C}$ is known to the learning algorithm but the identity of $c \in \mathcal{C}$ is not.

Since there are many different ways to represent any given Boolean function, we consider *representation classes* as well as concept classes. As an example to help clarify the distinction between a concept class and a representation class, we now define the class of *linear threshold functions* which plays an important role in this thesis. For $X = \{0,1\}^n$ or $X = \Re^n$ a concept $c : X \rightarrow \{-1, 1\}$ is a linear threshold function if there exist $w \in \Re^n$, $\theta \in \Re$ such that $c(x) = 1$ if and only if $w \cdot x \geq \theta$.

We say that the pair $(w, \theta) \in \Re^n \times \Re$ *represents* the linear threshold function c. Thus the concept class of linear threshold functions over X is a set of Boolean functions as described above, and the representation class of linear threshold functions over X is the set $\Re^n \times \Re$.

Any learning algorithm which outputs representations belonging to a class $\mathcal{H}$ must require at least enough time to write down an appropriate representation in $\mathcal{H}$. For $h \in \mathcal{H}$ we define size(h) to be the description length of h under some fixed reasonable encoding scheme. If $\mathcal{H}$ is a representation class which is associated with concept class $\mathcal{C}$, then for $c \in \mathcal{C}$ we define size(c) to be the minimum of size(h) across all representations $h \in \mathcal{H}$ of the concept c.

The two concept classes which we study most frequently in this thesis are the class of linear threshold functions described above and the class of *disjunctive normal form* formulae or DNF. A DNF formula is a depth 2 unbounded fanin circuit which computes an OR of ANDS of Boolean literals. We refer to the AND gates of a DNF as *terms*. Strictly speaking the class of DNF formulae is a representation class rather than a concept class; however we will sometimes abuse notation and refer to DNF as a concept class. Note that since any Boolean function has some representation as a DNF formula (possibly requiring exponentially many terms) the concept class of DNF is the class of all Boolean functions. In the context of DNF the value of size(c) is standardly defined to be the minimum number of terms in any DNF formula which represents the concept c. We will sometimes refer to the class of *s-term DNF*; this is the set of all Boolean functions which can be expressed as DNF formulae which have at most s terms.

2.2 PAC Learning

The Probably Approximately Correct (PAC) learning model was introduced by Valiant in (Valiant, 1984) and has since been widely studied. In the PAC model the learning algorithm is given access to an *example oracle* $EX(c, \mathcal{D})$ as its source of labeled examples. Here $c \in \mathcal{C}$ is the target concept and $\mathcal{D}$ is an unknown probability distribution over the instance domain X. The oracle $EX(c, \mathcal{D})$ takes no inputs; when invoked it returns a pair $\langle x, c(x) \rangle$ where $x \in X$ is chosen according to $\mathcal{D}$. The value $c(x)$ is referred to as the *label* of instance x, and the pair $\langle x, c(x) \rangle$ as a *labeled example*. The collection of labeled examples which a learning algorithm receives from $EX(c, \mathcal{D})$ is often referred to as a *sample*.

For $h \in \mathcal{H}$ we say that the *error rate* of h is the probability (taken with respect to $\mathcal{D}$) that the function represented by h disagrees with c, i.e. $\mathrm{error}(h) = \Pr_{x \in \mathcal{D}}[h(x) \neq c(x)]$. Note that the error rate of h is measured with respect to the same distribution $\mathcal{D}$ from which labeled examples are drawn; thus if a portion of the instance space X receives little or no weight under $\mathcal{D}$, a hypothesis h can err on this portion of X and still achieve low error overall.

A PAC learning algorithm works by drawing a sample from the example oracle $EX(c, \mathcal{D})$, performing some computation on this sample, and then outputing a hypothesis h which belongs to some representation class $\mathcal{H}$. The goal of a learning algorithm is to output a hypothesis h which, with probability $1 - \delta$, has $\mathrm{error}(h) \leq \epsilon$. The value δ is referred to as the *confidence* parameter and ϵ as the *accuracy* parameter.

As motivation for this "probably approximately correct" success criterion, we observe that there are two sources of difficulty which can confound a PAC learning algorithm. Since the source $EX(c, \mathcal{D})$ of labeled examples is probabilistic, it is possibile that a learner's sample will be highly unrepresentative of the target concept; if this event (the probability of which is controlled by the δ parameter) should occur then the resulting hypothesis might well have large error. On the other hand, even if the learning algorithm draws a "typical" sample it may not be possible to learn the target concept exactly. Thus we require only that the learning algorithm's hypothesis be approximately (as measured by ϵ) correct.

We have the following formal definition of PAC learning which is adapted from (Kearns and Vazirani, 1994):

Definition 2.1 *Let $\mathcal{C}$ be a concept class over X and let $\mathcal{H}$ be a corresponding representation class. An algorithm A is said to be a* PAC learning algorithm for $\mathcal{C}$ using $\mathcal{H}$ *if the following holds: for all $0 < \epsilon, \delta < 1$, for all $c \in \mathcal{C}$, for all distributions $\mathcal{D}$ over X, if A is given as input an accuracy parameter ϵ and a confidence parameter δ and A has access to the example oracle $EX(c, \mathcal{D})$, then with probability at least $1 - \delta$ algorithm A outputs a hypothesis $h \in \mathcal{H}$ such that $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] \leq \epsilon$.*

If A runs in time $p(n, \mathrm{size}(c), 1/\epsilon, 1/\delta)$ for some fixed polynomial p, we say that A is a *polynomial time PAC learning algorithm.* A concept class $\mathcal{C}$ is said to be *efficiently PAC learnable* if there exists a polynomial time PAC learning algorithm for $\mathcal{C}$.

In addition to the time complexity of learning algorithms we will also be concerned with the number of examples they require. Let $q(n, s, 1/\epsilon, 1/\delta)$ denote the maximum number of calls to $EX(c, \mathcal{D})$ which A ever makes for any target concept $c \in \mathcal{C}$ of size s and any distribution $\mathcal{D}$. The value $q(n, s, 1/\epsilon, 1/\delta)$ is known as the *sample complexity* of A. The sample complexity of a concept class $\mathcal{C}$ is the minimum sample complexity across all learning algorithms for $\mathcal{C}$.

2.3 Variations and Extensions of the PAC Model

Algorithms which satisfy Definition 2.1 are sometimes referred to as *strong learning algorithms* since they can achieve an arbitrarily low error rate ϵ with high probability. A more relaxed notion of learning known as *weak learning* was introduced by Kearns and Valiant (Kearns and Valiant, 1994). A weak learning algorithm cannot necessarily achieve an ϵ-accurate hypothesis for arbitrarily small ϵ, but is able to generate a hypothesis whose error rate is noticeably less than 1/2. More precisely, A is a weak learning algorithm if there is some polynomial $p(n,\text{size}(c))$ such that for any $c \in \mathcal{C}$ and any distribution $\mathcal{D}$, if A is given $\delta > 0$ and access to $EX(c, \mathcal{D})$, then with probability at least $1 - \delta$ algorithm A generates a hypothesis h such that $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] \leq 1/2 - 1/p(n,\text{size}(c))$.

It is easy to see that any concept class which is strongly PAC learnable is also weakly PAC learnable. In a celebrated result Schapire (Schapire, 1990) showed that the converse holds as well: any concept class which is efficiently weakly PAC learnable is also efficiently strongly PAC learnable. Schapire's proof was highly constructive in that he gave a *boosting* algorithm which can be used to construct a strong learning algorithm from any weak learning algorithm using the weak learning algorithm as a subroutine. Over the past decade boosting has become a major research topic in computational learning theory. We study and construct weak learning algorithms and boosting algorithms in Chapters 7, 8 and 9.

In addition to the standard PAC notions of weak and strong learning described above, we will also consider several well-studied variants of the PAC learning model. In Chapters 5, 6 and 10 we consider a distribution-specific version of the PAC model in which the learning algorithm is only required to succeed for some fixed distribution on X (typically the uniform distribution). In Chapter 8 we study an augmented PAC model in which the learning algorithm can additionally make *membership queries* for the value of the target concept c at designated points. In Chapters 7 and 9 we consider versions of the PAC learning model in which the distribution $\mathcal{D}$ depends on the target concept being learned. We describe each of these variants in more detail in the corresponding chapters.

In both the standard PAC model and the variants mentioned above, it is assumed that the example oracle $EX(c, \mathcal{D})$ always provides data points which are labeled correctly according to c. Several different models of learning from noisy data have been proposed in an attempt to relax this assumption. These models include the *classification noise* model in which each call to $EX(c, \mathcal{D})$ returns a mislabeled example with some fixed probability; the *monotonic noise* model in which "borderline" examples are more likely to be mislabeled; and the *malicious noise* model in which an omniscient adversary can corrupt both the labels of examples and the example points themselves. We define and study these models in Chapters 3, 6 and 9.

2.4 The Online Learning Model

We will occasionally consider a different learning model known as the *online learning model* which was introduced by Angluin (Angluin, 1988) and Littlestone (Littlestone, 1988). In this model learning is viewed as an interactive process between the learning algorithm and the outside world. Throughout the interaction the learning algorithm maintains a hypothesis $h : X \to \{-1, 1\}$ which can be updated as the learning algorithm receives more and more information about the target concept. Learning proceeds in a sequence of stages where each stage is structured as follows: First the learning algorithm is presented with an unlabeled example $x \in X$. The learning algorithm uses its current hypothesis to compute $h(x)$ and outputs the value $h(x)$. Then the learning algorithm is presented with the true value $c(x)$ of the target concept on example x. The learning algorithm may update its current hypothesis on the basis of this new information before the next stage begins.

A learning algorithm in this model is said to *make a mistake* whenever its prediction differs from the true value of the label, i.e. $h(x) \neq c(x)$. The performance of a learning algorithm is measured by the number of mistakes it makes on an example sequence. A learning algorithm A for a concept class $\mathcal{C}$ is said to have mistake bound

M if for any sequence of examples in X and any target concept $c \in \mathcal{C}$, algorithm A makes at most M mistakes. The *mistake bound* of a concept class $\mathcal{C}$ is the minimum mistake bound of any online learning algorithm for $\mathcal{C}$.

In order to facilitate comparisons between online and PAC learning algorithms we will sometimes want to analyze the PAC learning abilities of online learning algorithms. Several generic conversion techniques are known which can be used to convert any online learning algorithm into a PAC learning algorithm (Angluin, 1988; Haussler, 1988; Kearns *et al.*, 1987b; Littlestone, 1989a). In each of these conversions the sample complexity of the resulting PAC algorithm depends on the mistake bound of the online learning algorithm. The conversion procedure due to Littlestone is asymptotically most efficient in terms of the sample complexity of the resulting PAC algorithm and works as follows: The procedure first runs the online algorithm on a sequence of labeled examples obtained from successive calls to the PAC oracle and stores all of the hypotheses which the online algorithm generates on this example sequence. Then the procedure draws a fresh batch of labeled examples from the oracle and use these new examples to estimate the error rate of each of the stored hypotheses. The final hypothesis output by the procedure is the one with the lowest observed error rate. Littlestone proves the following theorem:

Theorem 2.2 (Littlestone, 1989a) *Let A be an online learning algorithm which changes its hypothesis only when it makes a mistake and which has a mistake bound of M for concept class C. Then there is a PAC-model learning algorithm A' for C as described above which has sample complexity*

$$O\left(\frac{1}{\epsilon}\left(\log\frac{1}{\delta}+M\right)\right).$$

2.5 Mathematical Background

We use a range of mathematical tools in this thesis. In Chapter 4 we use tools from cryptography to prove lower bounds on all polynomial time learning algorithms; in Chapters 5, 6, 7 and 9 we use geometric arguments to develop and analyze algorithms for learning linear threshold functions; and in Chapter 10 we use Fourier analysis over the Boolean cube to construct an algorithm for learning DNF. In each of these chapters we introduce the required background material as needed.

One task which we will need to perform on many occasions is to bound the deviation of a sum of random variables from its expected value. The following well known facts show that for a sum of many independent random variables the probability of a large deviation is exponentially small. Note that in the first fact below each random variable takes values in $\{0, 1\}$ and the deviation is measured multiplicatively, while in the second fact each random variable lies in a fixed interval $[a, b]$ and the deviation is measured additively.

Fact 2.3 (Chernoff, 1952) *Let $X_1, \ldots, X_t$ be independent 0/1-valued random variables which satisfy $\Pr[X_i = 1] = p_i$ and $0 < p_i < 1$ for all i. Let $X = \sum_{i=1}^{t} X_i$. For $\mu = \sum_{i=1}^{t} p_i$ and any $0 < \nu < 1$ we have*

$$\Pr\left[X \geq (1+\nu)\mu\right] < \exp(-\mu\nu^2/3)$$

and

$$\Pr\left[X \leq (1-\nu)\mu\right] < \exp(-\mu\nu^2/2).$$

Fact 2.4 (Hoeffding, 1963) *Let $X_1, \ldots, X_t$ be independent random variables such that $a \leq X_i \leq b$ for all i. If $X = \frac{1}{t}\sum_{i=1}^{t} X_i$ then for any $\nu > 0$ we have*

$$\Pr\left[X \geq E\left[X\right] + \nu\right] \leq \exp\left(\frac{-2t\nu^2}{(b-a)^2}\right)$$

and

$$\Pr\left[X \leq E\left[X\right] - \nu\right] \leq \exp\left(\frac{-2t\nu^2}{(b-a)^2}\right).$$

We close this chapter with some notation. For real-valued functions f and g we write $f(\rho) = \tilde{O}(g(\rho))$ if there is a fixed constant c such that $f(\rho) = O(g(\rho)\log^c \rho)$. The notations $\tilde{\Omega}(g(\rho))$ and $\tilde{\Theta}(g(\rho))$ are defined analogously. We write log to denote logarithm base 2 and ln to denote natural logarithm. For $x \in \Re^n$ we write x_i to denote the i-th coordinate of x and we write $\|x\|$ to denote the Euclidean norm $\sqrt{\sum_{i=1}^{n} x_i^2}$. If $x \in \{0, 1\}^n$ then x_i denotes the i-th bit of x. For $z \in \Re$ the value of $\mathrm{sign}(z)$ is 1 if $z \geq 0$ and is -1 if $z < 0$.

Chapter 3

Faster Learning of DNF via Polynomial Threshold Functions

This chapter establishes a new complexity-theoretic characterization of DNF formulae as thresholded polynomials and uses this characterization to obtain a fast learning algorithm. More precisely, our main result is a proof that any s-term DNF over n variables can be computed by a polynomial threshold function of degree $O(n^{1/3} \log s)$. This upper bound matches, up to a logarithmic factor, a longstanding lower bound for DNF which was given by Minsky and Papert in their 1968 book *Perceptrons*. As a direct consequence of this upper bound we obtain a $2^{O(n^{1/3} \log n \log s)}$-time algorithm for learning s-term DNF, which is the fastest known algorithm for this important problem.

3.1 Introduction

3.1.1 Polynomial Threshold Functions

Let f be a Boolean function $f : \{0,1\}^n \to \{-1,1\}$ and let p be a degree d polynomial in n variables with rational coefficients. If the sign of $p(x)$ equals $f(x)$ for every $x \in \{0,1\}^n$, then we say that f is computed by a *polynomial threshold function* of degree d. In their well known 1968 book *Perceptrons*, Minsky and Papert studied some computational aspects of polynomial threshold functions from an Artificial Intelligence perspective (Minsky and Papert, 1968). They proved, among other things, that no polynomial threshold function of degree less than n can compute the parity function on n variables, and that there is a read-once DNF formula which cannot be computed by any polynomial threshold function of degree less than $\Omega(n^{1/3})$. Since then, complexity theorists have used these and related properties of polynomial threshold functions to prove several important results in both circuit and structural complexity (Aspnes *et al.*, 1994; Beigel, 1993; Fu, 1992).

In the computational learning theory community, learning a polynomial threshold function from labeled examples has long been a central problem and continues to be an active area of research. A special focus of attention has been directed toward learning polynomial threshold functions of degree 1, which are known as *linear threshold functions*. The problem of learning a linear threshold function over $\{0, 1\}^n$ can be formulated as a linear programming problem and thus can be solved in poly(n) time in both the PAC model of learning from random examples and in the model of exact learning from equivalence queries (Blumer *et al.*, 1989; Maass and Turan, 1994). Refinements of the basic linear programming approach have led to polynomial-time algorithms for PAC learning linear threshold functions in the presence of classification noise (Blum *et al.*, 1997; Cohen, 1997). Much attention has also recently been given to fast, simple algorithms, most notably the Winnow and Perceptron algorithms, which learn linear threshold functions under restricted conditions (Bylander, 1998b; Freund and Schapire, 1998; Kivinen *et al.*, 1997; Littlestone, 1988; Schmitt, 1998; Servedio, 1999b). We give a detailed analysis of these and related algorithms in Chapters 5, 6, 7 and 9.

3.1.2 Learning DNF

Another intensively studied problem in computational learning theory, which has met with less success, is the problem of learning DNF formulae. DNF are attractive from a learning theory perspective because of their high expressive power (any Boolean function can be represented as a DNF) and because they seem to be a natural form of knowledge representation for humans. Valiant first posed the question of whether DNF are efficiently learnable in his seminal 1984 paper introducing the PAC learning model (Valiant, 1984); more than fifteen years later this question is widely regarded as one of the most important open problems in learning theory. While many partial results have been given for restricted versions of the DNF learning problem (see the introduction to Chapter 10 for an overview of some of these results), the difficulty of the unrestricted DNF learning problem is evidenced by the fact that, prior to the results of this chapter, only two algorithms were known which improve on the naive 2^n time bound (Bshouty, 1996; Tarui and Tsukiji, 1999).

The first subexponential time algorithm for learning DNF is due to Bshouty, who gave an algorithm which learns any s-term DNF over n variables in time $2^{O(\sqrt{n \log s} \log^{3/2} n)}$. At the heart of Bshouty's algorithm is a structural result which shows that that any s-term DNF can be expressed as a "decision list" of order $O(\sqrt{n \log n \log s})$; armed with this result, Bshouty uses a standard algorithm from (Helmbold *et al.*, 1990) for learning decision lists to obtain his DNF learning result.

Tarui and Tsukiji gave a completely different proof of a similar time bound for learning DNF. They adapted the machinery of "approximate inclusion/exclusion" developed by Linial and Nisan (Linial and Nisan, 1990) to show that for any s-term DNF f and any distribution $\mathcal{D}$ over $\{0,1\}^n$, there is a conjunction C of size $O(\sqrt{n} \log s)$ which has $|\Pr_{x \in \mathcal{D}}[C(x) = f(x)] - \frac{1}{2}| = 2^{-O(\sqrt{n} \log n \log s)}$. Using this result together with Freund's "boost-by-majority" algorithm (Freund, 1995), Tarui and Tsukiji obtained an algorithm for learning s-term DNF in time $2^{O(\sqrt{n} \log n \log s)}$.

3.1.3 A New Approach: Learning DNF via Polynomial Threshold Functions

We approach the DNF learning problem by representing a DNF formula as a low-degree polynomial threshold function. As we observe in Section 2, we can use known polynomial-time algorithms for learning linear threshold functions to learn polynomial threshold functions of degree d in time $n^{O(d)}$. Thus, upper bounds on the degree of polynomial threshold functions which compute DNF translate directly into bounds on the running time of a DNF learning algorithm.

Viewing DNF formulae as polynomial threshold functions immediately yields a new interpretation of the DNF learning algorithms of Bshouty and Tarui and Tsukiji. Since any r-decision list is equivalent to a polynomial threshold function of degree r (Ehrenfeucht *et al.*, 1989), in the language of polynomial threshold functions Bshouty's structural result implies that any s-term DNF can be expressed as a polynomial threshold function of degree $O(\sqrt{n \log n \log s})$. In the case of Tarui/Tsukiji, it can be shown as a corollary of their results that any s-term DNF can be expressed as a polynomial threshold function of degree $O(\sqrt{n}\log s)$. Thus, each of these earlier learning algorithms implies an $O(\sqrt{n}\log n)$ upper bound on the degree of a polynomial threshold function for any polynomial-size DNF. A substantial gap still remained, though, between these $O(\sqrt{n}\log n)$ upper bounds and the $\Omega(n^{1/3})$ lower bound due to Minsky and Papert.[1]

3.1.4 Our Results

Our first result is the following theorem:

Theorem 3.1 *Any s-term DNF over $\{0,1\}^n$ in which each conjunction is of size at most t can be expressed as a polynomial threshold function of degree $O(\sqrt{t}\log s)$.*

A useful feature of Theorem 3.1 is that the degree bound depends on $\sqrt{t}$ which can be much smaller than $\sqrt{n}$. Close inspection of the results due to Tarui/Tsukiji reveals that a similar theorem can be derived from their analysis. An advantage of our proof (which is self-contained and does not use approximate inclusion-exclusion or boosting) is that it highlights this dependence which plays a crucial role in our later results.

We then use Theorem 3.1 to give several new results about the degree of polynomial threshold functions which compute various classes of Boolean formulas.

By combining Theorem 3.1 with a decomposition technique due to Bshouty (Bshouty, 1996) we obtain our main result:

Theorem 3.2 *Any s-term DNF over $\{0,1\}^n$ can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log s)$.*

Theorem 3.2 essentially closes the gap which was left open by the $O(\sqrt{n}\log n)$ upper bounds implicit in (Bshouty, 1996; Tarui and Tsukiji, 1999); it shows that the Minsky-Papert lower bound is in fact tight, up to a logarithmic factor, for *all* polynomial-size DNF. Theorem 3.3 also yields a $2^{O(n^{1/3}\log^2 n)}$-time algorithm for learning polynomial-size DNF, which improves on the algorithms of Bshouty and Tarui/Tsukiji and is the fastest known algorithm for the unrestricted DNF learning problem.

We can improve upon the bounds of Theorem 3.2 for read-once DNF:

[1]It is stated in (Beigel *et al.*, 1991) that Minsky and Papert gave an $\Omega(\sqrt{n})$ lower bound for DNF but this was in error (Beigel, 2000).

Theorem 3.3 *Any read-once DNF over $\{0,1\}^n$ can be expressed as a polynomial threshold function of degree $O(n^{1/3}\log^{2/3} n)$.*

Finally, since a DNF formula is simply a Boolean circuit of depth 2, it is natural in this context to also consider Boolean circuits of fixed depth $d \geq 3$. We would ultimately like to prove upper bounds on the degree of polynomial threshold functions for arbitrary constant depth circuits; however this seems to be quite difficult. As a first step in this direction, we prove

Theorem 3.4 *For $d \geq 3$, any read-once Boolean formula of depth d over $\{\wedge, \vee, \neg\}$ can be computed by a polynomial threshold function of degree $\tilde{O}(n^{1-\frac{1}{3\cdot 2^{d-3}}})$.*

Theorem 3.4 implies that the class of read-once constant depth formulas can be learned in subexponential time.

3.2 Preliminaries

Recall that a *disjunctive normal form* formula (DNF) is a disjunction $T_1 \vee \cdots \vee T_s$ of conjunctions of Boolean literals. An s-term DNF is one which has at most s conjunctions (also known as terms) and a t-DNF is one in which each term is of size at most t. A DNF (or Boolean formula) is *read-once* if it contains at most one occurrence of each variable.

A k*-decision list* is a list $L = (T_1, f_1), \ldots, (T_m, f_m)$ where each T_i is a term of size at most k and each f_i is a Boolean function on $\{0,1\}^n$. Given an input $x \in \{0,1\}^n$ the value of $L(x)$ is $f_j(x)$ where $1 \leq j$ is such that $T_j(x) = 1$ and $T_i(x) = 0$ for $i < j$. If $T_i(x) = 0$ for all $1 \leq i \leq m$ then $L(x) = 1$.

A k*-decision tree* is a rooted binary tree where each internal node has 2 children and is labeled with a term of size at most k and each leaf is labeled with a Boolean function. A decision tree represents a Boolean function as follows: if the root is labeled with a term T then then to compute the value of the tree on an input $x \in \{0,1\}^n$ we go left from the root if $T(x) = 0$ and go right if $T(x) = 1$. We continue in this fashion until reaching a leaf ℓ labeled with some function f_ℓ and then output $f_\ell(x)$.

The *rank* of a decision tree T is defined inductively as follows:

- If T is a single leaf then $rank(T) = 0$.

- If T has subtrees T_0 and T_1 then

$$rank(T) = \begin{cases} \max\left(rank(T_0), rank(T_1)\right) & \text{if } rank(T_0) \neq rank(T_1) \\ rank(T_0) + 1 & \text{otherwise.} \end{cases}$$

The following lemma will be useful:

Lemma 3.5 (Blum, 1992) *Let T be a 1-decision tree of rank r whose leaves are labeled with the functions $f_1, \ldots, f_R$. Then there is an r-decision list $(T_1, f_1), \ldots, (T_R, f_R)$ which is equivalent to T.*

A *polynomial threshold function* is defined by a multivariate polynomial $p(x_1, \ldots, x_n)$. The value of the polynomial threshold function on input $x \in \{0,1\}^n$ is $\text{sign}(p(x))$, i.e. 1 if $p(x_1, \ldots, x_n) \geq 0$ and is -1 otherwise. The *degree* of a polynomial threshold function is simply the degree of the polynomial p. If each coefficient a_α of the polynomial is an integer, then the *weight* of the polynomial threshold function is $\sum |a_\alpha|$.

3.2.1 Efficient Algorithms for Learning Linear Threshold Functions

Our learning results for DNF hold in two widely studied learning models: the *Probably Approximately Correct* (PAC) model introduced by Valiant (Valiant, 1984) and the model of *exact learning from equivalence queries* introduced by Angluin (Angluin, 1988) and Littlestone (Littlestone, 1988). The PAC learning model was defined in Section 2.2; we now briefly describe the model of exact learning from equivalence queries.

In the model of exact learning from equivalence queries, learning proceeds in a sequence of stages. In each stage the learning algorithm submits an *equivalence query* (a Boolean function h) to the teacher. If h is equivalent to the target concept c then the teacher answers "YES" and learning halts; otherwise the teacher sends back a point $x \in \{0,1\}^n$ such that $h(x) \neq c(x)$. A learning algorithm A *learns concept class C in time t* if for all $c \in C$, algorithm A can exactly identify the target c in at most t time steps, using at most t equivalence queries, with hypotheses h which each can be represented with t bits and can be evaluated on any point $x \in \{0,1\}^n$ in time t.

The following fact is well known:

Fact 3.6 (Blumer *et al.*, 1989; Maass and Turan, 1994) *In both the PAC model and the model of exact learning from equivalence queries, there are algorithms which learn the class of linear threshold functions over $\{0,1\}^n$ in time poly(n).*

The algorithms of Fact 3.6 are based on polynomial time linear programming. We will need the following extension of Fact 3.6:

Fact 3.7 *Let C be a class of functions each of which can be expressed as an degree-d polynomial threshold function over $\{0,1\}^n$. Then in both the PAC learning model and the model of exact learning from equivalence queries, there is a learning algorithm for C which runs in time $n^{O(d)}$.*

Proof: For the model of exact learning from equivalence queries, we run the polynomial time algorithm for learning linear threshold functions over an expanded version of the input space. Since $z^2 = z$ for $z \in \{0,1\}$ we can suppose without loss of generality that the target polynomial threshold function is a multilinear polynomial of degree d. Such a polynomial threshold function can be viewed as a linear threshold function over the space of all multilinear monomials of degree at most d. There are $N = \sum_{i=1}^{d} \binom{n}{i} \leq n^d$ such monomials and hence by Fact 3.6 we can learn such a polynomial threshold function in the model of exact learning from equivalence queries by running a poly(N)-time algorithm for learning linear threshold functions over the domain $\{0,1\}^N$ where $N \leq n^d$.

For the PAC learning model, we can use a standard transformation from the model of exact learning from equivalence queries to the PAC learning model (Angluin, 1988). It is easy to verify that this transformation (in which equivalence queries are simulated by testing the hypothesis against a set of examples drawn from the PAC oracle) preserves running time up to polynomial factors. Alternatively, using the fact (Anthony, 1995) that the Vapnik-Chervonenkis dimension of the class of polynomial threshold functions of degree d over n variables is $\sum_{i=0}^{d} \binom{n}{d}$, a direct proof can be given along the lines of (Blumer *et al.*, 1989). ■

3.2.2 The Minsky-Papert Lower Bound

It is clear that any depth-1 circuit over $\{\wedge, \vee, \neg\}$ can be expressed as a polynomial threshold function of degree 1. In contrast, Minsky and Papert gave a $\Omega(n^{1/3})$ lower bound on the degree of any polynomial threshold function which computes a particular read-once DNF. For completeness we give their simple proof.

Theorem 3.8 (Minsky and Papert, 1968) *Let $f = T_1 \vee \cdots \vee T_m$ be an m-term DNF over $\{0,1\}^n$ where each term T_i is a conjunction over $4m^2$ variables, each variable appears in precisely one term, and $n = 4m^3$. Then any polynomial threshold function which computes f must have degree at least m.*

Proof: Let $p(x_1, \ldots, x_n)$ be a polynomial of degree d such that for all $x \in \{0,1\}^n$ we have $p(x) \geq 0$ iff x satisfies f. For $i = 1, \ldots, m$ let S_i be the set of $4m^2$ variables which appears in term T_i. It is clear that for any permutations $\pi_1, \ldots, \pi_m$ over a set of size $4m^2$, we have $p(S_1, \ldots, S_m) \geq 0$ iff $p(\pi_1(S_1), \ldots, \pi_m(S_m)) \geq 0$. Consequently the polynomial

$$q(x_1, \ldots, x_n) = \sum_{\pi_1, \ldots, \pi_m} p(\pi_1(S_1), \ldots, \pi_m(S_m))$$

is of degree at most d and has $q(x_1, \ldots, x_n) \geq 0$ iff x satisfies f. Since $q(x)$ is symmetric in the elements of each set S_i, one can straightforwardly show that there is a polynomial $r(\sum_{S_1} x_j, \cdots, \sum_{S_m} x_j)$ of degree at most d such that $r(\sum_{S_1} x_j, \ldots, \sum_{S_m} x_j) = q(x_1, \ldots, x_n)$ for all $x \in \{0,1\}^n$. It follows from the definition of f that for all $(a_1, \ldots, a_m) \in \{0, 1, \ldots, 4m^2\}^m$, we have $r(a_1, \ldots, a_m) \geq 0$ iff some $a_i = 4m^2$. Let $s(t)$ be the univariate polynomial $r(a_1, \ldots, a_m)$ where $a_i = 4m^2 - (t - (2i - 1))^2$ for $i = 1, \ldots, m$. Then the degree of s is at most $2d$, and moreover we have $s(0), s(2), s(4), \ldots, s(2m) < 0$ and $s(1), s(3), \ldots, s(2m-1) \geq 0$. Consequently s has at least $2m$ real zeros, so $2d \geq \deg(s) \geq 2m$. ∎

3.3 An Optimal Bound for Representing DNF by Polynomial Threshold Functions

In this section we prove our main result: any s-term DNF over $\{0,1\}^n$ can be computed by a polynomial threshold function of degree $O(n^{1/3} \log s)$.

3.3.1 Low-Order Polynomial Threshold Functions for DNF with Small Terms

We start by proving Theorem 3.1:

Theorem 3.1 *Any s-term t-DNF can be expressed as a polynomial threshold function of degree $O(\sqrt{t}\log s)$.*[2]

This theorem plays an important role in the proof of the main result. We discuss some other consequences of Theorem 3.1 in Section 3.4.

Proof of Theorem 3.1: The main tools used in the proof are the Chebyshev polynomials of the first kind. These polynomials play an important role in numerical analysis and approximation theory; here we will need only a few simple properties. The d-th Chebyshev polynomial of the first kind, $C_d(x)$, is a univariate degree-d polynomial which satisfies the following conditions (Cheney, 1966):

Lemma 3.9 *The polynomial $C_d(x)$ satisfies*

- $|C_d(x)| \le 1$ *for* $|x| \le 1$ *with* $C_d(1) = 1$;
- $C_d'(x) \ge d^2$ *for* $x > 1$ *with* $C_d'(1) = d^2$.

Let $f = T_1 \vee T_2 \vee \cdots \vee T_s$ be an s-term t-DNF. The *arithmetization* of a Boolean literal ℓ is x_j if $\ell = x_j$ and is $1 - x_j$ if $\ell = \overline{x}_j$. Let S_i denote the sum of the arithmetizations of the literals appearing in T_i and let t_i denote the number of literals in T_i. We define the polynomial

$$Q_i(x) = p\left(\frac{S_i}{t_i}\right)$$

where

$$p(y) = C_d\left(y\left(1 + \frac{1}{t}\right)\right).$$

Here C_d is the d-th Chebyshev polynomial of the first kind and $d = \lceil\sqrt{t}\rceil$. Lemma 3.9 now implies that $p(1) \ge 2$ but $|p(y)| \le 1$ for $y \in [0, 1 - \frac{1}{t}]$.

Consider the polynomial threshold function "$P(x) \ge s + \frac{1}{2}$" where

$$P(x) = \sum_{i=1}^{s} Q_i(x)^{\log 2s}.$$

Since C_d is a polynomial of degree $d = \sqrt{t}$ and S_i is a polynomial of degree 1, this polynomial threshold function has degree $\sqrt{t}\log 2s$. We will show that this polynomial threshold function computes the DNF f exactly.

[2]We analyze the size of the coefficients of the polynomial threshold function in Section 3.4.

Fix any element $x \in \{0,1\}^n$.

- If $f(x) = 0$ then in each term T_i at least one arithmetized literal takes value 0 on x. Thus for each $i = 1, \dots, s$ we have $S_i/t_i \le (t_i - 1)/t_i \le 1 - \frac{1}{t}$ and hence each $|Q_i(x)| \le 1$. Consequently $P(x) \le s$.

- If $f(x) = 1$ then some term T_i must be satisfied by x so $S_i/t_i = 1$. Consequently $Q_i(x) \ge 2$ and hence $Q_i(x)^{\log 2s}$ contributes at least $2s$ to $P(x)$. Since $Q_i(x)^{\log 2s} \ge -1$ for all i, we have $P(x) \ge s + 1$. ■

3.3.2 From DNF to Decision Trees

Let f be an arbitrary s-term DNF over n variables. As the first step in our construction of a polynomial threshold function for f, we transform f into a 1-decision tree in which each leaf is a DNF with small terms; this is a refinement of a transformation given in (Bshouty, 1996). Our original proof gave a bound on the size of the resulting decision tree. S. Lokam (Lokam, 2001) has observed that a slightly stronger bound can be obtained by considering the rank of the decision tree instead. We use Lokam's approach in the following lemma:

Lemma 3.10 *Let $f : \{0,1\}^n \to \{-1,1\}$ be an s-term DNF. For any value $1 \le t \le n$, f can be expressed as a 1-decision tree T where*

- *each leaf of T contains an s-term t-DNF,*

- *T has rank at most $(2n/t)\ln s + 1$.*

Proof of Lemma 3.10: Let $T_1, \dots, T_p$ be the terms of f that have size at least t. Since each term T_i contains at least t literals, there must be some variable x_i that occurs (either negated or unnegated) in at least pt/n of these terms. This variable x_i is placed in the root of the decision tree, and the left and right children of x_i will be decision trees for the restrictions $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$ respectively. This construction is recursively carried out for each of the functions $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$, stopping when a DNF with no terms larger than t is obtained.

It is clear that this recursive procedure generates some 1-decision tree T. Since the function obtained by fixing some subset of variables of an s-term DNF is an s-term DNF, we have that each leaf of T contains an s-term t-DNF.

Let $r(n,p)$ be the maximum (taken over all DNFs on n variables with p terms having size at least t) rank of the decision tree generated by the above procedure. We bound $r(n,p)$ using the following simple observation: if T_a is a term of f which contains an unnegated (negated) variable x_i ($\overline{x}_i$), then the restriction $f|_{x_i \leftarrow 0}$ ($f|_{x_i \leftarrow 1}$) causes the term T_a to vanish. Since the variable x_i at the root of T occurs in at least pt/n terms of size at least t, for at least one of the bit values $b \in \{0,1\}$ the restriction $f|_{x_i \leftarrow b}$ will be a DNF which has at most $p(1-\frac{t}{2n})$ terms of size at least t. Let T_0 (T_1) denote the subtree of T which corresponds to the restriction $f|_{x_i \leftarrow 0}$ ($f|_{x_i \leftarrow 1}$), and suppose without loss of generality that $f|_{x_i \leftarrow 0}$ is a s-term DNF which has at most $p(1-\frac{t}{2n})$ terms of size at least t. Note that $rank(T_0) \leq r(n-1, p(1-\frac{t}{2n}))$ and $rank(T_1) \leq r(n-1,p)$. We consider several cases:

- If $rank(T_0) < rank(T_1)$, then $rank(T) = rank(T_1)$ and hence $r(n,p) \leq r(n-1,p)$.
- If $rank(T_0) > rank(T_1)$, then $rank(T) = rank(T_0)$ and hence $r(n,p) \leq r(n-1, p(1-\frac{t}{2n}))$.
- If $rank(T_0) = rank(T_1)$, then $rank(T) = rank(T_0) + 1$ and hence $r(n,p) \leq r(n-1, p(1-\frac{t}{2n})) + 1$.

To establish initial conditions for the recurrence relation we consider the case $p = 1$. In this case there is one term in f which contains more than t variables; without loss of generality we suppose that this term is $v_1 v_2 \dots v_\ell$. Then the 1-decision list

$$(\overline{v}_1, f|_{v_1 \leftarrow 0}), \dots, (\overline{v}_\ell, f|_{v_\ell \leftarrow 0})$$

is equivalent to a rank-1 decision tree in which each leaf contains an s-term t-DNF. Hence for any n we have $r(n,1) = 1$.

Solving this easy recurrence relation for $r(n,p)$ shows that $r(n,p) \leq (2n/t)\ln p + 1$. Since $p \leq s$ the theorem is proved. ■

3.3.3 An Optimal Bound for Representing DNF by Polynomial Threshold Functions

Theorem 3.2 *Let f be an s-term DNF over n variables. Then f can be expressed as a polynomial threshold function of degree $O(n^{1/3} \log s)$.*

Proof: From Lemma 3.10 and Theorem 3.1, we know that f can be expressed as a 1-decision tree T of rank $(2n/t)\ln s + 1$ where each leaf contains a polynomial threshold function of degree $O(\sqrt{t}\log s)$ (the value of t will be fixed later). From Lemma 3.5 we know that this decision tree T can be expressed as an r-decision list where $r = (2n/t)\ln s + 1$ and each output of the decision list is a polynomial threshold function of degree $O(\sqrt{t}\log s)$. Call this decision list L.

Let $C_1, \ldots, C_R$ be the conjunctions contained in the successive nodes of L and let $P_1(x), \ldots, P_R(x)$ be the corresponding polynomials for the associated polynomial threshold functions at the outputs, i.e. the polynomial threshold function corresponding to the j-th conjunction C_j computes the function "$P_j(x) \geq 0$." If $P_j(x) = 0$ for some $x \in \{0,1\}^n$ then we can replace $P_j(x)$ by $P_j(x) + \delta/2$, where $\delta = \min\{-P_j(x) \ : \ x \in \{0,1\}^n \text{ and } P_j(x) < 0\}$, without changing the function computed by the polynomial threshold function. Now by scaling each P_j by an appropriate multiplicative factor we can suppose without loss of generality that for each $j = 1, \ldots, R$ we have $\min_{x\in\{0,1\}^n} |P_j(x)| \geq 1$.

Consider the polynomial

$$Q(x) = A_1\tilde{C}_1(x)P_1(x) + A_2\tilde{C}_2(x)P_2(x) + \cdots + A_R\tilde{C}_R(x)P_R(x). \tag{3.1}$$

Here $\tilde{C}_j$ is the zero/one valued polynomial which corresponds to the monomial C_j (e.g. if C_j is $x_3\overline{x}_4x_5$ then $\tilde{C}_j(x)$ is $x_3(1-x_4)x_5$). Each value A_j is a positive constant chosen so as to satisfy the following conditions:

$$\begin{aligned}
A_R &= 1, \\
A_{R-1} &> \max_{x\in\{0,1\}^n} |A_R\tilde{C}_R(x)P_R(x)|, \\
&\ \vdots \\
A_j &> \max_{x\in\{0,1\}^n} |A_{j+1}\tilde{C}_{j+1}(x)P_{j+1}(x) + \cdots + A_R\tilde{C}_R(x)P_R(x)|. \\
&\ \vdots \\
A_1 &> \max_{x\in\{0,1\}^n} |A_2\tilde{C}_2(x)P_2(x) + \cdots + A_R\tilde{C}_R(x)P_R(x)|.
\end{aligned}$$

Then the polynomial threshold function "$Q(x) \geq 0$" computes exactly the same function as the decision list L. To see this, fix an input $x \in \{0,1\}^n$. If j is the index of the first conjunction C_j which is satisfied by x, then $\tilde{C}_1(x) = \tilde{C}_2(x) = \cdots = \tilde{C}_{j-1}(x) = 0$, so the only terms of (3.1) which make a nonzero contribution to Q are $A_i\tilde{C}_i(x)P_i(x)$ for $i \geq j$. Since $\tilde{C}_j(x) = 1$ and $|P_j(x)| \geq 1$, the choice of A_j ensures that the sign of $Q(x)$ will be the same as the sign of $P_j(x)$.

The degree of the polynomial $Q(x)$ is at most $(2n/t)\ln s + 1 + O(\sqrt{t}\log s)$. If we take $t = n^{2/3}$ then this value is $O(n^{1/3}\log s)$. ■

Applying Fact 3.7 gives our main DNF learning result:

Corollary 3.11 *The class of s-term DNF formulae over $\{0,1\}^n$ can be learned (in both the PAC model and the model of exact learning from equivalence queries) in time $2^{O(n^{1/3}\log n \log s)}$.*

Remark: Several algorithms are known (Blum *et al.*, 1997; Cohen, 1997) for PAC learning linear threshold functions over $\{0,1\}^n$ in the presence of classification noise in time poly(n). It follows that our time bounds for learning DNF continue to hold in the presence of classification noise.

Corollary 3.12 *The $\Omega(n^{1/3})$ lower bound given by Minsky and Papert for the degree of a polynomial threshold function required to compute a polynomial size DNF over $\{0,1\}^n$ is tight up to a logarithmic factor.*

3.4 Discussion

Since $t \leq n$ in Theorem 3.1, Fact 3.7 already implies that there is a linear-programming based algorithm for PAC learning DNF which takes $2^{O(\sqrt{n}\log n \log s)}$ time steps. Tarui and Tsukiji gave an identical time bound for a different algorithm based on hypothesis boosting using conjunctions. In this section we note that the proof of Theorem 3.1 gives an upper bound on the weight of the resulting polynomial threshold function. This observation can be used to prove correctness of the Tarui/Tsukiji boosting-based algorithm and to show that simpler algorithms such as Winnow or Perceptron can be used to learn $\tilde{O}(\sqrt{n})$ degree polynomial threshold functions which compute DNF (instead of boosting algorithms or algorithms for solving linear programs).

The d-th Chebyshev polynomial $C_d(x) = \sum_{i=0}^{d} a_i x^i$ has all integer coefficients with each $|a_i| \leq 2^d$ (Cheney, 1966). By inspection of the proof of Theorem 3.1 we obtain

Corollary 3.13 *Any s-term t-DNF can be expressed as a polynomial threshold function of degree $O(\sqrt{t}\log s)$ and weight $t^{O(\sqrt{t}\log s)}$.*

Using this corollary we obtain a simple proof of one of the main theorems from (Tarui and Tsukiji, 1999), described in Section 3.1.2, which asserts that for any DNF f and any probability distribution $\mathcal{D}$ there exists some short conjunction which is noticeably correlated with f under $\mathcal{D}$. We use a simple lemma due to Goldmann, Hastad and Razborov ((Goldmann *et al.*, 1992) Lemma 4) which states that if a function f over $\{0,1\}^n$ can be expressed as a majority of at most W ± 1-valued functions (possibly with repetitions) drawn from a set H, then for any distribution $\mathcal{D}$ over $\{0,1\}^n$ there is some function $h \in H$ such that $|\Pr_{x \in \mathcal{D}}[h(x) = f(x)] - \frac{1}{2}| \geq \frac{1}{W}$. In our setting we take H to be the set of all conjunctions of length $O(\sqrt{t}\log s)$ and their negations. There is a clear correspondence between polynomial threshold functions with integer coefficients and depth-2 circuits with a MAJORITY gate at the root and (possibly negated) AND gates at depth 1. Corollary 3.13 gives the required bound on W, and we obtain

Corollary 3.14 *Given any s-term t-DNF f and any distribution $\mathcal{D}$ over $\{0,1\}^n$, there is a conjunction C of size at most $O(\sqrt{t}\log s)$ such that $|\Pr_{x \in \mathcal{D}}[C(x) = f(x)] - \frac{1}{2}| = 2^{-O(\sqrt{t}\log t \log s)}$.*

Taking $t = n$ gives Tarui and Tsukiji's Theorem 1.1, which immediately implies the existence of a boosting-based algorithm for learning DNF in time $2^{\tilde{O}(n^{1/2})}$.

Finally, we observe that the weight bound given in Corollary 3.13 implies that we do not need to solve linear programs (or even use boosting algorithms) in order to learn polynomial-sized DNF in time $2^{\tilde{O}(\sqrt{n})}$. If f is a polynomial threshold function of degree 1 and weight W over the domain $\{0,1\}^N$, then either the Perceptron algorithm or the Winnow algorithm can be used to learn f in $\mathrm{poly}(N, W)$ time steps (see Theorems 5.2 and 5.3 in Chapter 5). As in Fact 3.7, we can view an degree-d polynomial threshold function over $\{0,1\}^n$ as an degree-1 polynomial threshold function over $\{0,1\}^{n^d}$, and thus we can use either the Perceptron or Winnow algorithm to learn s-term DNF in time $2^{O(\sqrt{n}\log n \log s)}$.

3.5 Low-Degree Polynomial Threshold Functions for Read-Once DNF

As seen in Section 3.2.2 the Minsky-Papert $\Omega(n^{1/3})$ lower bound on polynomial threshold function degree for polynomial size DNF is proved using a read-once DNF. Since any read-once DNF can have at most n terms, Theorem 3.2 implies that any read-once DNF can be expressed as a polynomial threshold function of degree $O(n^{1/3} \log n)$. Here we give a slightly better bound:

Theorem 3.3 *Any read-once DNF over variables $x_1, \ldots, x_n$ can be expressed as a polynomial threshold function of degree $O(n^{1/3} \log^{2/3} n)$.*

To prove Theorem 3.3 we use the following sharper version of Lemma 3.10:

Lemma 3.15 *Let $f : \{0,1\}^n \to \{-1,1\}$ be a read-once DNF. For any value $1 \le t \le n$, f can be expressed as a 1-decision tree T where each leaf of T contains a read-once t-DNF and T has rank at most n/t.*

Proof of Lemma 3.15: Let $T_1, \ldots, T_p$ be the terms of f that have size at least t. We use the same decomposition procedure as in Lemma 3.10, and we let $r(n,p)$ be the maximum (taken over all read-once DNFs on n variables with p terms having size at least t) rank of the decision tree generated by the decomposition procedure. Since each variable occurs in at most one term, the recurrence which we obtain in this setting is $r(n,p) \le r(n-1, p-1) + 1$. As before the initial condition is $r(n,1) = 1$ for all n, and thus $r(n,p) \le p$. Since f is read-once we have that $p \le n/t$, and the lemma is proved. (Lemma 3.15) ■

Proof of Theorem 3.3: Let f be a s-term read-once DNF over $\{0,1\}^n$. Lemma 3.15, Theorem 3.1 and Lemma 3.5 together imply that f is computed by a (n/t)-decision list where each output of the decision list is a polynomial threshold function of degree $O(\sqrt{t} \log s)$. As in the proof of Theorem 3.2 there is a polynomial threshold function for f which is of degree $n/t + O(\sqrt{t} \log s)$. Since f is read-once s can be at most n, and taking $t = n^{2/3} / \log^{2/3} n$ proves the theorem. (Theorem 3.3) ■

By the arguments given in Section 2, we immediately have

Corollary 3.16 *The class of read-once DNF can be learned (in both the PAC model and the model of exact learning from equivalence queries) in time $2^{O(n^{1/3} \log^{5/3} n)}$.*

Remark: Standard reductions are known in learning theory which reduce the problem of PAC learning DNF to that of PAC learning read-once DNF (Kearns *et al.*, 1987a). We note that applying these reductions here does *not* yield a $2^{\tilde{O}(n^{1/3})}$-time algorithm for learning arbitrary polynomial-size DNF. The reductions work by converting a DNF with $p(n)$ total occurrences of variables to a read-once DNF over $p(n)$ variables, and thus if used in conjunction with our theorem would yield a $2^{\tilde{O}(p(n)^{1/3})}$-time algorithm for learning such a DNF.

3.6 Low-Order Polynomial Threshold Functions for Read-Once Constant Depth Formulae

A natural generalization of our results for DNF would be to prove upper bounds on the degree of polynomial threshold functions which compute circuits of size s and depth d for fixed values of $d \geq 3$. Obtaining such upper bounds, however, seems to be quite difficult. We note that the parity function on n variables is known to require polynomial threshold functions of degree n (Minsky and Papert, 1968; Aspnes *et al.*, 1994), and hence any proof of even an $n-1$ upper bound on the degree of polynomial threshold functions for constant depth polynomial size circuits would immediately yield a new proof of the well-known fact that parity is not computable by constant depth polynomial size circuits (Furst *et al.*, 1984; Hastad, 1986).

As a first step towards proving upper bounds on polynomial threshold function degree for constant depth circuits, we show how the techniques of this chapter can be used to obtain a nontrivial upper bound on the degree of polynomial threshold functions for read-once constant depth formulae.

Theorem 3.4 *For $d \geq 2$, any read-once Boolean formula of depth d over $\{\wedge, \vee, \neg\}$ can be computed by a polynomial threshold function of degree $O(n^{1-\frac{1}{3\cdot 2^{d-3}}} \log^{\frac{1}{3\cdot 2^{d-3}}} n)$.*

Proof: The proof is by induction on d. The base case $d = 2$ is supplied by Theorem 3.3. We suppose that the theorem holds for $d = 2, \ldots, k-1$ and prove it for $d = k$.

Let f be a depth-k read-once formula. We say that a *term* is a gate at the bottom level of f together with the literals that feed into it. Since f is read-once there can be at most n/t terms of size greater than t. We apply the decomposition procedure described in the proof of Lemma 3.10 to transform f into a 1-decision tree whose leaves each contain a depth-k read-once formula in which each term is of size at most t. As in Lemma 3.15 this decision tree is of rank at most n/t.

In each leaf of this tree, we replace each term with a new "dummy" variable that appears only once. We thus obtain a decision tree of rank n/t whose leaves each contain a read-once formula of depth $k-1$ over these dummy variables. By the induction hypothesis, each such formula is equivalent to a polynomial threshold function of degree $O(n^{1-\frac{1}{3\cdot 2^{k-4}}} \log^{\frac{1}{3\cdot 2^{k-4}}} n)$ which is defined over the dummy variables described above.

In each such polynomial threshold function, we now replace each dummy variable with a real-valued polynomial over the original variables which interpolates precisely the Boolean function computed by the original term. Since each term was of size at most t, each such polynomial is of degree at most t. Consequently the function computed at each leaf of the decision tree is a polynomial threshold function of degree $O(tn^{1-\frac{1}{3\cdot 2^{k-4}}} \log^{\frac{1}{3\cdot 2^{k-4}}} n)$.

As in the proof of Theorem 3.3, our original function f can now be expressed as a polynomial threshold function of degree

$$\frac{n}{t} + O(tn^{1-\frac{1}{3\cdot 2^{k-4}}} \log^{\frac{1}{3\cdot 2^{k-4}}} n).$$

Taking $t = n^{\frac{1}{3\cdot 2^{k-3}}} / \log^{\frac{1}{3\cdot 2^{k-3}}} n$ proves the theorem. ■

Corollary 3.17 *The class of read-once constant depth formulae can be learned in subexponential time.*

Chapter 4

Computational Sample Complexity and Attribute-Efficient Learning

4.1 Introduction

We have seen in Chapter 3 that surprisingly efficient learning algorithms can be given for the broad and expressive class of DNF formulae. In contrast to this positive result, we now study the inherent limitations of computationally efficient algorithms for various learning problems. We show that even for concept classes which contain only very simple concepts such as decision lists, a strong inherent tradeoff can exist between the running time and sample complexity of any learning algorithm. As an example of this tradeoff, we construct a concept class of decision lists which, while it can be learned attribute efficiently (by a non polynomial time algorithm) and can be learned in polynomial time (by a non attribute efficient algorithm), cannot be learned attribute efficiently in polynomial time. This gives the first demonstration of a learning problem for which attribute efficient learning is *computationally* hard.

Since unconditional proofs of computational hardness seem to lie well beyond the current state of the art in theoretical computer science, the hardness results which we establish must rely on unproven assumptions. Our results in this chapter are based on general assumptions (the existence of length-preserving one-way permutations) which are widely held in theoretical cryptography and do not rely solely on the hardness of specific computational problems such as factoring or discrete logarithms.

4.1.1 Motivation

In the Probably Approximately Correct (PAC) model of concept learning, the *sample complexity* of a concept class C is the minimum number of labeled examples which any successful learning algorithm for C must require. The sample complexity of a concept class is one of its most important parameters from a learning theory perspective; for instance it is clear that any concept class which has superpolynomial sample complexity cannot be learned in polynomial time. In an important result, Ehrenfeucht *et al.* gave a general lower bound on sample complexity by proving that any algorithm which PAC learns a concept class of Vapnik-Chervonenkis dimension d must use $\Omega(d/\epsilon)$ examples in the worst case (Ehrenfeucht *et al.*, 1989). Since Blumer *et al.* have shown that $\tilde{O}(d/\epsilon)$ examples are information-theoretically sufficient for PAC learning, the Vapnik-Chervonenkis dimension is known to completely characterize the sample complexity of any concept class (Blumer *et al.*, 1989).

While this information-theoretic characterization of sample complexity in terms of VC dimension is both interesting and important, from a computational standpoint the picture is still far from complete. This is because these results of Ehrenfeucht *et al.* and Blumer *et al.* do not address the question of how many examples a *polynomial time* learning algorithm may require for any given concept class. (Of course, since drawing an example takes at least one time step, a polynomial time learning algorithm can require at most polynomially many examples.) Since issues of computational efficiency have occupied a central place in learning theory ever since Valiant's seminal paper (Valiant, 1984), it is of considerable interest to gain a better understanding of the number of examples which polynomial time learning algorithms require.

The first indication that polynomial time learning may be computationally hard for certain concept classes was given by Valiant (Valiant, 1984), who observed that the existence of polynomial time computable pseudorandom functions (Goldreich *et al.*, 1986) implies that the class of polynomial size Boolean circuits is not PAC learnable. Kearns and Valiant subsequently proved that under certain cryptographic hardness assumptions the concept class of polynomial size Boolean formulae cannot be learned in polynomial time (Kearns and Valiant, 1994). This result was significantly refined and extended by Kharitonov (Kharitonov, 1995), who gave a cryptographic hardness result for a more powerful learning model (learning using membership queries), a more restricted concept class (the class AC^0 of constant depth polynomial size circuits), and a weaker criterion for successful learning (learning to accuracy $1/2 - 1/\text{poly}(n)$ under the uniform distribution).

Decatur, Goldreich and Ron were the first to study learning problems for which polynomial time learning is feasible but requires more examples than learning using a computationally unbounded algorithm (Decatur *et al.*, 1999). They introduced the notion of the *computational sample complexity* of a concept class C, which is the minimum number of labeled examples which any polynomial time learning algorithm for C must require. Among other results they established the existence of concept classes which have arbitrarily large gaps between their sample complexity and computational sample complexity:

Theorem 4.2 (Decatur *et al.*, 1999) *Let $p(n)$ be any polynomial such that $p(n) \geq n$. If any one-way function exists, then there is a concept class C of polynomial size circuits such that*

- *any polynomial time PAC learning algorithm for C must use $\Omega(p(n)/\epsilon)$ examples,*
- *there is a computationally unbounded PAC learning algorithm for C which uses $O(n/\epsilon)$ examples.*

The proof of Theorem 4.2 relies essentially on the idea that a pseudorandom generator can be used to hide information from a computationally bounded learner but not from a computationally unbounded learner.

4.1.2 Computational Sample Complexity versus Sample Complexity

We strengthen the results of Decatur *et al.* by establishing stronger gaps between sample complexity and computational sample complexity and by showing that such gaps can exist even for classes whose concepts have an extremely simple and natural representation as decision lists. Our first construction yields a concept class whose concepts are 1-decision lists and which has the following property: a computationally unbounded learner can learn the class from $O(1/\epsilon)$ examples, but under a standard cryptographic assumption any polynomial time learner requires almost $\Theta(n/\epsilon)$ examples. This construction uses error-correcting codes and requires only very basic cryptographic ideas (the notion of a one-way function).

Our second construction makes more extensive use of cryptographic machinery and yields the following result: for any $k \geq 1$ there is a concept class of k-decision lists which can be learned by a computationally unbounded algorithm from $O(1/\epsilon)$ examples, but cannot be learned by any polynomial time algorithm from fewer than $\Omega(n^k/\epsilon)$ examples (under a widely held cryptographic assumption). As we show, this is within a logarithmic factor of the largest possible gap for concept classes of k-decision lists.

We also consider the implications of our results for the phenomenon of *attribute-efficient* learning. Roughly speaking, a concept class C is said to be attribute-efficiently learnable if there is a learning algorithm for C which requires only poly(size(c), $\log n$) examples to learn any concept $c \in C$ over n variables (we give a precise definition in Section 4.5). Attribute-efficient learning algorithms are useful when the target concept depends on few variables but n, the total number of variables, is large. Results of Haussler (Haussler, 1988) and Littlestone (Littlestone, 1988) yield attribute-efficient learning algorithms for k-CNF and k-DNF formulae; more recent results on attribute-efficiency can be found in (Blum *et al.*, 1995; Blum, 1996; Uehara *et al.*, 1997; Valiant, 1999).

Let L_k be the class of 1-decision lists over $\{0,1\}^n$ which are of length k. Since the Vapnik-Chervonenkis dimension of L_k is easily shown to be $O(k \log n)$, the results of Blumer *et al.* imply that there is a brute-force learning algorithm for L_k which uses $O(k \log n)$ examples. This algorithm, while it is attribute-efficient, does not run in poly(n) time. Blum (Blum, 1996) and Valiant (Valiant, 1999) have each posed the question of whether there exists a polynomial time attribute-efficient learning algorithm for L_k. Such an algorithm could potentially be a very useful tool in machine learning.

We take a step toward answering Blum and Valiant's question by providing the first proof that attribute-efficient learning can be *computationally* hard. We do this by exhibiting a concept class of decision lists which can be learned in polynomial time and can be learned by a computationally unbounded attribute-efficient learning algorithm but cannot (under a plausible cryptographic assumption) be learned in polynomial time by any attribute-efficient algorithm.

4.1.3 Our Approach

A common paradigm for concepts and examples is used for all our results. In each of the concept classes which we consider each concept is associated with a secret key; it is easy to exactly identify the target concept if this key is known. Also, in each of our constructions examples come in two types, which we call *useful* and *useless*. Useful examples each contain an encrypted version of the secret key as well as a small amount of unencrypted information about the target concept. Useless examples all have label 0 and contain no information about the target concept.

Our constructions are based on the following simple idea: a computationally unbounded learning algorithm can decrypt the secret key and hence can learn the target concept exactly from a single useful example. Consequently, such a learning algorithm requires few examples. On the other hand, a polynomial time learner cannot decrypt the secret key; instead, it can only use the small amount of unencrypted information in each useful example. Hence intuitively a polynomial time learner will need many useful examples in order to acquire a significant amount of information about the target concept.

4.2 Preliminaries

Throughout this chapter we will only consider classes of concepts which are defined over the Boolean cube $\{0,1\}^n$. Recall that in the Boolean PAC learning model the learner has access to an *example oracle* $EX(c, \mathcal{D}_n)$ which on each call takes one time step and outputs a labeled Boolean example $\langle x, c(x) \rangle$ where x is drawn from the distribution $\mathcal{D}_n$ over $\{0,1\}^n$. Given two Boolean functions h, c and a distribution $\mathcal{D}_n$ over $\{0,1\}^n$, we say that *h is ϵ-accurate under $\mathcal{D}_n$ with respect to c* if $\Pr_{x \in \mathcal{D}_n}[h(x) \neq c(x)] < \epsilon$; alternatively, such a function h is said to *ϵ-approximate the concept c under $\mathcal{D}_n$*. An algorithm L is said to be a *PAC learning algorithm for concept class C* if the following condition holds: for all $n \geq 1$, for every distribution $\mathcal{D}_n$ over $\{0,1\}^n$, for every $c \in C$ and for every $0 < \epsilon, \delta < 1$, if L is given access to $EX(c, \mathcal{D}_n)$ then with probability at least $1 - \delta$, algorithm L outputs a hypothesis h which ϵ-approximates c under $\mathcal{D}_n$.

The following definitions are from (Decatur *et al.*, 1999): The *distribution free information theoretic sample complexity* of a concept class C, denoted $\mathcal{ITSC}(C; n, \epsilon)$, is the minimum sample size (as a function of n and ϵ) needed for PAC learning the class C with accuracy ϵ and confidence $\delta = 9/10$, where no computational limitations exist on the learning algorithms which may be used. The *distribution free computational sample complexity* of a concept class C, denoted $\mathcal{CSC}(C; n, \epsilon)$, is the minimum sample size (as a function of n and ϵ) needed for PAC learning the class C with accuracy ϵ and confidence $\delta = 9/10$, where the learning algorithm must operate in polynomial (in n and $1/\epsilon$) time.

A *k-decision list of length ℓ* over the Boolean variables $x_1, \ldots, x_n$ is a Boolean function L which is represented by a list of ℓ pairs $(m_1, b_1), (m_2, b_2), \ldots, (m_\ell, b_\ell)$, where each m_i is a conjunction of at most k literals over $x_1, \ldots, x_n$ and each b_i is either 0 or 1. Given any $x \in \{0,1\}^n$, the value of $L(x)$ is b_i if i is the smallest index such that m_i is satisfied; if no m_i is satisfied then $L(x) = 0$.

We write $x \circ y$ to denote the concatenation of binary strings x, y and $|x|$ to denote the length of x. We say that a permutation $f : \{0,1\}^* \to \{0,1\}^*$ is *length-preserving* if $|f(x)| = |x|$ for all $x \in \{0,1\}^*$.

A *length-preserving one-way permutation* is a length-preserving permutation f which has the following properties: there is a deterministic polynomial time algorithm which computes f, but for sufficiently large n there is no probabilistic polynomial time algorithm which inverts f on a $1/\text{poly}(n)$ fraction of $\{0,1\}^n$.

4.3 A Construction Using Error-Correcting Codes

In this section we prove the following theorem, which establishes a gap between computational sample complexity and sample complexity for a class of particularly simple concepts.

Theorem 4.3 *Let $0 < \tau < 1$ be any constant. If length-preserving one-way permutations exist, then there is a concept class C_τ which has*

$$\mathcal{ITSC}(C_\tau; n, \epsilon) = O(1/\epsilon)$$

and

$$\Omega(n^{1-\tau}/\epsilon) = \mathcal{CSC}(C_\tau; n, \epsilon) = O(n/\epsilon)$$

where each concept in C_τ *is a* 1*-decision list over* $\{0,1\}^n$.

4.3.1 Error-Correcting Codes

We need some basic terminology from the theory of error-correcting codes; here we follow the terminology of (Sipser and Spielman, 1996; Spielman, 1996). We say that a *binary code of block length* ℓ *and rate* r_ℓ is a code in which codewords are ℓ bits long, where $r_\ell \cdot \ell$ positions are "message bits" that can be filled with any combination of 0's and 1's and the remaining $(1 - r_\ell)\ell$ positions have their contents determined by the message bits. Let $A_\ell : \{0,1\}^{r_\ell \cdot \ell} \to \{0,1\}^\ell$ be a binary code of block length ℓ and rate r_ℓ; for $x \in \{0,1\}^{r_\ell \cdot \ell}$, the j-th bit of the ℓ-bit string $A_\ell(x)$ is denoted by $A_\ell(x)_j$.

We say that the code A_ℓ has *minimum relative distance* δ_ℓ if any pair of distinct codewords $\{A_\ell(x), A_\ell(y)\}$ has Hamming distance at least $\delta_\ell \cdot \ell$. For $\alpha_\ell < \delta_\ell/2$, we say that an algorithm D is an α_ℓ*-decoding algorithm for* A_ℓ if, when D is given a string $z \in \{0,1\}^\ell$ which has Hamming distance at most $\alpha_\ell \cdot \ell$ from some codeword $A_\ell(x)$, the algorithm D outputs x.

The following important theorem is well known in coding theory; strong versions can be found in, e.g., (Sipser and Spielman, 1996; Spielman, 1996).

Theorem 4.4 *There exists a polynomial time-constructible family* $\{A_\ell\}_{\ell=1}^\infty$ *of binary error-correcting codes, where each* A_ℓ *is a function from* $\{0,1\}^{r_\ell \cdot \ell}$ *to* $\{0,1\}^\ell$*, with the following properties:*

- $\lim_{\ell\to\infty} r_\ell > 0$*,* $\lim_{\ell\to\infty} \delta_\ell > 0$ *and* $\lim_{\ell\to\infty} \alpha_\ell > 0$*,*
- *For each* ℓ*, there is an* α_ℓ*-decoding algorithm for* A_ℓ *which runs in time poly*(ℓ)*.*

Recall that in the PAC framework, a learning algorithm succeeds if it can construct a hypothesis which agrees with the target concept on all but a small fraction of points. In the construction which we use to prove Theorem 4.3, such a hypothesis will yield a string z which is close to a codeword $A_\ell(x)$. By the polynomial time decoding algorithm of Theorem 4.4, the ability to find an accurate hypothesis in polynomial time would thus imply the ability to find x in polynomial time. However, we will show that this is impossible (under a cryptographic assumption) if few examples have been seen.

4.3.2 The Concept Class C_τ

Before giving a formal description of the concept class C_τ, we mention that in this concept class the secret key for each concept is composed of many small subkeys, each of which is encrypted separately. The reason is that each useful example will contain a small amount of unencrypted information about exactly one of the subkeys. Hence, unless many useful examples have been seen, there will exist subkeys about which no unencrypted information has been revealed.

Before we can describe the concept class C_τ we must first specify some parameters. Let $\{A_\ell\}_{\ell=1}^{\infty}$ be a fixed family of error-correcting codes with the properties stated in Theorem 4.4 (so the rate is r_ℓ, the minimum relative distance is δ_ℓ, and there is a poly(ℓ)-time α_ℓ-decoding algorithm for A_ℓ). Given a positive integer m, let $q = m^{\frac{1-\tau}{\tau}}$, let ℓ be the smallest integer such that $r_\ell \cdot \ell \geq m$, and let $n = mq + q\ell$. The following facts can be easily verified:

Fact 4.5 $\alpha_\ell = \Theta(1)$, $r_\ell = \Theta(1)$ *(follows from Theorem 4.4).*

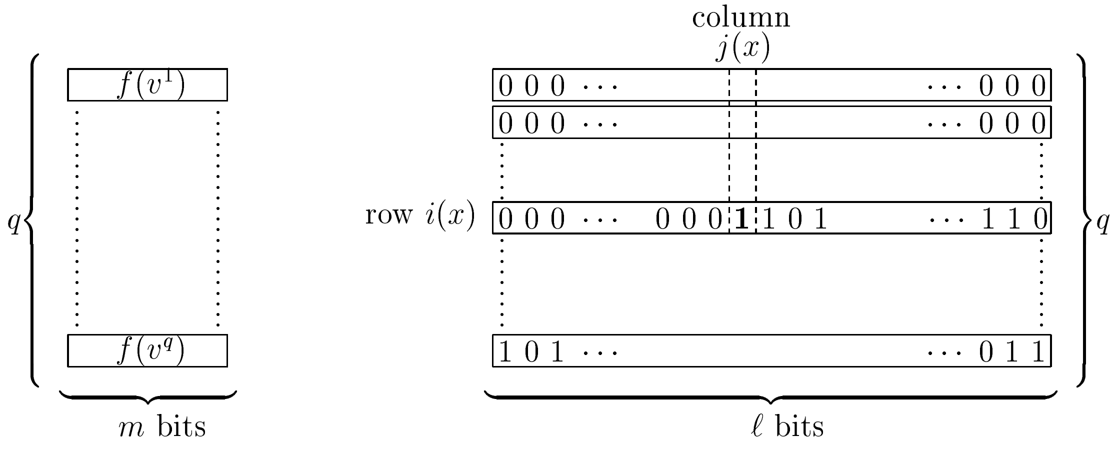
Fact 4.6 $\ell = \Theta(m)$ *(follows from definition of ℓ and Fact 4.5).*

Fact 4.7 $n = \Theta(mq)$ *(follows from definition of n and Fact 4.6).*

Fact 4.8 $m = \Theta(n^\tau)$, $q = \Theta(n^{1-\tau})$ *(follows from definition of q and Fact 4.7).*

Let f be a fixed length-preserving one-way permutation. The set $(\{0,1\}^m)^q$ will be our set of secret keys; each secret key $v = (v^1, \ldots, v^q) \in (\{0,1\}^m)^q$ is composed of q subkeys each of which is m bits long. The class C_τ has a concept c_v for each secret key v.

We now describe a concept c_v over $\{0,1\}^n$. If c_v is the target concept, then an example $x \in \{0,1\}^n$ is said to be *useful* if $x_1 \cdots x_{mq} = f(v^1) \circ \cdots \circ f(v^q)$ and is *useless* otherwise. Given an example $x \in \{0,1\}^n$, let $i(x) \in \{1, \ldots, q\}, j(x) \in \{1, \ldots, \ell\}$ be such that $x_{mq+(i(x)-1)\ell+j(x)}$ is the first bit of $x_{mq+1} \cdots x_{mq+q\ell}$ whose value is 1. If $x_{mq+1} = \cdots = x_{mq+q\ell} = 0$ then we say that $i(x) = j(x) = 0$. Figure 4.1 illustrates the structure of a useful example. The concept c_v is defined as follows:



(a) **(b)**

Figure 4.1: A useful example $\langle x, A_\ell(v^{i(x)})_{j(x)}\rangle$.

Part **(a)** depicts the mq-bit prefix of x; since x is useful this must be $f(v^1)\circ\cdots\circ f(v^q)$. Part **(b)** depicts the $q\ell$-bit suffix $x_{mq+1}\ldots x_n$, where the bit $x_{mq+(r-1)\ell+c}$ is in row r and column c for $1 \le r \le q$, $1 \le c \le \ell$. As shown in **(b)**, the values of $i(x)$ and $j(x)$ are determined by the location of the first 1 in the $q\ell$-bit suffix of x.

- $c_v(x) = 0$ if x is useless,
- $c_v(x) = A_\ell(v^{i(x)})_{j(x)}$, the $j(x)$-th bit of $A_\ell(v^{i(x)})$, if x is useful and $i(x), j(x) \ge 1$. If x is useful and $i(x) = j(x) = 0$ then $c_v(x) = 0$.

4.3.3 Proof of Theorem 4.3

First we establish that c_v is a 1-decision list. For each $1 \le k \le mq$, let ℓ_k denote the literal $\overline{x}_k$ if the k-th bit of $f(v^1)\circ\cdots\circ f(v^q)$ is 1, and let ℓ_k denote x_k otherwise. Then the following is seen to be a 1-decision list which computes c_v :

$$(\ell_1, 0), \ldots, (\ell_{mq}, 0), (x_{mq+1}, A_\ell(v^1)_1), (x_{mq+2}, A_\ell(v^1)_2), \ldots,$$
$$(x_{mq+(i-1)\ell+j}, A_\ell(v^i)_j), \ldots, (x_{mq+q\ell}, A_\ell(v^q)_\ell).$$

This is because the first mq pairs ensure that all useless examples will be labeled 0, and the ordering of the last $q\ell$ pairs ensures that the label of each useful example will be as described in Section 4.3.2.

To prove the information-theoretic sample complexity upper bound, we must show that under any distribution at most $O(1/\epsilon)$ examples are required. Since each positive example contains $f(v^1) \circ \ldots \circ f(v^q)$, a computationally unbounded learner can learn the target concept exactly from a single positive example by inverting the one-way permutation f to find each v^i and then computing each $A_\ell(v^i)$. Such a learner can thus make, say, $20/\epsilon$ calls to the oracle $EX(c, \mathcal{D}_n)$ and output the identically zero hypothesis if all examples are negative, otherwise output the correct hypothesis as described above. A simple calculation shows that this algorithm finds an ϵ-accurate hypothesis with high probability, and hence $\mathcal{ITSC}(C_\tau; n, \epsilon) = O(1/\epsilon)$.

It remains to bound the computational sample complexity of C_τ; we begin with the simpler upper bound. We say that a 1-decision list over $\{0,1\}^n$ is *well-structured* if its length is exactly n and it has the following structure: for $1 \leq t \leq mq$ the t-th pair of the decision list has x_t or $\overline{x}_t$ as its conjunction and has 0 as its output bit, and for $mq + 1 \leq t \leq mq + q\ell$ the t-th term of the decision list has x_t as its conjunction. Given a sample S of examples which are labeled according to the concept c_v, it is easy for a polynomial time algorithm to find a well-structured 1-decision list which is consistent with S. Any positive example of S identifies the first mq literals of the well-structured 1-decision list, and each useful example provides the output bit for one of the last $q\ell$ pairs (note that it is possible to identify useful examples as long as S contains at least one positive example). Since there are 2^n well-structured 1-decision lists, Occam's Razor (Blumer *et al.*, 1987) immediately implies that $O(n/\epsilon)$ examples suffice for this polynomial time learning algorithm.

Now we show the lower bound on $\mathcal{CSC}(C_\tau; n, \epsilon)$. The idea of the proof is as follows: we will exhibit a particular distribution on $\{0,1\}^n$ and show that any polynomial time learning algorithm for C_τ which learns to accuracy ϵ using $q\alpha_\ell/18\epsilon$ examples drawn from this distribution can be used to invert the one-way permutation f in polynomial time with nonnegligible success probability. This contradiction implies that every polynomial time learning algorithm must use more than $q\alpha_\ell/18\epsilon$ examples. Since Facts 4.5 and 4.8 together imply that $q\alpha_\ell/18\epsilon = \Theta(n^{1-\tau}/\epsilon)$, this will prove that $\mathcal{CSC}(C_\tau; n, \epsilon) = \Omega(n^{1-\tau}/\epsilon)$ as desired.

Let $\mathcal{D}_n$ be the distribution on $\{0,1\}^n$ which assigns weight $3\epsilon/(\alpha_\ell \cdot q\ell)$ to each of the $q\ell$ useful examples

$$\{f(v^1) \circ \cdots f(v^q) \circ 0^k 10^{q\ell-k-1}\}_{k=0}^{q\ell-1}.$$

and assigns the remaining $1 - 3\epsilon/\alpha_\ell$ weight to the single useless example 0^n. Recall from Section 4.3.1 that α_ℓ is the frequency of errors up to which the error-correcting codes of 4.4 can be successfully decoded using a poly(ℓ)-time algorithm. Note that under this distribution, each bit of each $A_\ell(v^i)$ is equally likely to occur as the label of a useful example.

Let S be a sample of $q\alpha_\ell/18\epsilon$ examples which are drawn from $EX(c, \mathcal{D}_n)$. Since the expected number of useful examples in S is $q/6$, a simple application of Chernoff bounds shows that with overwhelmingly high probability the sample S will contain at least one useful example. Since each useful example contains $f(v^1) \circ \cdots f(v^q)$ as its mq-bit prefix, it follows that with overwhelmingly high probability a polynomial time learning algorithm which has access to S can identify the strings $f(v^1), \ldots, f(v^q)$.

Now suppose that a polynomial time learning algorithm could achieve an ϵ-accurate hypothesis from the sample S. Since the learning algorithm knows $f(v^1), \ldots,$ $f(v^q)$, the algorithm can apply its ϵ-accurate hypothesis to each of the $q\ell$ useful examples described above. The algorithm can thus construct $B^1, \ldots, B^q$ in polynomial time, where each B^i is an ℓ-bit string which is the learning algorithm's "guess" at the string $A_\ell(v^i)$. Since by assumption the hypothesis is ϵ-accurate under $\mathcal{D}_n$, at most an $\alpha_\ell/3$ fraction of the $q\ell$ total bits in the strings $B^1, \ldots, B^q$ can be incorrect. By Markov's inequality, at least 2/3 of the B^i's must each have at most $\alpha_\ell \cdot \ell$ incorrect bits; consequently, by using the polynomial time decoding algorithm for A_ℓ, the learning algorithm can find at least 2/3 of the subkeys $\{v^1, \ldots, v^q\}$ in polynomial time. However, since as noted earlier the expected number of useful examples in S is $q/6$, by a straightforward application of Chernoff bounds it is extremely unlikely that S contained more than $q/3$ useful examples. As a result, we have that with very high probability the polynomial time learner has received no information at all (other than $f(v^i)$) for at least 2/3 of the subkeys. It follows that the poly(n)-time learner was able to invert f on at least 1/3 of the $f(v^i)$'s "from scratch." Since each subkey v^i is $m = \Theta(n^\tau)$ bits long, though, our poly(n)-time learner is also a poly(m)-time algorithm; but this contradicts the fact that f is one-way. Hence $\mathcal{CSC}(C_\tau; n, \epsilon) > q\alpha_\ell/18\epsilon = \Omega(n^{1-\tau}/\epsilon)$. (Theorem 4.3) ■

4.4 A Stronger Gap

Now we make more extensive use of cryptographic machinery to prove the following:

Theorem 4.9 *Let $k \geq 1$ be any integer. If length-preserving one-way permutations exist, then there is a concept class C_k which has*

$$\mathcal{ITSC}(C_k; n, \epsilon) = O(1/\epsilon)$$

and

$$\mathcal{CSC}(C_k; n, \epsilon) = \Theta(n^k/\epsilon)$$

where each concept in C_k is a k-decision list over $\{0,1\}^n$.

This strengthens the result of Decatur *et. al.* on distribution-free computational versus information-theoretic sample complexity in two ways: we improve the upper bound on information-theoretic sample complexity from $O(n/\epsilon)$ to $O(1/\epsilon)$, and we prove this stronger gap for a class consisting of much simpler concepts (k-decision lists rather than polynomial size circuits).

4.4.1 Cryptographic Preliminaries

The cryptographic definitions we present in this section are slightly more general than the standard definitions which can be found in, e.g., (Goldwasser and Bellare, 1996); we will need this extra generality in Section 4.5. Throughout this section the function $q(\cdot)$ denotes an arbitrary nondecreasing integer-valued function which satisfies $q(n) \geq n$. The standard cryptographic definitions are obtained if $q(n)$ is taken to be a polynomial in n (the reader is encouraged to verify this for herself). Intuitively, the faster $q(n)$ grows, the less plausible are the resulting cryptographic assumptions. In Section 4.5 we will take $q(n)$ to be a function which grows very slightly faster than any polynomial in n; this is a stronger-than-standard cryptographic assumption, but as we discuss in Section 4.5 we believe that it is still quite a reasonable assumption.

The notation "$x \in \mathcal{D}_n$" means that x is selected from the set $\{0,1\}^n$ according to distribution $\mathcal{D}_n$. We write $\mathcal{U}_n$ to denote the uniform distribution over $\{0,1\}^n$.

Definition 4.10 *A length-preserving permutation f is said to be* $q(n)$-one-way *if the following conditions hold:*

- *there is a deterministic algorithm which runs in time poly($|x|$) and computes $f(x)$,*
- *for all probabilistic poly($q(n)$)-time algorithms A, for all polynomials Q, for all sufficiently large n, we have*

$$\Pr_{x \in \mathcal{U}_n}[A(f(x)) = x] < \frac{1}{Q(q(n))}.$$

Definition 4.11 *Let f be a length-preserving permutation. A polynomial time computable predicate $B : \{0,1\}^* \to \{0,1\}$ is said to be a $q(n)$-*hard-core predicate of f *if the following condition holds: for all probabilistic poly($q(n)$)-time decision algorithms A, for all polynomials Q, for all sufficiently large n, we have*

$$\Pr_{x \in \mathcal{U}_n}[A(f(x)) = B(x)] < \frac{1}{2} + \frac{1}{Q(q(n))}.$$

Suppose that g is a length-preserving poly(n)-one-way permutation. Let $x = p \circ y$ where $|p| = |y| = n$, and let f be the function defined as $f(x) = p \circ g(y)$. It is easy to check that f is also a length-preserving poly(n)-one-way permutation. Goldreich and Levin (Goldreich and Levin, 1989) have shown that $B(x) = \sum_{i=1}^{n} p_i y_i \pmod 2$ is a poly(n)-hard-core predicate for f (see Appendix C.2 of (Goldreich, 1998) for a very readable proof of this result). An entirely straightforward modification of their proof shows that if g is a length-preserving $q(n)$-one-way permutation, then f is a length-preserving $q(n)$-one-way permutation and $B(x)$ is a $q(n)$-hard-core predicate for f.

Definition 4.12 *A family of probability distributions $\{\mathcal{X}_{q(n)}\}$ on $\{0,1\}^{q(n)}$ is said to be $q(n)$-*pseudorandom *if $\{\mathcal{X}_{q(n)}\}$ is poly($q(n)$)-time indistinguishable from $\{\mathcal{U}_{q(n)}\}$. That is, for all probabilistic poly($q(n)$)-time decision algorithms A, for all polynomials Q, for all sufficiently large n, we have*

$$\left| \Pr_{z \in \mathcal{X}_{q(n)}}[A(z) = 1] - \Pr_{z \in \mathcal{U}_{q(n)}}[A(z) = 1] \right| < \frac{1}{Q(q(n))}.$$

Definition 4.13 *A poly($q(n)$)-time deterministic algorithm $G : \{0,1\}^n \to \{0,1\}^{q(n)}$ is said to be a $q(n)$-*pseudorandom generator *if $\{\mathcal{G}_{q(n)}\}$ is a $q(n)$-pseudorandom family of distributions, where $\mathcal{G}_{q(n)}$ is the distribution on $\{0,1\}^{q(n)}$ obtained as follows: to* *select $z \in \mathcal{G}_{q(n)}$, pick $x \in \mathcal{U}_n$ and set $z = G(x)$. We write $G(z)_i$ to denote the i-th bit of $G(z)$.*

Now we can state the following useful theorem:

Theorem 4.14 *Let f be a length-preserving $q(n)$-one-way permutation and let B be a $q(n)$-hard-core predicate of f. Let $G : \{0,1\}^n \to \{0,1\}^{q(n)}$ be defined as follows:*

$$G(x) = B(x) \circ B(f(x)) \circ B(f(f(x))) \circ \cdots \circ B(f^{q(n)-1}(x)).$$

Then G is a $q(n)$-pseudorandom generator. Moreover, the distributions

$$\{G(z) \circ f^{q(n)}(z)\}_{z \in \mathcal{U}_n}$$

and

$$\{w \circ f^{q(n)}(z)\}_{w \in \mathcal{U}_{q(n)}, z \in \mathcal{U}_n}$$

are poly($q(n)$)-time indistinguishable.

In the case where $q(n)$ is a polynomial this theorem is a standard result; see e.g., Proposition 3.17 of (Goldreich, 1995). This construction of a pseudorandom generator, along with the definition of a pseudorandom generator, is originally from (Blum and Micali, 1984). The proof of the more general theorem stated above (where $q(n)$ need not be a polynomial) is a straighforward modification of the proof of the standard result, entirely analogous to the modification of the Goldreich-Levin theorem mentioned above.

Observation 4.15 *We note that by Theorem 4.14, even if a poly($q(n)$)-time algorithm is given $f^{q(n)}(z)$ along with some bits of $G(z)$, the algorithm still cannot predict the unseen bits of $G(z)$ with accuracy significantly better than $1/2$. This is because the ability to do such prediction would violate the poly($q(n)$)-time indistinguishability which is asserted in Theorem 4.14, since clearly no poly($q(n)$)-time algorithm (in fact, no algorithm at all) can successfully predict the unseen bits of a uniformly selected random string.*

4.4.2 The Concept Class C_k

Let f be a length-preserving one-way permutation. The set $\{0,1\}^m$ will be our set of secret keys. As discussed in Section 4.4.1 we can suppose without loss of generality that f has a hard-core predicate. Let G be the $\binom{n}{k}$-pseudorandom generator associated with f whose existence is asserted by Theorem 4.14, so G maps inputs of length m to outputs of length $\binom{m}{k}$. Let $n = 2m$. For $1 \leq i \leq \binom{m}{k}$, let T_i denote the i-th k-element subset of the set $\{m+1, \ldots, 2m\}$ under some fixed and easily computable ordering (e.g., lexicographic), and let z_i be the conjunction $\prod_{j \in T_i} x_j$. Given any input $x \in \{0,1\}^n$, let $i(x)$ be the smallest index in $\{1, \ldots, \binom{m}{k}\}$ such that $z_{i(x)}$ is satisfied by x. If no $z_{i(x)}$ is satisfied by x for $1 \leq i(x) \leq \binom{m}{k}$ then let $i(x) = 0$.

The class C_k has a concept c_v for each secret key $v \in \{0,1\}^m$. If c_v is the target concept, then an example x is *useful* if $x_1 \cdots x_m = f^{\binom{m}{k}}(v)$ and is *useless* otherwise. As in Section 4.4.1, $f^{\binom{m}{k}}(v)$ denotes the result of applying f exactly $\binom{m}{k}$ times to v. The concept c_v is defined as follows:

- $c_v(x) = 0$ if x is useless,
- $c_v(x) = G(v)_{i(x)}$, the $i(x)$-th bit of $G(v)$, if x is useful and $i(x) \geq 1$. If x is useful and $i(x) = 0$ then $c_v(x) = 0$.

4.4.3 Proof of Theorem 4.9

First we show that c_v is a k-decision list. For each $1 \leq j \leq m$, let ℓ_j denote the literal $\overline{x}_j$ if the j-th bit of $f^{\binom{m}{k}}(v)$ is 1, and let ℓ_j denote x_j otherwise. The following k-decision list of length $m + \binom{m}{k}$ computes c_v :

$$(\ell_1, 0), \ldots, (\ell_m, 0), (z_1, G(v)_1), \ldots, (z_{\binom{m}{k}}, G(v)_{\binom{m}{k}}).$$

To bound $\mathcal{ITSC}(C_k; n, \epsilon)$, note that upon receiving a single positive example an unbounded learner can invert $f^{\binom{m}{k}}(v)$ to find v (this is possible since f is a permutation) and thus learn the target concept c_v exactly. As in the proof of Theorem 4.3, it follows that $\mathcal{ITSC}(C_k; n, \epsilon) = O(1/\epsilon)$.

An analogous argument to the computational sample complexity upper bound proof of Theorem 4.3 establishes that $\mathcal{CSC}(C_k; n, \epsilon) = O(\binom{m}{k}/\epsilon) = O(n^k/\epsilon)$.

For the computational lower bound, consider the distribution $\mathcal{D}_n$ over $\{0,1\}^n$ which assigns weight $1-6\epsilon$ to the single useless example 0^n and assigns weight $6\epsilon/\binom{m}{k}$ to each of the $\binom{m}{k}$ useful examples

$$\{f^{\binom{m}{k}}(v) \circ T_i\}_{i=1}^{\binom{m}{k}}$$

(here we are viewing each T_i as an m-bit string in the obvious way). Let S be a sample of $\binom{m}{k}/24\epsilon$ examples which are drawn from $EX(c,\mathcal{D}_n)$. By Theorem 4.14, we have that the string-valued random variables

$$\{G(z) \circ f^{\binom{m}{k}}(z)\}_{z\in\mathcal{U}_m}$$

and

$$\{w \circ f^{\binom{m}{k}}(z)\}_{w\in\mathcal{U}_{\binom{m}{k}}, z\in\mathcal{U}_m}$$

are polynomial time indistinguishable. Consequently, even though a polynomial time learner which has drawn the sample S may discover $f^{\binom{m}{k}}(v)$ from any positive example, by Observation 4.15 such a learner cannot predict the bits of $G(v)$ which it has not seen with accuracy significantly better than 1/2. Since the expected number of useful examples in S is $\binom{m}{k}/4$, a straightforward application of Chernoff bounds shows that with very high probability S will contain fewer than $\binom{m}{k}/2$ useful examples, and thus with very high probability the polynomial time learner will have seen at most half of the $\binom{m}{k}$ bits of $G(v)$. Since useful examples which correspond to the unseen bits of $G(v)$ have weight at least 3ϵ under the distribution $\mathcal{D}_n$, the polynomial time learner's overall error rate will exceed ϵ with very high probability. Hence $\binom{m}{k}/24\epsilon$ examples do not suffice for polynomial time learnability, and we have that $\mathcal{CSC}(C_k; n, \epsilon) \geq \binom{m}{k}/24\epsilon = \Theta(n^k/\epsilon)$. (Theorem 4.9) ■

4.4.4 An Optimal Gap

It is interesting to contrast the bounds given in Theorem 4.9 with other known bounds. The upper bound on information-theoretic sample complexity which is given in Theorem 4.9 is the best possible for nontrivial concept classes. Rivest's polynomial time algorithm for learning k-decision lists (Rivest, 1987) requires $O(\frac{n^k}{\epsilon}\min\{\log n, \log\frac{1}{\epsilon}\})$ examples; thus our lower bound on computational sample complexity could be improved by at most a logarithmic factor for concept classes of k-decision lists. Ehrenfeucht *et. al.* (Ehrenfeucht *et al.*, 1989) have shown that the Vapnik-Chervonenkis dimension of the class of k-decision lists on n variables is $\Theta(n^k)$, and hence $\Omega(n^k/\epsilon)$ examples are required for information-theoretic reasons for learning k-decision lists. Our Theorem 4.9 shows that $\Omega(n^k/\epsilon)$ examples can be required for learning subclasses of k-decision lists for *computational* reasons, even in the absence of any information-theoretic barriers to learning from far fewer examples.

4.5 Hardness of Attribute-Efficient Learning

We now turn our attention to *attribute-efficient* learning algorithms. These algorithms require very few examples relative to the total number of input variables (i.e., attributes), and hence have exceptionally good performance over high-dimensional input spaces which contain many irrelevant attributes. This property has led researchers to apply attribute-efficient learning algorithms to real-world problems such as calendar scheduling (Blum, 1997), text categorization (Dagan *et al.*, 1997), gene structure prediction (Deaton and Servedio, 2001), and context-sensitive spelling correction (Golding and Roth, 1999).

Attribute-efficiency has chiefly been studied in the *online mistake bound* model of concept learning. In this model learning proceeds in a series of trials, where in each trial the learner is given an unlabeled Boolean example $x \in \{0,1\}^n$ and must predict the value $c(x)$. After each prediction the learner is told the true value of $c(x)$ and can update its hypothesis. The mistake bound of a learning algorithm on a target concept c is measured by the worst-case number of mistakes that the algorithm makes over all sequences of examples, and the mistake bound of a learning algorithm for a concept class C is the worst-case mistake bound across all concepts $c \in C$. A learning algorithm L for a concept class C over $\{0,1\}^n$ is said to run in polynomial time if the mistake bound of L on C is $\text{poly}(n)$ and the time required by L to make its prediction and update its hypothesis on each example is $\text{poly}(n)$.

A Boolean function c over $x_1, \ldots, x_n$ is said to *depend* on a variable x_i if there are two strings $y, z \in \{0,1\}^n$ which have $y_j = z_j$ for all $j \neq i$, $y_i \neq z_i$, and $c(y) \neq c(z)$. Let C be a class of Boolean functions on $x_1, \ldots, x_n$ each of which depends on at most r variables and each of which has a description of length at most s under some reasonable encoding scheme. Following (Blum *et al.*, 1995), we say that a learning algorithm L for C in the mistake-bound model is *attribute-efficient* if the mistake bound of L on any concept $c \in C$ is $\mathrm{poly}(r, s, \log n)$. Thus an attribute efficient learning algorithm can learn simple concepts defined over few relevant variables with a mistake bound which can be considerably smaller than the total number of variables, even when the identity of the relevant variables is unknown.

In this section we provide strong evidence that there are concept classes learnable in polynomial time for which attribute-efficient learning is information-theoretically possible but computationally hard. We do this by proving the following theorem:

Theorem 4.16 *For any integer $c \geq 2$, let $\log(c, n)$ denote*

$$\overbrace{\log \log \cdots \log}^{c} n.$$

Let $q(c, n) = n^{\log(c,n)}$. If there is some integer $c \geq 2$ such that length-preserving $q(c, n)$-one-way permutations exist, then there exists a concept class C of $O(\log(c, n))$-decision lists which has the following properties in the mistake-bound model:

- *A computationally unbounded learner can learn C with at most 1 mistake,*
- *C can be learned in polynomial time,*
- *C cannot be learned in polynomial time by an attribute-efficient learning algorithm.*

While the existence of length-preserving $q(c, n)$-one-way permutations is not a standard cryptographic assumption, we believe that it is still a very reasonable assumption. As evidence for this belief, in Section 4.5.2 we sketch an argument which shows that if there does not exist a collection of $q(c, n)$-one-way permutations (where each permutation in the collection is defined over a finite domain)[1], then there must exist algorithms for factoring Blum integers which are far more powerful than any currently known algorithms for this well-studied problem.

[1] This is a slightly different notion of a one-way function than the notion which we have been using thus far. See Section 2.4.2 of (Goldreich, 1995) for a discussion of the relationship between these two notions.

4.5.1 Proof of Theorem 4.16

First we define the concept class C. This construction is similar to the construction of Section 4.4.2 but with some different parameters.

Let f be a length-preserving $q(c,n)$-one-way permutation; as before, the set $\{0,1\}^m$ will be our set of secret keys. Let G be the $q(c,n)$-pseudorandom generator whose existence is guaranteed by Theorem 4.14. Let n be such that $m = n^{1/\log(c,n)}$, and let $k(n)$ denote the least integer such that $\binom{m}{k(n)} \geq q(c,m)$. We will use the following claim:

Claim 4.17 *For q, m, and k as defined above, we have:*

1. $q(c,m) = n^{1-o(1)}$.
2. $k(n) = O(\log(c,n))$.

Proof: Recall that $\log(2,n) = \log\log n$ and $\log(c,n) = \log(c-1, \log n)$ for $c > 2$. We first note that

$$\begin{aligned}
\log(c,m) &= \log(c, n^{1/\log(c,n)}) \\
&= \log(c-1, \log(n^{1/\log(c,n)})) \\
&= \log(c-1, \log n/\log(c,n)) \\
&= \log(c-2, \log(\log n/\log(c,n))) \\
&= \log(c-2, \log\log n - \log(c+1,n))
\end{aligned}$$

It follows that

$$\begin{aligned}
q(c,m) &= m^{\log(c,m)} \\
&= \left(n^{1/\log(c,n)}\right)^{\log(c-2, \log\log n - \log(c+1,n))} \\
&= n^{1-o(1)}
\end{aligned}$$

which proves the first part of the claim.

For the second part of the claim, recall that $k(n)$ is the least integer such that $\binom{m}{k(n)} \geq q(c,m)$. We prove that $k(n) \leq 2\log(c,n)$ by showing that $\binom{m}{2\log(c,n)} \geq q(c,m)$.

To see this, note that by the standard inequality $\binom{x}{y} \geq (x/y)^y$, we have

$$\binom{m}{2\log(c,n)} = \binom{n^{1/\log(c,n)}}{2\log(c,n)} \geq \left(\frac{n^{1/\log(c,n)}}{2\log(c,n)}\right)^{2\log(c,n)} = \frac{n^2}{(2\log(c,n))^{2\log(c,n)}} = n^{2-o(1)}.$$

Since $q(c,m) = n^{1-o(1)}$, the claim is proved. ■

For $i = 1, \ldots, q(c,m)$ let T_i denote the i-th $k(n)$-element subset of the set $\{m+1, \ldots, 2m\}$ and let z_i be the conjunction $\prod_{j \in T_i} x_j$. Given any input $x \in \{0,1\}^n$, let $i(x)$ be the the smallest index in $\{1, \ldots, q(c,m)\}$ such that $z_{i(x)}$ is satisfied by x. If no z_i is satisfied by x then $i(x) = 0$.

For each secret key $v \in \{0,1\}^m$, there exists a corresponding concept $c_v \in C$. If c_v is the target concept, then an example x is *useful* if $x_1 \cdots x_m = f^{q(c,m)}(v)$ and is *useless* otherwise. The concept c_v is defined as follows:

- $c_v(x) = 0$ if x is useless,
- $c_v(x) = G(v)_{i(x)}$, the $i(x)$-th bit of $G(v)$, if x is useful and $i(x) \geq 1$. If x is useful and $i(x) = 0$ then $c_v(x) = 0$.

Now we prove that C has the properties listed in Theorem 4.16. The first property is easy: a computationally unbounded learner can achieve a mistake bound of 1 by predicting 0 until it makes a mistake. From this positive example the unbounded learner can compute v (by inverting $f^{q(c,m)}(v)$) and hence can exactly identify the target concept.

For the second property, note that the concept c_v can be represented as a $O(\log(c,n))$-decision list of length at most $m+q(c,m)$. As in the computational sample complexity upper bound of Theorem 4.3, a polynomial time algorithm can learn the first m pairs of the target decision list from a single positive example, and will make at most one mistake for each of the last $q(c,m)$ pairs of the decision list. Since $q(c,m) = n^{1-o(1)}$, such an algorithm will make poly(n) mistakes, and it follows that C can be learned in polynomial time.

Now suppose that there is a polynomial time attribute-efficient learning algorithm for the concept class C. Since each concept c_v has an m-bit description (the string v), we have that $s = O(m)$. Each function c_v depends only on the variables $x_1, \ldots, x_{2m}$, so r is also $O(m)$. Hence any attribute-efficient learning algorithm for C must have mistake bound poly$(m, \log n) =$ poly(m).

Consider the $q(c,m)$-long sequence S of useful examples $\{f^{q(c,m)}(v) \circ T_i \circ 0^{n-2m}\}_{i=1}^{q(c,m)}$. From Theorem 4.14, we have that no poly$(q(c,m))$-time learning algorithm can predict an unseen bit of $G(v)$ with accuracy significantly better than $1/2$. Since $q(c,m) = n^{1-o(1)}$, we have that poly$(q(c,m))$ = poly(n). Consequently, any poly(n)-time learning algorithm will have probability $1/2$ of making a mistake on each example in the sequence S; it follows that with very high probability, any poly(n)-time algorithm will make $\Theta(q(c,m))$ mistakes on S. But this means that no polynomial time attribute-efficient learning algorithm can exist for the concept class C, since poly(m) $= o(q(c,m))$. (Theorem 4.16) ■

4.5.2 Plausibility of the Cryptographic Assumption

Let J_n be the set of all n-bit primes which are congruent to 3 mod 4. An *n-bit Blum integer* is an integer of the form $N = p_1 \cdot p_2$ where $p_1 \neq p_2$ and $p_1, p_2 \in J_{n/2}$. Given an n-bit Blum integer N, let Q_N denote the set of quadratic residues modulo N, and let $f_N : Q_N \to Q_N$ be the function $f_N(z) = z^2 \bmod N$. Blum and Williams have noted that f_N is a permutation on Q_N (see Lemma 2.3.29 of (Goldwasser and Bellare, 1996) for a proof).

As discussed in Section 2.4.3 of (Goldreich, 1995), it is widely believed that the collection

$$\{f_N\}_{Blum} \equiv \{f_N : N \text{ is a Blum integer}\}$$

has the following properties:

1. Given n, it is computationally easy to uniformly select a random n-bit Blum integer N (with negligible error probability) by taking $N = p_1 \cdot p_2$, where p_1, p_2 are uniformly selected $n/2$-bit primes with $p_1 < p_2$ and $p_1 \equiv p_2 \equiv 3 \bmod 4$ (this assumes that the set $J_{n/2}$ of such primes is non-negligibly dense in the set of $n/2$-bit integers).

2. Given an n-bit Blum integer N, it is easy to uniformly select a quadratic residue $r \bmod N$ (this can be done by squaring a randomly chosen element of Z_N^*).

3. For every sufficiently large n, for every probabilistic poly(n)-time algorithm A',

for every polynomial Q, given N and r selected as described above, we have

$$\Pr[A'(f_N(r), N) = r] < \frac{1}{Q(n)}.$$

As in (Goldreich, 1995), we say that $\{f_N\}_{Blum}$ is a *collection of one-way permutations.*

Now consider the following variant of Property 3:

3′. For every sufficiently large n, for every probabilistic poly$(q(n))$-time algorithm A', for every polynomial Q, given N and r selected as described above, we have

$$\Pr[A'(f_N(r), N) = r] < \frac{1}{Q(q(n))}.$$

If $\{f_N\}_{Blum}$ satisfies Properties 1, 2 and 3′ then we say that $\{f_N\}_{Blum}$ is a *collection of $q(n)$-one-way permutations.*

Rabin's results in (Rabin, 1979) yield the following: Suppose that A' is a probabilistic poly$(q(n))$-time algorithm which, when given as input the pair $(f_N(r), N)$ with N and r selected as described above, outputs r with probability at least $1/\text{poly}(q(n))$. Then there is a poly$(q(n))$-time algorithm A which, when given a uniformly selected n-bit Blum integer N, factors N with success probability at least $1/\text{poly}(q(n))$.

Thus, if $\{f_N\}_{Blum}$ is not a collection of $q(c,n)$-one-way permutations, then there is a $q(c,n)$-time algorithm which factors randomly selected Blum integers with success probability at least $1/\text{poly}(q(c,n))$. This would be quite a surprising result, since the fastest known algorithms for factoring n-bit Blum integers require time $2^{\Omega(n^{1/3})}$, which is a much faster-growing function than $q(c,n)$ for all $c \geq 2$ (recall that $q(2,n) = n^{\log\log n}$, $q(3,n) = n^{\log\log\log n}$, etc.).

Chapter 5

PAC Learning with Perceptron and Winnow

In the last chapter we used tools from cryptography to obtain lower bounds on computational sample complexity which hold for any learning algorithm. We now change our focus and take a close look at the abilities and limitations of two particular algorithms for one of the most fundamental problems in machine learning: learning an unknown linear threshold function from labeled examples. The first algorithm we consider, the Perceptron algorithm, is perhaps the most widely known algorithm in machine learning. The second algorithm we consider is Littlestone's Winnow algorithm, a variant of the Perceptron algorithm which has received intensive study over the past dozen or so years. We give both positive and negative results for these two algorithms in the PAC learning model.

5.1 Introduction

The classical Perceptron algorithm (Block, 1962; Novikoff, 1962; Rosenblatt, 1962) and Littlestone's Winnow algorithm (Littlestone, 1988; Littlestone, 1989b) are two algorithms for learning linear threshold functions which have been studied extensively in the online mistake bound model in recent years; see e.g. (Auer and Warmuth, 1995; Bylander, 1998a; Freund and Schapire, 1998; Kivinen *et al.*, 1997; Littlestone, 1991; Maass and Warmuth, 1998; Valiant, 1999). Despite this widespread interest in the Perceptron and Winnow algorithms, relatively little is known about their performance in Valiant's PAC learning model. In this chapter we establish positive and negative results on the PAC learning abilities of Winnow and Perceptron, thus helping to clarify their status in the PAC model.

5.1.1 The Perceptron and Winnow Algorithms

Both Perceptron and Winnow are online mistake bound learning algorithms; we refer the reader to Section 2.4 for a description of this model.

The Perceptron Algorithm

Throughout its execution the Perceptron algorithm maintains a weight vector $w \in \Re^n$ and a threshold $\theta \in \Re$ as its current hypothesis. Initially the Perceptron algorithm starts with $w = (0, \ldots, 0)$ and $\theta = 0$. Upon receiving an example x, the algorithm predicts according to the linear threshold function $w \cdot x \geq \theta$. If the prediction is incorrect the hypothesis is updated according to the following rule:

- If the prediction is 1 and the label is -1 (false positive prediction), set $w \leftarrow w - x$ and $\theta = \theta + 1$.

- If the prediction is -1 and the label is 1 (false negative prediction), set $w \leftarrow w + x$ and $\theta = \theta - 1$.

No change is made if the hypothesis was correct on x. It is clear that if each example x belongs to $\{0, 1\}^n$ then each w_i and θ will always be integers; this fact will prove useful later.

The well-known Perceptron Convergence Theorem gives an upper bound on the number of mistakes which the Perceptron algorithm can make.

Theorem 5.2 (Block, 1962; Novikoff, 1962; Rosenblatt, 1962) *Let $\langle x^1, y_1\rangle, \ldots, \langle x^m, y_m\rangle$ be a sequence of labeled examples with $\|x^i\| \leq R$ and $y_i \in \{-1, 1\}$ for all i. Let u be a vector and θ, ξ be such that $y_i(u \cdot x^i - \theta) \geq \xi$ for all i, where $\xi > 0$. Then the total number of mistakes made by the Perceptron algorithm on this example sequences is at most*

$$\frac{(R^2 + 1)(\|u\|^2 + \theta^2)}{\xi^2}.$$

The Winnow Algorithm

The Winnow algorithm is essentially a modified Perceptron algorithm in which updates are performed multiplicatively rather than additively on each coordinate of the hypothesis vector. As described in (Littlestone, 1991) the Winnow algorithm takes as input an initial vector $w^I \in \Re^n$, a promotion factor $\alpha \in \Re$, and a threshold $\theta \in \Re$. The algorithm requires that the vector w^I be positive (i.e. each coordinate w_i^I is positive), that $\alpha > 1$, and that $\theta > 0$. Like the Perceptron algorithm, Winnow predicts in each stage according to the threshold function $w \cdot x \geq \theta$. If the prediction is correct then no update is performed, otherwise the weights are updated as follows:

- On a false positive prediction, for all i set $w_i \leftarrow \alpha^{-x_i} w_i$;
- On a false negative prediction, for all i set $w_i \leftarrow \alpha^{x_i} w_i$.

It should be noted that in this form Winnow is only capable of expressing positive threshold functions as its hypotheses. As described in (Littlestone, 1988; Littlestone, 1989b) this limitation can be easily overcome by using various simple transformations on the input examples.

Littlestone has proved the following result, analogous to the Perceptron convergence theorem, bounding the number of mistakes which Winnow makes:

Theorem 5.3 (Littlestone, 1991) *Let $\langle x^1, y_1\rangle, \ldots, \langle x^m, y_m\rangle$ be a sequence of labeled examples with $x^i \in [0,1]^n$ and $y_i \in \{-1,1\}$ for all i. Let u be a vector and $\delta > 0$ be such that whenever $y_i = 1$ we have $u \cdot x \geq 1$ and whenever $y_i = -1$ we have $u \cdot x \leq 1 - \delta$. If Winnow is run on this example sequence with initial parameters w^I, α, θ where $1 < \alpha < \frac{1}{1-\delta}$, then the total number of mistakes made by Winnow on this example sequence is at most*

$$\frac{\frac{\alpha(2-\delta)\ln\alpha}{\alpha-1}\sum_{i=1}^n \frac{w_i^I}{\theta} + (\alpha+1)\sum_{i=1}^n u_i \ln \frac{u_i\theta}{e(1-\delta)w_i^I}}{(1-\alpha(1-\delta))\ln\alpha}.$$

If $\alpha = 1 + \frac{\delta}{2}$ and $w^I = (1,\ldots,1)$ then the number of mistakes made by Winnow is at most

$$\frac{8n}{\delta^2\theta} + \max\{0, \frac{14}{\delta^2}\sum_{i=1}^n u_i \ln(u_i\theta)\}.$$

5.1.2 PAC Learning with Perceptron and Winnow: Previous Work

Here we briefly summarize relevant previous research on PAC learning using Winnow and Perceptron. Recall that in the PAC learning model there is a fixed unknown distribution $\mathcal{D}$ from which labeled examples are drawn, and the goal of the learner is to find a hypothesis which closely approximates the target function under distribution $\mathcal{D}$.

As described in Chapter 2, generic techniques exist to convert any online mistake bound learning algorithm to a PAC algorithm. It is easy to see that the conversion procedure due to Littlestone which is described in Chapter 2 (Theorem 2.2) yields a PAC learning algorithm whose running time is polynomially related to the running time of the original online algorithm. Theorems 2.2, 5.2 and 5.3 thus imply that Perceptron and Winnow yield efficient PAC learning algorithms for certain restricted linear threshold learning problems. For example, these theorems can be straightforwardly combined to prove

Corollary 5.4 *Let C_W be the class of linear threshold functions $w \cdot x \geq \theta$ over $\{0,1\}^n$ such that each w_i is an integer and $\sum_{i=1}^{n} |w_i| \leq W$. Then either Perceptron or Winnow can be used to obtain a PAC learning algorithm for C_W which runs in time poly(n, W).*

This corollary implies polynomial-time PAC learnability using Perceptron or Winnow when $W = \text{poly}(n)$. It is well known, though, that there exist linear threshold functions over $\{0,1\}^n$ which require coefficients of size $2^{\Omega(n)}$ (we give an example of such a linear threshold function in Section 5.3). For functions such as these the time bound of Corollary 5.4 is exponentially large and hence not very informative. Can stronger bounds be established on the PAC learning abilities of Perceptron and Winnow? What bounds can be obtained for learning under specific natural distributions such as the uniform distribution?

These questions have been considered by several researchers. Baum has shown that if $\mathcal{D}$ is restricted to be the uniform distribution on the n-dimensional unit sphere S^{n-1} in $\Re^n$, then Perceptron is an efficient PAC learning algorithm for the unrestricted class of linear threshold functions (Baum, 1990). Schmitt has shown that Perceptron is not an efficient PAC learning algorithm for the class of linear threshold functions over $\{0,1\}^n$ (Schmitt, 1998). His proof works by exhibiting a *nested* Boolean function (defined in Section 5.2.2) and a distribution which is concentrated on “hard” examples for that function.

Prior to the results of this chapter, neither positive nor negative results analogous to the above PAC model results for Perceptron had been obtained for the Winnow algorithm. As Schmitt noted, it was not known “whether Littlestone’s rules can PAC identify in polynomial time” (Schmitt, 1998).

5.1.3 PAC Learning with Perceptron and Winnow: Our Results

We first give a negative answer to Schmitt’s question about Winnow by proving that the Winnow algorithm is *not* a polynomial time PAC learning algorithm for the class of positive linear threshold functions. To the best of our knowledge this is the first negative result for Winnow in the PAC model.

We then strengthen Schmitt’s negative result for Perceptron by giving a simple proof that the Perceptron is not an efficient PAC algorithm for the class of linear threshold functions even under the uniform distribution on $\{0,1\}^n$. This negative result provides an interesting contrast to Baum’s positive result for Perceptron under the uniform distribution on the unit sphere.

Finally, we show that under the uniform distribution on Boolean examples, the Perceptron algorithm is an efficient PAC algorithm for the class of nested Boolean functions. This suggests that Schmitt’s negative result for Perceptron on nested functions depends on choosing an adversarial distribution.

5.2 Preliminaries

Recall that a concept class C on an example space X is a collection of Boolean functions on X. In this chapter X will be the Boolean cube $\{0,1\}^n$. A Boolean function $f : X \rightarrow \{-1,1\}$ is a linear threshold function if there is a weight vector $w \in \Re^n$ and a threshold $\theta \in \Re$ such that $f(x) = 1$ iff $w \cdot x \geq \theta$. Such a pair (w, θ) is said to *represent* f. See (Dertouzos, 1965; Muroga, 1971) for extensive treatments of linear threshold functions over $\{0,1\}^n$. We say that $w \cdot x \geq \theta$ is a *positive* threshold function if each w_i is positive.

5.2.1 PAC Learning Using Online Learning Algorithms

In the PAC learning model the learning algorithm has access to an *example oracle* $EX(c, \mathcal{D})$ which, in one time step, provides a labeled example $\langle x, c(x) \rangle$ where x is drawn from the distribution $\mathcal{D}$ on the example space X. The function $c \in C$ is called the *target concept*, and the goal of the learning algorithm is to construct a hypothesis h which, with high probability, has low error with respect to c. We thus have the following natural definition of PAC learning using an online learning algorithm:

Definition 5.5 *We say that an online learning algorithm (such as Winnow or Perceptron) is an* efficient PAC learning algorithm for concept class C over X *if there is a polynomial $p(\cdot, \cdot, \cdot)$ such that the following conditions hold for any $n \geq 1$, any distribution $\mathcal{D}$ over X, any $c \in C$, and any $0 < \epsilon, \delta < 1$:*

- *Given any example $x \in X$, algorithm A always evaluates its hypothesis on x and (once provided with $c(x)$) updates its hypothesis in poly(n) time;*
- *if algorithm A is run on a sequence of examples generated by successive calls to $EX(c, \mathcal{D})$, then with probability at least $1 - \delta$ algorithm A will generate a hypothesis h such that $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] < \epsilon$ after at most $p(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ calls to $EX(c, \mathcal{D})$.*

An online algorithm A is an efficient PAC learning algorithm under distribution $\mathcal{D}$ if it satisfies the above definition for a fixed distribution $\mathcal{D}$.

5.2.2 Nested Functions

Several of our results involve the class of *nested* functions. This class was introduced by Anthony, Brightwell and Shawe-Taylor in (Anthony *et al.*, 1995).

Definition 5.6 *The class of nested functions over $x_1, \ldots, x_n$, denoted NF_n, is defined as follows:*

1. *for $n = 1$, the functions x_1 and $\overline{x}_1$ are nested.*
2. *for $n > 1$, $f(x_1, \ldots, x_n)$ is nested if $f = g * l_n$, where g is a nested function on $x_1, \ldots, x_{n-1}$, $*$ is either $\vee$ or $\wedge$, and l_n is either x_n or $\overline{x}_n$.*

It is easy to verify that the class of nested functions is equivalent to the class of 1-decision lists of length n in which the variables appear in the reverse order $x_n, \ldots, x_1$. The following lemma establishes a canonical representation of nested functions as linear threshold functions:

Lemma 5.7 *Any $f \in NF_n$ can be represented by a linear threshold function*

$$w_1 x_1 + \cdots + w_n x_n \geq \theta_n,$$

with $\theta_n = k + \frac{1}{2}$ for some integer k, $w_i = \pm 2^{i-1}$, and $\sum_{w_i<0} w_i < \theta_n < \sum_{w_i>0} w_i$.

Proof: The proof is by induction on n. For $n = 1$ the appropriate threshold function is $x_1 \geq \frac{1}{2}$ or $-x_1 \geq -\frac{1}{2}$. For $n > 1$, f must be of the form $g * l_n$, where g is a nested function on $x_1, \ldots, x_{n-1}$. By the induction hypothesis, g can be expressed as a threshold function $w_1 x_1 + \cdots + w_{n-1} x_{n-1} \geq \theta_{n-1}$, with $w_1, \ldots, w_{n-1}, \theta_{n-1}$ satisfying the conditions of the lemma. There are four possibilities:

1. $f = g \wedge x_n$: then f is

$$w_1 x_1 + \ldots + w_{n-1} x_{n-1} + 2^{n-1} x_n \geq \theta_n = \theta_{n-1} + 2^{n-1}$$

2. $f = g \wedge \overline{x}_n$: then f is

$$w_1 x_1 + \ldots + w_{n-1} x_{n-1} - 2^{n-1} x_n \geq \theta_n = \theta_{n-1}$$

3. $f = g \vee x_n$: then f is

$$w_1 x_1 + \ldots + w_{n-1} x_{n-1} + 2^{n-1} x_n \geq \theta_n = \theta_{n-1}$$

4. $f = g \vee \overline{x}_n$: then f is

$$w_1 x_1 + \ldots + w_{n-1} x_{n-1} - 2^{n-1} x_n \geq \theta_n = \theta_{n-1} - 2^{n-1}.$$

In each case it can be easily verified that the stated threshold function is equivalent to f and that $w_1, \ldots, w_n, \theta_n$ satisfy the conditions of the lemma. ■

5.3 Winnow Cannot PAC Learn Positive Halfspaces

Although the Winnow algorithm has been studied extensively in the online mistake bound learning model, little is known about its performance in other learning models. In this section we prove the following theorem which shows that Winnow is not an efficient PAC learning algorithm for the class of positive threshold functions over $\{0,1\}^n$.

Theorem 5.8 *Given any positive start vector w^I, any promotion factor $\alpha > 1$ and any threshold $\theta > 0$, there is a positive threshold function c, a distribution $\mathcal{D}$ on $\{0,1\}^n$, and a value $\epsilon > 0$ for which $Winnow(w^I, \alpha, \theta)$ will not generate a hypothesis h such that $\Pr_{x \in \mathcal{D}}[h(x) \neq c(x)] \leq \epsilon$ in poly$(n, \frac{1}{\epsilon})$ steps.*

As a consequence of the proof of this theorem, we will obtain an explicit family of "hard" threshold functions and corresponding example sequences which cause Winnow to make exponentially many mistakes. Maass and Turan (Maass and Turan, 1994) have used a counting argument to show that given any triple (w^I, α, θ), there exists a target threshold function and example sequence which cause Winnow(w^I, α, θ) to make exponentially many mistakes, but no explicit construction was known.

The proof of Theorem 5.8 is based on several lemmas. In the first lemma, we show that a nested Boolean function with alternating connectives requires exponentially large coefficients. Similar results can be found in (Muroga, 1971; Parberry, 1994).

Lemma 5.9 *Let n be odd and let $u \cdot x \geq \theta$ be a positive threshold function on $\{0,1\}^n$ which represents the nested function*

$$f_n = (\ldots(x_1 \vee x_2) \wedge x_3) \vee x_4) \ldots) \vee x_{n-1}) \wedge x_n.$$

For $3 \leq i \leq n$ we have $u_i \geq F_{i-3} u_3$, where F_i is the i-th Fibonacci number: $F_0 = F_1 = F_2 = 1, F_3 = 2, F_4 = 3$, etc..

Proof: The proof is by induction on k, where $n = 2k+1$. For clarity we use two base cases. The case $k = 1$ is trivial. If $k = 2$, then since $f_5(0,0,0,1,1) = 1$ and $f_5(0,0,1,0,1) = 0$, we find that $u_4 > u_3$. Similarly, since $f_5(0,0,1,1,0) = 0$, we find that $u_5 > u_3$.

We now suppose that the lemma is true for the values $k = 1, 2, \ldots, j-1$ and let $n = 2j+1$. By assumption, $(u_1, \ldots, u_n)$ and θ are such that $u \cdot x \geq \theta$ represents f_n. If we fix $x_n = 1$ and $x_{n-1} = 0$, then it follows that $u_1 x_1 + \cdots + u_{n-2} x_{n-2} \geq \theta - u_n$ is a threshold function which represents f_{n-2}, so by the induction hypothesis the lemma holds for $u_3, \ldots, u_{n-2}$, and we need only show that it holds for u_{n-1} and u_n.

Since $f_n(1, 1, \ldots, 1, 0, 0, 1) = 0$, we have that $u_1 + u_2 + \cdots + u_{n-3} + u_n < \theta$. On the other hand, since $f_n(0, 0, \ldots, 0, 1, 1) = 1$, we have that $u_{n-1} + u_n \geq \theta$. From these two inequalities it follows that

$$u_{n-1} > u_1 + u_2 + \cdots + u_{n-3}.$$

Since u is positive, this inequality implies that

$$u_{n-1} > u_3 + u_4 + \cdots + u_{n-3}.$$

Using the induction hypothesis we obtain the inequality

$$u_{n-1} > (1 + F_1 + \cdots + F_{n-6}) u_3 = F_{n-4} u_3.$$

Similarly, since $f_n(1, 1, \ldots, 1, 0) = 0$, we have that $u_1 + u_2 + \cdots + u_{n-1} < \theta$, so

$$u_n > u_1 + u_2 + \cdots + u_{n-2} > u_3 + u_4 + \cdots + u_{n-2}.$$

By the induction hypothesis, we find that

$$u_n > (1 + F_1 + \cdots + F_{n-5}) u_3 = F_{n-3} u_3,$$

and the lemma is proved. ∎

Since $F_n = \Omega(\phi^n)$, where $\phi = \frac{1+\sqrt{5}}{2}$, we have shown that f_n must have coefficients whose ratio is exponentially large.

We require a definition from (Anthony *et al.*, 1995) before stating the next lemma.

Definition 5.10 *Let c be a linear threshold function over $\{0,1\}^n$. We say that c is* consistent *with a set $S = \{\langle x^1, b_1 \rangle, \langle x^2, b_2 \rangle, \ldots, \langle x^m, b_m \rangle\}$ of labeled examples if $c(x^i) = b_i$ for all i. The set S is said to* specify *c if c is the only linear threshold* *function over* $\{0,1\}^n$ *which is consistent with* S*; we say that such a set is a* specifying set *for* c*. The* specification number *of* c*, denoted* $\sigma_n(c)$*, is the smallest size of any specifying set for* c*.*

The following results are proved in (Anthony *et al.*, 1995):

Lemma 5.11

1. *Every linear threshold function* c *over* $\{0,1\}^n$ *has a unique specifying set of size* $\sigma_n(c)$*.*

2. *If* $c \in NF_n$ *then* $\sigma_n(c) = n+1$*.*

3. *Let* $c\!\uparrow\!(x_1,\ldots,x_{n-1})$ *denote* $c(x_1,\ldots,x_{n-1},1)$ *and let* $c\!\downarrow\!(x_1,\ldots,x_{n-1})$ *denote* $c(x_1,\ldots,x_{n-1},0)$*. Then*

$$\sigma_n(c) \leq \sigma_{n-1}(c\!\uparrow) + \sigma_{n-1}(c\!\downarrow).$$

4. *If* c *is a linear threshold function over* $\{0,1\}^n$ *which depends only on coordinates* $1,2,\ldots,k$*, then the specification number of* c *is*

$$\sigma_n(c) = 2^{n-k}\sigma_k(c).$$

We use Lemma 5.11 to show that the function g_n defined below has a small specifying set.

Lemma 5.12 *Let* n *be even and let* g_n *be the Boolean threshold function represented by*

$$x_1 + 2x_2 + 4x_3 + \cdots + 2^{n-2}x_{n-1} + (2^{n-2}+1)x_n \geq 4 + 16 + \cdots + 2^{n-4} + 2^{n-2} + \frac{1}{2}.$$

Then $\sigma_n(g_n) \leq 5n-8$*.*

Proof: The function $g_n\!\downarrow$ is represented by

$$x_1 + 2x_2 + 4x_3 + \cdots + 2^{n-2}x_{n-1} \geq 4 + 16 + \cdots + 2^{n-2} + \frac{1}{2}.$$

It is straightforward to verify that this is precisely the nested function

$$(\ldots(x_1 \vee x_2) \wedge x_3) \vee x_4) \ldots) \wedge x_{n-1}$$

on $n-1$ variables. By Part 2 of Lemma 5.11, we have that $\sigma_{n-1}(g_n \downarrow) = n$.

The function $g_n \uparrow$ is represented by

$$x_1 + 2x_2 + 4x_3 + \cdots + 2^{n-2}x_{n-1} \geq 4 + 16 + \cdots + 2^{n-4} - \frac{1}{2}.$$

Again, one can easily verify that this is precisely the nested function

$$(\ldots(x_3 \vee x_4) \wedge x_5) \vee x_6) \ldots) \wedge x_{n-3}) \vee x_{n-2}) \vee x_{n-1}$$

on the $n-3$ variables $x_3, \ldots, x_{n-1}$; in this nested function the Boolean connectives alternate between $\vee$ and $\wedge$ until the very end, where two consecutive $\vee$'s occur. Since $g_n \uparrow$ does not depend on the two variables x_1, x_2, parts 4 and 2 of Lemma 5.11 imply that $\sigma_{n-1}(g_n \uparrow) = 4\sigma_{n-3}(g_n \uparrow) = 4(n-2)$. It follows from part 3 of Lemma 5.11 that $\sigma_n(g_n) \leq 5n - 8$. ■

One more lemma is required. We prove that the coefficients of x_{n-1} and x_n in any representation of g_n must be almost, but not exactly, equal.

Lemma 5.13 *Let n be even and let g_n be defined as in Lemma 5.12. Let $v \cdot x \geq \theta$ represent g_n. Then $v_{n-1} < v_n < v_{n-1} + v_3$.*

Proof: Let e_j denote the Boolean vector whose j-th coordinate is 1 and all of whose other coordinates are 0. Let $a = e_3 + e_5 + e_7 + \cdots + e_{n-1}$. Since $g_n(a) = 0$, it follows that $v_3 + v_5 + \cdots + v_{n-1} < \theta$. On the other hand, let $b = e_3 + e_5 + \cdots + e_{n-3} + e_n$. Since $g_n(b) = 1$, it follows that $v_3 + v_5 + \cdots + v_{n-3} + v_n \geq \theta$. Combining these two inequalities we find that $v_n > v_{n-1}$.

To see that v_n cannot be much greater than v_{n-1}, let $c = e_1 + e_5 + e_7 + e_9 + \cdots + e_{n-3} + e_n$. Since $g_n(c) = 0$, we have $v_1 + v_5 + \cdots + v_{n-3} + v_n < \theta$. On the other hand, let $d = e_1 + e_3 + \cdots + e_{n-1}$. Since $g_n(d) = 1$, we have that $v_1 + v_3 + \cdots + v_{n-1} \geq \theta$. Combining these two inequalities, we find that $v_n < v_{n-1} + v_3$, and the lemma is proved. ■

Proof of Theorem 5.8: We first prove the theorem for the restricted case in which we assume that $w^I = (1, \ldots, 1)$. After proving this case we will show how this assumption can be eliminated.

Fix $\alpha > 1$ and $\theta > 0$. Let S denote the specifying set for the threshold function g_n; we know from Lemma 5.12 that $|S| \leq 5n - 8$. Let $\mathcal{D}$ be the distribution on $\{0,1\}^n$ which is uniform on S and gives zero weight to vectors outside of S. We will show that with g_n as the target concept, $\mathcal{D}$ as the distribution over examples, and $\epsilon = \frac{1}{5n-7}$ as the error parameter, $Winnow((1, \ldots, 1), \alpha, \theta)$ cannot achieve a hypothesis $h(x)$ which satisfies $\Pr_{\mathcal{D}}[h(x) \neq c(x)] \leq \epsilon$ in $\text{poly}(n, \frac{1}{\epsilon}) = \text{poly}(n)$ steps. To see this, note first that by our choice of ϵ and $\mathcal{D}$, any threshold function $w \cdot x \geq \theta$ which is ϵ-accurate with respect to g_n must be consistent with S. (This technique was first used in (Pitt and Valiant, 1988).) Since S is a specifying set for g_n, though, if $w \cdot x \geq \theta$ is consistent with S then it must in fact agree exactly with g_n. We will show that there is no value of α which would enable Winnow to generate a vector w such that $w \cdot x \geq \theta$ represents $g_n(x)$ in $\text{poly}(n)$ steps.

Let (w, θ) be such that Winnow generates w and $w \cdot x \geq \theta$ represents $g_n(x)$. Since $g_n \downarrow$ is precisely the nested function f_{n-1} of Lemma 5.9 and w is positive, by Lemma 5.9 we have that $w_{n-1} \geq F_{n-4} w_3$. Combining this with Lemma 5.13, we obtain

$$1 < \frac{w_n}{w_{n-1}} < 1 + \frac{1}{F_{n-4}} = 1 + \frac{1}{\Omega(\phi^n)}.$$

Since we assumed that $w^I = (1, \ldots, 1)$, and every example for Winnow lies in $\{0,1\}^n$, it follows that $\frac{w_n}{w_{n-1}} = \alpha^j$ for some positive integer j, and hence that $\alpha = 1 + \frac{1}{\Omega(\phi^n)}$. But then $\Omega(n\phi^n)$ update steps are required to achieve $w_{n-1} \geq F_{n-4} w_3$; consequently, no hypothesis consistent with g_n can be achieved in $\text{poly}(n)$ steps.

Now we consider the case of an arbitrary positive start vector w^I; so fix some positive w^I, $\alpha > 1$, and $\theta > 0$. We assume without loss of generality that $w_1^I \leq w_2^I \leq \ldots \leq w_n^I$, since if this is not already the case renaming variables will make it so. Since all examples for Winnow are in $\{0,1\}^n$, at every point in the execution of Winnow the ratio of weights w_i and w_j must be $\frac{w_i^I}{w_j^I} \cdot \alpha^c$ for some integer c. If there is no integer i_1 such that

$$1 < \frac{w_n^I}{w_{n-1}^I} \cdot \alpha^{i_1} < 1 + \frac{1}{F_{n-4}},$$

then Winnow can never achieve a hypothesis which represents the threshold function $g_n(x)$, and thus can never achieve ϵ-accuracy; so we assume that such an i_1 exists. Similarly, if there is no integer i_2 such that

$$1 < \frac{w^I_{n-1}}{w^I_n} \cdot \alpha^{i_2} < 1 + \frac{1}{F_{n-4}},$$

then there is a threshold function which Winnow can never express (g_n with variables permuted), so such an i_2 must exist as well. Taking the product of these inequalities we find that

$$1 < \alpha^{i_1+i_2} < \Big(1 + \frac{1}{F_{n-4}}\Big)^2.$$

Since $i_1 + i_2$ must be a positive integer, this implies that $\alpha < \Big(1 + \frac{1}{F_{n-4}}\Big)^2$. Now consider a threshold function which requires that $w_1 \geq F_{n-4} w_n$ (again, g_n with a permutation on the variables is such a function). Since $w^I_1 \leq w^I_n$ and $\alpha < \Big(1 + \frac{1}{F_{n-4}}\Big)^2$, it follows that $\Omega(n\phi^n)$ update steps will be required, so no consistent hypothesis can be achieved in polynomial time. (Theorem 5.8) ■

As an easy consequence of this proof, we note that the example sequence which simply cycles through S will force Winnow to make exponentially many mistakes on g_n.

5.3.1 A General PAC Lower Bound for Winnow

The construction of Theorem 5.8 can be used to show that Corollary 5.4 is tight for Winnow up to polynomial factors. For even values $k \leq n$ let g^k_n be the function g_k over variables $x_1, \ldots, x_k$ which we view as belonging to a larger variable set $x_1, \ldots, x_n$. Consider again the construction of Theorem 5.8, but using g^k_n in place of g_n and taking every example x which has nonzero weight under $\mathcal{D}$ to have $x_{k+1} = \cdots = x_n = 0$. The construction goes through as before, and we find that Winnow requires $\Omega(k\phi^k)$ steps to PAC learn g^k_n. It is easily seen that the following lemma implies that there exist w, θ such that $w \cdot x \geq \theta$ represents g^k_n and each w_i is an integer with $\sum_{i=1}^n |w_i| = \Theta(\phi^k)$:

Lemma 5.14 *Let g_n be as defined in Lemma 5.12. Then there exist w, θ such that $w \cdot x \geq \theta$ represents g_n and each w_i is an integer with $\sum_{i=1}^n |w_i| = \Theta(\phi^n)$.*

Proof: Let w, θ be defined as follows:

- $w_1 = 1/4$ and $w_2 = 1/4$;
- for $3 \le i \le n-1$ $w_i = F_{i-1}$ where F_j is the j-th Fibonacci number as defined in Lemma 5.9;
- $w_n = F_{n-2} + 1/4$ and $\theta = F_{n-1} - 3/4$.

Let h_n be the linear threshold function $h_n : \{0,1\}^n \to \{-1,1\}$ which is represented by the above w, θ. It is straightforward to verify that $h_n\!\downarrow$ computes precisely the function

$$(\ldots(x_1 \vee x_2) \wedge x_3) \vee x_4) \ldots) \wedge x_{n-1}.$$

It is also straightforward to verify that $h_n\!\uparrow$ computes precisely the function

$$(\ldots(x_3 \vee x_4) \wedge x_5) \vee x_6) \ldots) \wedge x_{n-3}) \vee x_{n-2}) \vee x_{n-1}.$$

By Lemma 5.12 this implies that h_n and g_n are identical on all of $\{0,1\}^n$. By multiplying each w_i and θ by 4 we obtain a linear threshold function with all integer coefficients which satisfies $\sum_{i=1}^n |w_i| = \Theta(\phi^n)$. ■

We thus have the following result which is complementary to Corollary 5.4.

Corollary 5.15 *The Winnow algorithm, used as a PAC learning algorithm for the class C_W, must run for $\Omega(W \log W)$ time steps.*

5.4 Perceptron is Slow under Uniform Distributions

In the construction just presented the "hard" distribution for Winnow depends on the target linear threshold function. In this section we show that a single natural distribution (the uniform distribution over $\{0,1\}^n$) can thwart the Perceptron algorithm. A very simple argument suffices to establish this result. We require the following definition:

Definition 5.16 *The* weight complexity *of a linear threshold function f over $\{0,1\}^n$ is the smallest natural number t such that f can be represented as $w \cdot x \ge \theta$ with each w_i and θ an integer and $max\{|w_1|, \ldots, |w_n|, |\theta|\} \le t$.*

Theorem 5.17 *The Perceptron algorithm is not an efficient PAC learning algorithm for the class of linear threshold functions under the uniform distribution on* $\{0,1\}^n$.

Proof: We take $\epsilon = \frac{1}{2^n+1}$, so any ϵ-accurate hypothesis must agree exactly with the target concept since misclassification of a single example would imply an error rate under the uniform distribution of at least $\frac{1}{2^n} > \epsilon$. Hastad (Hastad, 1994) has constructed a Boolean threshold function which has weight complexity $2^{\Omega(n \log n)}$. If we take this as our target concept, then it follows that at least $2^{\Omega(n \log n)}$ update steps must be performed by the Perceptron algorithm in order to achieve exact identification since Perceptron hypothesis weights are always integral and each weight is increased by at most 1 during each Perceptron update step. But the amount of computation time which a PAC learning algorithm is allowed is only $\text{poly}(n, \frac{1}{\epsilon}) = \text{poly}(n, 2^n) = 2^{O(n)}$. ■

5.5 Perceptron is Fast for Nested Functions under Uniform Distributions

In this section we establish a sufficient condition for a class of threshold functions to be efficiently learnable by Perceptron under the uniform distribution on $\{0,1\}^n$. We prove that nested functions satisfy this condition and thus obtain the main result of this section. This complements Schmitt's result that the Perceptron algorithm cannot efficiently PAC learn nested functions under arbitrary distributions.

Definition 5.18 *Let* G_n *be a collection of hyperplanes in* $\Re^n$. *A family* $G = \cup_{n \geq 1} G_n$ *of hyperplanes is said to be* gradual *if there is some constant* $c > 0$ *such that the following condition holds: for every* $\tau \geq 0$, *every* $n \geq 1$ *and every hyperplane in* G_n, *at most* $c\tau 2^n$ *Boolean examples* $x \in \{0,1\}^n$ *lie within Euclidean distance* τ *of the hyperplane. A class of linear threshold functions* F *over* $\{0,1\}^n$ *is said to be gradual if there is a mapping* $\varphi : F \to G$, *where* G *is a gradual family of hyperplanes, such that for all* $f \in F$, *if* $\varphi(f)$ *is the hyperplane* $w \cdot x = \theta$, *then* (w, θ) *represents the linear threshold function* f.

Lemma 5.19 *Nested functions can be represented by gradual linear threshold functions over* $\{0,1\}^n$.

Proof: We use the representation established in Lemma 5.7. Let $f \in NF_n$ and let $w \cdot x \geq \theta$ be a linear threshold function which represents f with $w_i = \pm 2^{i-1}$ and $\theta = k + \frac{1}{2}$ for some integer k. For $x \in \{0,1\}^n$, if $w \cdot x = t$ then t must be an integer, but since every integer has a unique binary representation, at most one $x \in \{0,1\}^n$ can satisfy $w \cdot x = t$ for any given value of t. Consequently, no example $x \in \{0,1\}^n$ can have $|w \cdot x - \theta| < \frac{1}{2}$, and

$$|\{x \in \{0,1\}^n : |w \cdot x - \theta| \leq m\}| \leq 2m + 1$$

for $m \geq \frac{1}{2}$. Since the distance from a point x' to the hyperplane $w \cdot x = \theta$ is $\|w\|^{-1} \cdot |w \cdot x' - \theta|$, the lemma follows by noting that we have $\|w\| = (\frac{4^n - 1}{3})^{1/2} = \Theta(2^n)$. ■

This lemma ensures that relatively few points can lie close to to the separating hyperplane for a nested function; consequently, as we run Perceptron, most of the updates will cause the algorithm to make significant progress, and it will achieve ϵ-accuracy in polynomial time. The following theorem formalizes this intuition:

Theorem 5.20 *If C is a gradual class of linear threshold functions over $\{0,1\}^n$ then the Perceptron is a PAC learning algorithm for C under the uniform distribution on $\{0,1\}^n$.*

Proof: The proof is similar to the proof of Theorem 1 in (Baum, 1990). Let $w \cdot x \geq \theta$ be a linear threshold function which represents c. We assume without loss of generality that w, θ have been normalized, i.e. $\|w\| = 1$, so $|w \cdot x - \theta|$ is the distance from x to the hyperplane. By Definition 5.18, there is some constant k such that for all $\tau > 0$, the probability that a random example drawn uniformly from $\{0,1\}^n$ is within distance τ of the hyperplane $w \cdot x = \theta$ is at most $\tau/2k$. Letting $\tau = k\epsilon$, we have that with probability at most $\epsilon/2$ a random example drawn from $\{0,1\}^n$ is within distance $k\epsilon$ of the hyperplane. Let $B \in \{0,1\}^n$ be the set of examples x which lie within distance $k\epsilon$ of the hyperplane, so $\Pr[x \in B] \leq \epsilon/2$.

Let (w_t, θ_t) represent the Perceptron algorithm's hypothesis after t updates have been made. If (w_t, θ_t) is not yet ϵ-accurate, then with probability at most $1/2$ the next example which causes an update will be in B. Consider the following potential function:

$$N_t(\alpha) = \|\alpha w - w_t\|^2 + (\alpha\theta - \theta_t)^2.$$

Recalling the Perceptron update rule, we see that $N_{t+1}(\alpha) - N_t(\alpha)$ is

$$\begin{aligned}
\Delta N(\alpha) &= N_{t+1} - N_t \\
&= \|\alpha w - w_{t+1}\|^2 + (\alpha\theta - \theta_{t+1})^2 - \|\alpha w - w_t\|^2 - (\alpha\theta - \theta_t)^2 \\
&= \mp 2\alpha w\cdot x \pm 2\alpha\theta \pm 2 w_t\cdot x \mp 2\theta_t + \|x\|^2 + 1 \\
&\leq 2\alpha A \pm 2(w_t\cdot x - \theta_t) + n + 1
\end{aligned}$$

Here $A = \mp(w \cdot x - \theta)$ and the top (bottom) sign in each $\pm$ and $\mp$ is in effect if the $(t+1)$-st update is due to a false positive (false negative) example. The last inequality above holds because $\|x\| \leq \sqrt{n}$ for all $x \in \{0,1\}^n$. Since x was misclassified, we know that $\pm(w_t\cdot x - \theta_t) < 0$, and hence $\Delta N(\alpha) < 2\alpha A + n + 1$. It is easy to see that if $x \in B$ then $A \leq 0$ and if $x \notin B$ then $A \leq -k\epsilon$. As a result, $\Delta N(\alpha) < n + 1$ for $x \in B$, and $\Delta N(\alpha) < n + 1 - 2k\epsilon\alpha$ for $x \notin B$.

Suppose that during the course of its execution the Perceptron algorithm has made r updates on examples in B and s updates on examples outside B. Since (w, θ) have been normalized we may assume that $|\theta| \leq \sqrt{n}$, and hence $N_0(\alpha) \leq \alpha^2(n+1)$. Since $N_t(\alpha)$ must always be nonnegative, it follows that

$$0 \leq r(n+1) + s(n+1-2k\epsilon\alpha) + \alpha^2(n+1).$$

If we set $\alpha = \frac{12(n+1)}{5k\epsilon}$, then simplifying the above inequality we obtain

$$0 \leq r - \frac{19}{5}s + \frac{144(n+1)^2}{25(k\epsilon)^2},$$

from which it follows that if $m_1 = \frac{144(n+1)^2}{25(k\epsilon)^2}$ updates have been made, at least 7/12 of the updates must have been on examples in B.

Let $m = \max\{144 \ln \frac{\delta}{2}, m_1\}$ and consider running the Perceptron algorithm for $2m/\epsilon$ examples. Let $h_1, h_2, \ldots$ denote the hypotheses which are generated by the Perceptron algorithm during the course of its execution on these $2m/\epsilon$ examples. We have that

$$\begin{aligned}
\Pr[\text{each } h_i \text{ has error } > \epsilon] &= \Pr[(\text{each } h_i \text{ has error } > \epsilon)\ \& \\
&\qquad (\text{fewer than } m \text{ updates take place})] +
\end{aligned}$$

$$
\begin{aligned}
& \Pr[(\text{each } h_i \text{ has error } > \epsilon) \ \& \\
& \quad (\text{at least } m \text{ updates take place})] \\
\leq \ & \Pr[(\text{fewer than } m \text{ updates take place}) \mid \\
& \quad (\text{each } h_i \text{ has error} > \epsilon)] + \\
& \Pr[(\text{at least } m \text{ updates take place}) \mid \\
& \quad (\text{each } h_i \text{ has error} > \epsilon)].
\end{aligned}
$$

To bound the first of these conditional probabilities, we note that conditioned on the event that each h_i has error at least ϵ, the expected number of updates is at least $2m$. A straighforward Chernoff bound shows that this first conditional probability is at most $\delta/2$. To bound the second conditional probability, we note that

$$
\begin{aligned}
& \Pr[(\text{at least } m \text{ updates take place}) \mid (\text{each } h_i \text{ has error} > \epsilon)] \\
& \quad \leq \ \Pr[(\text{at least } 7/12 \text{ of the } m \text{ updates are on examples in } B) \mid \\
& \quad \quad (\text{each } h_i \text{ has error} > \epsilon)].
\end{aligned}
$$

However, as noted earlier if each h_i has error at least ϵ, then for each update the probability that the update is in B is at most $1/2$. Another application of Chernoff bounds shows that the above probability is at most $\delta/2$, and hence the theorem is proved. ■

As an immediate corollary of Theorem 5.20, we have that the Perceptron is a PAC learning algorithm for the class of nested functions under the uniform distribution on $\{0,1\}^n$.

Chapter 6

Learing Origin-Centered Halfspaces under the Uniform Distribution

In the last chapter we were chiefly concerned with the broad distinction between polynomial and superpolynomial running time for the Perceptron and Winnow algorithms in various PAC model learning scenarios. Now we narrow our focus yet again and give a detailed running time analysis of a new algorithm for a restricted halfspace learning problem. The problem we consider is learning an unknown linear threshold function with threshold $\theta = 0$ (also known as an *origin-centered halfspace*) under the uniform distribution on the unit sphere in $\Re^n$; to compensate for the rather specific nature of this problem we require our learning algorithm to succeed in a fairly demanding noise model. We show that our new algorithm is significantly faster than several previous algorithms which have been considered for this problem. Our approach to this problem, which is extremely simple, introduces a geometric averaging technique which we generalize and extend significantly in the next chapter.

6.1 Introduction

6.1.1 Previous Work

The problem of learning an unknown origin-centered halfspace in $\Re^n$ given access to examples drawn uniformly from the unit sphere has been the subject of considerable research (Baum, 1990; Baum and Lyuu, 1991; Gardner and Derrida, 1989; Kearns, 1998; Long, 1994; Long, 1995; Opper and Haussler, 1991; Seung *et al.*, 1992). While this problem may appear to be significantly easier than the general problem of learning an arbitrary halfspace under an arbitrary distribution, Long has shown that any algorithm for this problem must use at least $\Omega(\frac{n}{\epsilon})$ examples to learn to accuracy ϵ (Long, 1995). Since the general halfspace learning problem can be efficiently solved to accuracy ϵ using $\tilde{O}(\frac{n}{\epsilon})$ examples (Blumer *et al.*, 1989), the two problems are not significantly different in terms of sample complexity. Long has also shown (Long, 1994) that by applying Vaidya's polynomial time linear programming algorithm (Vaidya, 1989) using $O(n^{2.38})$ time matrix multiplication as a subroutine (Coppersmith and Winograd, 1987) it is possible to learn to accuracy ϵ in $\tilde{O}(\frac{n^2}{\epsilon} + n^{3.38})$ time steps using $\tilde{O}(\frac{n}{\epsilon})$ examples.[1] Baum has shown that the Perceptron algorithm learns to accuracy ϵ using $\tilde{O}(\frac{n}{\epsilon})$ examples and running in time $O(\frac{n^2}{\epsilon^3})$ (Baum, 1990). These results of Long and Baum hold in a noise-free setting in which it is assumed that the learning algorithm never receives an example which has been labeled incorrectly.

Angluin and Laird introduced the *classification noise* model in which the label of each example given to the learner is randomly and independently flipped with probability η (Angluin and Laird, 1988). Kearns gave a simple statistical query based algorithm for learning origin-centered halfspaces under the uniform distribution in the presence of classification noise (Kearns, 1998). His algorithm requires $\tilde{O}(\frac{n^2}{\epsilon^2(1-2\eta)^2})$ examples and runs in time $\tilde{O}(\frac{n^3}{\epsilon^2(1-2\eta)^2})$. Blum et al. (Blum *et al.*, 1997) and Cohen (Cohen, 1997) have given polynomial-time algorithms for learning an arbitrary halfspace in the presence of classification noise under an arbitrary distribution. Their algorithms have time complexity at least $O(\frac{n^{14}}{\epsilon^2(1-2\eta)^2})$, though, and hence are not particularly efficient for the uniform distribution problem.

An even more challenging noise model was proposed by Bylander, who introduced the notion of *monotonic noise* for halfspaces (Bylander, 1998a). In this model the probability that an example is labelled incorrectly decreases with the distance of the example from the separating hyperplane; this captures the intuition that examples closer to the decision boundary are harder to label correctly. Bylander gave a

[1] In all of the results which we discuss here, as well as in our own results, we adopt the convention that performing a single arithmetic operation such as multiplication or comparison on a pair of real numbers takes one time step.

variant of the Perceptron algorithm which, for certain distributions, learns an unknown halfspace even in the presence of monotonic noise. Under the uniform distribution his monotonic noise algorithm requires $\tilde{O}(\frac{n^3}{\epsilon^4(1-2\eta)^4})$ examples and runs in time $\tilde{O}(\frac{n^5}{\epsilon^6(1-2\eta)^6})$.

6.1.2 A New Algorithm

We give a new algorithm which learns origin-centered halfspaces under the uniform distribution on the unit sphere. In the presence of monotonic or classification noise our algorithm requires $\tilde{O}(\frac{n}{\epsilon^2(1-2\eta)^2})$ examples and runs in time $\tilde{O}(\frac{n^2}{\epsilon^2(1-2\eta)^2})$. Our algorithm thus improves on the time and sample complexity of previous algorithms in the classification noise and monotonic noise models, improves on the time complexity of the Perceptron algorithm in the noise-free setting, and is incomparable to (but much simpler than) Long's algorithm in the noise free setting.

The new algorithm is extremely simple: it averages a set of labeled examples to obtain a hypothesis vector. The algorithm thus learns using the same hypothesis class (linear threshold functions) as the earlier algorithms mentioned above.

6.2 Preliminaries

Here we provide some useful definitions and probability facts. We denote the n-dimensional unit sphere by

$$S^{n-1} \equiv \{x \in \Re^n : \|x\| = 1\}$$

(note that S^{n-1} lies in $\Re^n$ and not in $\Re^{n-1}$). An *origin-centered halfspace* over S^{n-1} is a function from S^{n-1} to $\{-1, 1\}$ which is defined by a nonzero vector $u \in \Re^n$; the value of $u(x)$ is 1 if $u \cdot x \geq 0$ and is -1 if $u \cdot x < 0$.

In Bylander's demanding *monotonic noise* model, examples closer to the decision boundary are more likely to be mislabeled. In this model the learning algorithm cannot access the noise-free oracle EX but instead must use the monotonic noise oracle EX^{η}_{MN} where $0 \leq \eta < 1/2$ is a fixed noise rate. Given a halfspace defined by a unit vector $u \in S^{n-1}$, the monotonic noise oracle $EX^{\eta}_{MN}(u, \mathcal{D})$ is defined as follows: the oracle first draws an example x according to $\mathcal{D}$ and then flips a biased

Input: example oracle MX
integer $t > 0$

Output: vector v

1. **for** $i = 1$ **to** t **do**
2. Draw $\langle x^i, b_i \rangle$ from MX
3. **return** $v = \frac{1}{t} \sum_{i=1}^{t} b_i x^i$

Figure 6.1: The `Average` algorithm.

coin with Pr[heads] $= \hat{\eta}(|u \cdot x|)$. The oracle outputs $\langle x, -u(x) \rangle$ if the coin comes up heads and outputs $\langle x, u(x) \rangle$ if it comes up tails. The function $\hat{\eta} : [0,1] \to [0,1]$ can be any nonincreasing function such that the overall probability that the oracle returns a mislabelled example is η. Note that the classification noise model is obtained when $\hat{\eta}(\cdot)$ is restricted to be the constant function η, and the original noise-free model is obtained when $\eta = 0$. As in the standard PAC model the learner's goal is to output a hypothesis h which with probability at least $1 - \delta$ has $\Pr_{x \in \mathcal{D}}[h(x) \neq u(x)] \leq \epsilon$, but now the learner is given access to $EX^{\eta}_{MN}(u, \mathcal{D})$, ϵ, δ, and η, and the learner is allowed time polynomial in $n, \frac{1}{\epsilon}, \frac{1}{\delta}$, and $\frac{1}{1-2\eta}$. The dependence on $\frac{1}{1-2\eta}$ is necessary because there is less and less information in the labeling of each example as η approaches $1/2$.

Now we give some useful probability facts. We let $\mathcal{U}_{n-1}$ denote the uniform distribution on S^{n-1}. If $u \in S^{n-1}$ represents the target halfspace and $v \in S^{n-1}$ represents a hypothesis halfspace, then $\Pr_{x \in \mathcal{U}_{n-1}}[v(x) \neq u(x)]$, the error of v with respect to u under $\mathcal{U}_{n-1}$, is the fraction of S^{n-1} which lies in the symmetric difference of the two halfspaces. This is easily seen to be $\theta(u, v)/\pi$, where $\theta(u, v) = \arccos(u \cdot v)$ is the angle between u and v.

Let A_{n-1} denote the surface area of the unit sphere S^{n-1}. It is known (see, e.g., (Baum, 1990)) that $A_{n-1} = 2\pi^{n/2}/\Gamma(n/2)$, where Γ is the classical gamma function. Using basic properties of the gamma function (Artin, 1964) one can show that $A_{n-2}/A_{n-1} = \Theta(n^{1/2})$. For $n \geq 3$, if $u \in S^{n-1}$ is fixed, then as noted in (Baum, 1990) we have

$$\Pr_{x \in \mathcal{U}_{n-1}}[\alpha \leq u \cdot x \leq \beta] = \frac{A_{n-2}}{A_{n-1}} \cdot \int_{\alpha}^{\beta} \left(\sqrt{1 - z^2}\right)^{n-3} dz.$$

6.3 The Algorithm

Throughout the rest of this chapter u denotes the unit vector in $\Re^n$ which represents the target halfspace and MX denotes the monotonic example oracle $EX_{MN}^{\eta}(u, \mathcal{U}_{n-1})$. The learning algorithm we consider, called `Average`, is given in Figure 6.1. It is clear that `Average`(MX, t) uses t examples and runs in time $\Theta(nt)$.

6.3.1 Comparison of `Average` and Perceptron

It is interesting to note that the `Average` algorithm is very similar to the Perceptron algorithm. The only difference betwen the two algorithms, in fact, is that Perceptron is *conservative*, i.e., it only updates its current hypothesis v on an example if v predicts incorrectly on that example. The `Average` algorithm, on the other hand, can be viewed as performing an update on every example. Baum has shown that in the absence of noise the Perceptron algorithm achieves an ϵ-accurate hypothesis with high probability after $O(n/\epsilon^2)$ updates (Baum, 1990). Once the Perceptron hypothesis has achieved $\Theta(\epsilon)$ accuracy, though, an expected $\Theta(1/\epsilon)$ examples are required to generate a single example which will cause an update, and hence $O(n/\epsilon^3)$ examples are required overall. We prove in Section 6.4 that in the absence of noise the `Average` algorithm requires $\tilde{O}(n/\epsilon^2)$ updates. However, since `Average` performs an update on every example, it needs only $\tilde{O}(n/\epsilon^2)$ examples in total. This difference between the two algorithms appears to be a real one and not just an artifact of the analysis. Figure 6.3(a) graphs the error rates of the two algorithms as a function of the number of examples used. It is clear from this graph that Perceptron requires far more examples than `Average` to achieve a given error rate.

Another difference between the two algorithms is their ability to tolerate noise. We prove in Section 6.4 that `Average` can achieve an arbitrarily accurate hypothesis even in the presence of monotonic noise. No such guarantees are known for Perceptron, and it seems likely that none can be given: Figure 6.3(b) graphs the error rates of the two algorithms when run using examples with a classification noise rate of $\eta = 0.10$. One possible explanation for Perceptron's inability to achieve a low error rate is the following: Initially Perceptron's error rate decreases (see Figure 6.3(b)). Once the error rate dips below a certain level, though, the algorithm performs most of its updates on mislabeled examples. Performing these "bad" updates then causes

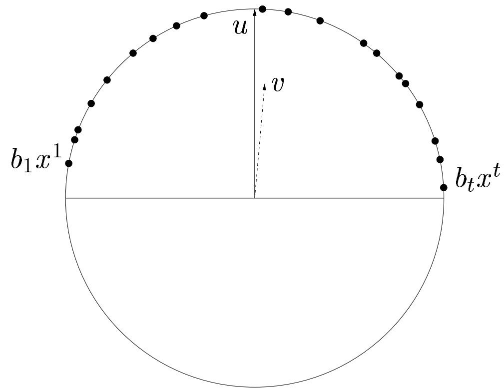


Figure 6.2: An execution of the `Average` algorithm.
In the noise-free case each point b_ix^i lies above the target hyperplane defined by the vector u. Here v is the vector average of $b_1x^1, \ldots, b_tx^t$.

the hypothesis to become less accurate, though, so the error rate increases again. In contrast, since `Average` updates on every example (and most examples are not mislabeled since $\eta < 1/2$), its error rate continues to decrease.

6.3.2 Why `Average` Works

The idea underlying the `Average` algorithm is very simple. If there were no noise in the labeling of the examples, then for each labeled example $\langle x, b\rangle$ it would be the case that $u \cdot (bx) \geq 0$. Furthermore, since the distribution of x is uniform on the unit sphere, the distribution of bx would be symmetrical around the vector u, and hence the expected value of bx would be a vector which points precisely in the direction of u (see Figure 6.2). In fact, even in the presence of monotonic noise at a noise rate of $\eta < 1/2$, it is still the case that $E[bx]$ is a vector which points precisely in the direction of u (recall that for an example x, the probability that x is corrupted by monotonic noise depends only on $|u \cdot x|$). Thus, if we can approximate $E[bx]$ we can find an accurate hypothesis; the `Average` algorithm simply uses sampling to approximate the expected value of bx.

The analysis of the algorithm consists of proving that for a suitable value of t, the angle between u and v (and hence the error of the hypothesis v) does not deviate very much from zero. We do this in two phases: in Section 6.4.1 we show that with high probability the vector v will have a large component lying in the direction of

u, and in Section 6.4.2 we prove that with high probability v will have only a small component which is orthogonal to u. Finally, in Section 6.4.3, we combine these results to complete the proof.

6.4 Analyzing the `Average` Algorithm

6.4.1 A Large Parallel Component

Our first lemma is a lower bound on the expectation of $u \cdot (bx)$. We use this bound later to prove the main result of this subsection.

Lemma 6.2 *Let $\langle x, b\rangle$ be obtained by querying MX. Then we have that*

$$E[u \cdot (bx)] = \Omega\left(\frac{1-2\eta}{n^{1/2}}\right).$$

Proof: Let $\hat{\eta}$ be the monotonic noise function. We first show that $E[u \cdot (bx)] = \Omega(\frac{1-2\eta}{n^{1/2}})$ if $\hat{\eta}$ is the constant function η. If this is the case then for each $x \in S^{n-1}$ we have that $\Pr[b = u(x)] = 1 - \eta$ and $\Pr[b \neq u(x)] = \eta$, so

$$\begin{aligned}
E[u \cdot (bx)] &= 2 \cdot \frac{A_{n-2}}{A_{n-1}} \cdot \int_0^1 (1-2\eta) z \left(\sqrt{1-z^2}\right)^{n-3} dz \\
&= \Omega\left((1-2\eta) n^{1/2} \int_0^{n^{-1/2}} z \left(\sqrt{1-z^2}\right)^{n-3} dz\right) \\
&= \Omega\left((1-2\eta) n^{1/2} \int_0^{n^{-1/2}} z \, dz\right) \\
&= \Omega\left(\frac{1-2\eta}{n^{1/2}}\right).
\end{aligned}$$

The second equality holds because the integrand is never negative, and the third equality holds since $(\sqrt{1-z^2})^{n-3} = \Theta(1)$ for $0 \leq z \leq n^{-1/2}$.

We now show that $E[u \cdot (bx)]$ is minimized when $\hat{\eta}(\cdot)$ is the constant function η. Since

$$E[u \cdot (bx)] = 2 \cdot \frac{A_{n-2}}{A_{n-1}} \cdot \int_0^1 z \cdot (1 - 2\hat{\eta}(z)) \left(\sqrt{1-z^2}\right)^{n-3} dz$$

it suffices to prove that

$$\int_0^1 \hat{\eta}(z) z \cdot \left(\sqrt{1-z^2}\right)^{n-3} dz \leq \int_0^1 \eta z \cdot \left(\sqrt{1-z^2}\right)^{n-3} dz \tag{6.1}$$

for any function $\hat{\eta}$ which satisfies the conditions of monotonic noise. Since the overall probability that $b \neq u(x)$ is η, we have

$$\begin{aligned}\eta &= 2\eta \cdot \frac{A_{n-2}}{A_{n-1}} \cdot \int_0^1 \left(\sqrt{1-z^2}\right)^{n-3} dz \\ &= 2 \cdot \frac{A_{n-2}}{A_{n-1}} \cdot \int_0^1 \hat{\eta}(z) \left(\sqrt{1-z^2}\right)^{n-3} dz.\end{aligned}$$

This implies that for all $0 \leq \beta \leq 1$,

$$\int_0^\beta (\hat{\eta}(z) - \eta) \left(\sqrt{1-z^2}\right)^{n-3} dz \;=\; \int_\beta^1 (\eta - \hat{\eta}(z)) \left(\sqrt{1-z^2}\right)^{n-3} dz.$$

Now let us choose β such that $\hat{\eta}(z) \geq \eta$ for $0 \leq z < \beta$ and $\hat{\eta}(z) \leq \eta$ for $\beta < z \leq 1$. Such a β must exist because $\hat{\eta}$ is a nonincreasing function and the overall probability of misclassification is η. We then have

$$\begin{aligned}\int_0^\beta z \cdot (\hat{\eta}(z) - \eta) \left(\sqrt{1-z^2}\right)^{n-3} dz &\leq \int_0^\beta \beta \cdot (\hat{\eta}(z) - \eta) \left(\sqrt{1-z^2}\right)^{n-3} dz \\ &= \int_\beta^1 \beta \cdot (\eta - \hat{\eta}(z)) \left(\sqrt{1-z^2}\right)^{n-3} dz \\ &\leq \int_\beta^1 z \cdot (\eta - \hat{\eta}(z)) \left(\sqrt{1-z^2}\right)^{n-3} dz.\end{aligned}$$

This implies inequality (6.1), so we have shown that $E[u \cdot (bx)] = \Omega(\frac{1-2\eta}{n^{1/2}})$ for any monotonic noise function $\hat{\eta}$. ■

Now we prove the main result of this section, which establishes that the output of `Average`(MX, t) will with high probability have a substantial component in the direction of u.

Lemma 6.3 *Let $v =$ `Average`(MX, t). Then there is some value $t = \tilde{\Theta}(\frac{n}{(1-2\eta)^2})$ such that*

$$u \cdot v = \Omega\left(\frac{1-2\eta}{n^{1/2}}\right)$$

with probability at least $1 - \delta/2$.

Proof: Since $u \cdot v$ is $\frac{1}{t}$ times a sum of t independent random variables, each of which lies in $[-1, 1]$, we can use Hoeffding's tail bound to obtain

$$\Pr\left[u \cdot v \leq \frac{1}{2}E[u \cdot v]\right] \leq 2 \cdot \exp\left(\frac{-t(E[u \cdot v])^2}{8}\right).$$

By linearity of expectation and Lemma 6.2 we have that

$$E[u \cdot v] = \Omega\left(\frac{1 - 2\eta}{n^{1/2}}\right).$$

Hence in order to bound the above probability by $\delta/2$, it suffices to take

$$t = \Theta\left(\frac{n \log \frac{1}{\delta}}{(1 - 2\eta)^2}\right) = \tilde{\Theta}\left(\frac{n}{(1 - 2\eta)^2}\right).$$

■

6.4.2 A Small Orthogonal Component

Now we must show that with high probability the component of v which is orthogonal to u is not very large. The following lemma shows that with high probability a random flight in R^n which proceeds for t steps will end up at most a distance of (approximately) $t^{1/2}$ away from its starting point. We will use this lemma to prove our desired upper bound.

Lemma 6.4 *Let $x^1, \ldots, x^t$ be independently drawn from $\mathcal{U}_{n-1}$. Then with probability at least $1 - \delta/2$ we have*

$$\left\| \sum_{i=1}^{t} x^i \right\| = \tilde{O}(t^{1/2}).$$

Proof: Consider a fixed coordinate $j \in \{1, \ldots, n\}$. Since $\sum_{i=1}^t x_j^i$ is a sum of independent random variables each of which lies in the range $[-1, 1]$, we could simply apply Hoeffding's tail bound to obtain a bound on the probability that $\sum_{i=1}^t x_j^i$ deviates significantly from its expected value of zero. This straightforward approach, however, yields a weaker final bound than one which we can obtain with a little additional work by using conditional probabilities and expectations. By conditioning on the magnitude of each x_j^i, we can replace $[-1,1]$ with a much smaller interval and thereby obtain a tighter bound.

For any fixed coordinate $j \in \{1,\ldots,n\}$ and any $\alpha \geq 0$, we have that

$$\begin{aligned} \Pr_{x \in \mathcal{U}_{n-1}}[|x_j| \geq \alpha] &= 2 \cdot \frac{A_{n-2}}{A_{n-1}} \cdot \int_\alpha^1 \left(\sqrt{1-z^2}\right)^{n-3} dz \\ &\leq 2 \cdot \frac{A_{n-2}}{A_{n-1}} \cdot \left(\sqrt{1-\alpha^2}\right)^{n-3}. \end{aligned}$$

Consequently for $\alpha = \omega(\log^{1/2}(tn/\delta)/n^{1/2})$, we have

$$\Pr[|x_j| \geq \alpha] < \frac{\delta}{4tn}.$$

Let $\mathcal{C}$ denote the event that $|x_j^i| < \alpha$ for $1 \leq i \leq t$ (so $\Pr[\mathcal{C}] \geq 1 - \delta/4n$ by the union bound). Since by symmetry we have $E[\sum_{i=1}^t x_j^i \mid \mathcal{C}] = 0$, we can apply Hoeffding's tail bound to obtain the following: for any $\nu \geq 0$,

$$\Pr\left[\left|\frac{1}{t}\sum_{i=1}^t x_j^i\right| > \nu \mid \mathcal{C}\right] \leq 2 \cdot \exp\left(\frac{-t\nu^2}{2\alpha^2}\right).$$

Taking $\alpha = \tilde{\Theta}(n^{-1/2})$ and $\nu = \tilde{O}((nt)^{-1/2})$, this probability can be bounded by $\delta/4n$. Since $\Pr[\mathcal{C}] \geq 1 - \delta/4n$, this implies that with probability at least $1 - \delta/2n$ we have

$$\left(\sum_{i=1}^t x_j^i\right)^2 = \tilde{O}\left(\frac{t}{n}\right).$$

Using the union bound over $x_1, \ldots, x_n$, we find that with probability at least $1 - \delta/2$ we have

$$\left(\sum_{i=1}^t x_1^i\right)^2 + \cdots + \left(\sum_{i=1}^t x_n^i\right)^2 = \tilde{O}(t)$$

which proves the lemma. ∎

Now we use Lemma 6.4 to give an upper bound on the magnitude of the component of v which is orthogonal to u.

Lemma 6.5 *Let* $v = \texttt{Average}(MX, t)$ *and let* v' *be the component of* v *which is orthogonal to* u *(i.e.* $v' = v - (u \cdot v)u$*). Then with probability at least* $1 - \delta/2$ *we have that*

$$\|tv'\|^2 = \tilde{O}(t).$$

Proof: Without loss of generality we can suppose that the target vector u is $(1, 0, \ldots, 0)$, so our goal is to bound

$$t^2(v_2^2 + \cdots + v_n^2).$$

Recall that $tv = \sum_{i=1}^t b_i x^i$. Since each x^i is drawn from $\mathcal{U}_{n-1}$, the distribution of each x_j^i is symmetric around 0. For any $i \in \{1, \ldots, t\}$ and $j \in \{2, \ldots, n\}$ the distribution of b_i is independent of the distribution of x_j^i, so we have that the distribution of $b_i x_j^i$ is identical to that of x_j^i, and hence the distribution of $\sum_{i=1}^t b_i x_j^i$ is identical to that of $\sum_{i=1}^t x_j^i$ for $j = 2, \ldots, n$. Since

$$t^2 v_j^2 = \left(\sum_{i=1}^t b_i x_j^i\right)^2$$

for $j = 2, \ldots, n$, the distribution of $t^2(v_2^2 + \cdots + v_n^2)$ is identical to that of

$$\left(\sum_{i=1}^t x_2^i\right)^2 + \cdots + \left(\sum_{i=1}^t x_n^i\right)^2.$$

Lemma 6.4 implies that

$$\left(\sum_{i=1}^t x_2^i\right)^2 + \cdots + \left(\sum_{i=1}^t x_n^i\right)^2 = \tilde{O}(t)$$

with probability at least $1 - \delta/2$, so the lemma is proved. ■

6.4.3 Putting it All Together

We have almost reached our goal. By Lemma 6.3, if we take t sufficiently large, then with probability at least $1 - \delta/2$ we have

$$u \cdot v = \Omega\left(\frac{1 - 2\eta}{n^{1/2}}\right).$$

On the other hand, Lemma 6.5 tells us that

$$\|tv'\| = \tilde{O}(t^{1/2})$$

with probability at least $1-\delta/2$. Consequently, with probability $1-\delta$ we have

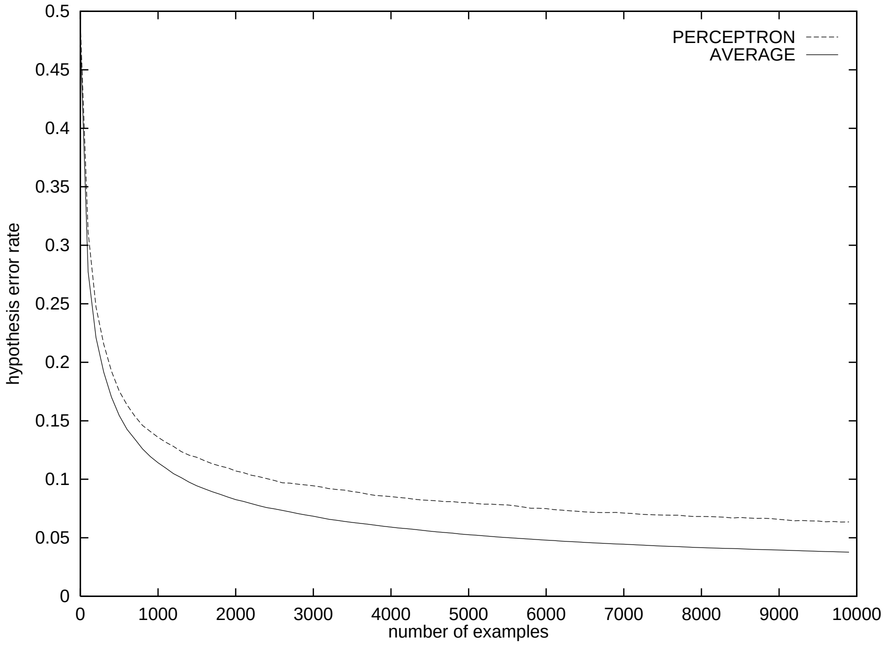
$$\frac{\|v'\|}{u\cdot v}=\tilde{O}\left(\frac{n^{1/2}}{t^{1/2}(1-2\eta)}\right).$$

Note that $\|v'\|/(u\cdot v)$ is the tangent of $\theta(u,v)$, the angle between u and v. Also, as mentioned in Section 6.2, the error of v under $\mathcal{U}_{n-1}$, which we denote by $error(v)$, is $\theta(u,v)/\pi$. Consequently, with probability $1-\delta$ we have

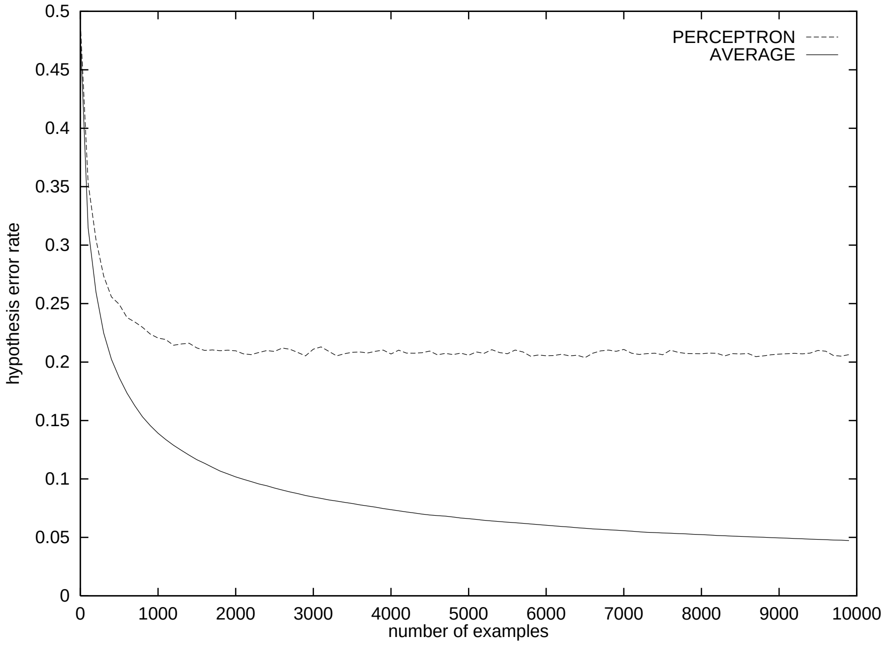
$$\begin{aligned} error(v) &= \frac{1}{\pi}\cdot\arctan\left(\frac{\|v'\|}{u\cdot v}\right) \\ &\leq \frac{\|v'\|}{(u\cdot v)\cdot\pi} \\ &= \tilde{O}\left(\frac{n^{1/2}}{t^{1/2}(1-2\eta)}\right) \end{aligned}$$

(the inequality holds because $\arctan(x)\leq x$ for $x\geq 0$). By choosing $t=\tilde{\Theta}(\frac{n}{\epsilon^2(1-2\eta)^2})$ we can satisfy all of the required conditions and obtain $error(v)<\epsilon$. Thus, we have proved the following theorem:

Theorem 6.6 *For some value of* $t=\tilde{\Theta}(\frac{n}{\epsilon^2(1-2\eta)^2})$, *the algorithm* `Average`$(MX,t)$ *is a PAC learning algorithm for the class of origin-centered halfspaces under* $\mathcal{U}_{n-1}$ *which uses* $\tilde{\Theta}(\frac{n}{\epsilon^2(1-2\eta)^2})$ *examples and runs in* $\tilde{\Theta}(\frac{n^2}{\epsilon^2(1-2\eta)^2})$ *time steps in the presence of monotonic noise.*



(a)



(b)

Figure 6.3: A performance comparison of `Average` and Perceptron.

Part (a) shows the error rates of `Average` and Perceptron on noise-free examples drawn from $\mathcal{U}_{n-1}$ with $n = 100$. Part (b) shows the error rates of `Average` and Perceptron on examples drawn from $\mathcal{U}_{n-1}$ with $n = 100$ and a classification noise rate of $\eta = 0.10$. Each plot represents an average of 100 independent runs of each algorithm.

Chapter 7

PAC Analogues of Perceptron and Winnow via Boosting the Margin

In Chapter 6 we saw that a simple geometric averaging technique can be used to obtain a surprisingly effective algorithm for learning origin-centered halfspaces under the uniform distribution. We now extend this averaging approach to obtain a new family of algorithms for learning linear threshold functions under much more general conditions. The new algorithms work by combining a generalized version of the `Average` algorithm with a powerful and well-studied learning theory technique known as *boosting*. We prove performance guarantees for our new algorithms which are remarkably similar to known bounds for online linear threshold learning algorithms including Perceptron and Winnow. This similarity strongly suggests that these well-studied online algorithms in some sense correspond to instances of boosting.

7.1 Introduction

7.1.1 The `Average` Algorithm Revisited

Recall that the `Average` algorithm described in Chapter 6 works in a very simple way. The first step of the algorithm is to draw a set of labeled examples and *normalize* each negative example by reflecting it through the origin. The algorithm then computes the geometric average of the positive examples and uses that average vector as the hypothesis linear threshold function. We saw that this simple algorithm is successful when the target concept is an origin-centered halfspace and the examples used for

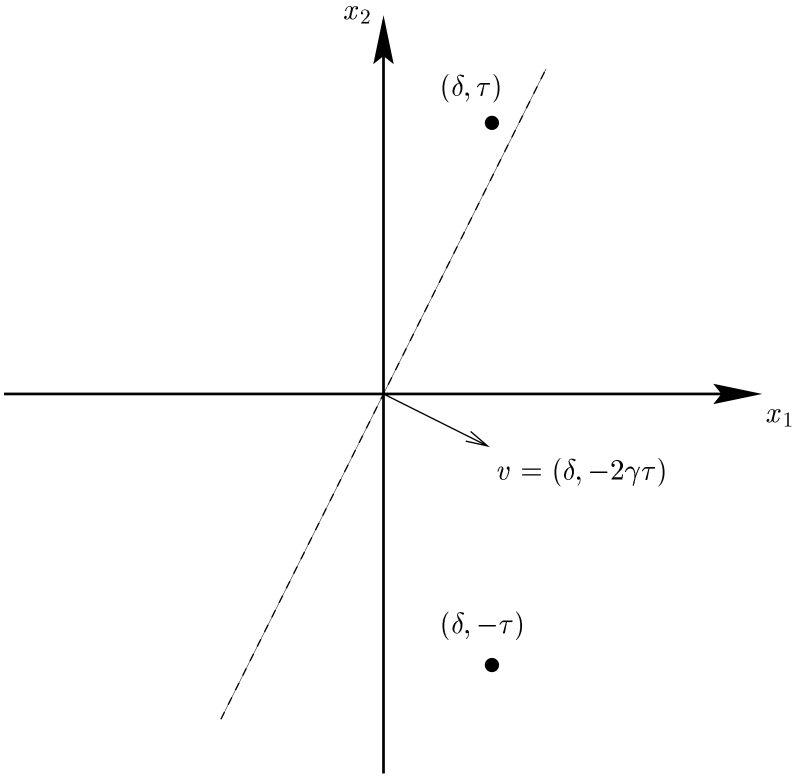


Figure 7.1: A worst-case data set for the `Average` algorithm. The target concept is defined by the halfspace $x_1 \geq 0$. The distribution $\mathcal{D}$ places weight $\frac{1}{2} - \gamma$ on the positive example (δ, τ) and places weight $\frac{1}{2} + \gamma$ on the positive example $(\delta, -\tau)$ where γ is a positive value satisfying $\gamma > \delta^2/(2\tau^2)$. The average vector under this distribution is $v = (\delta, -2\gamma\tau)$ which has error $\frac{1}{2} - \gamma$ under $\mathcal{D}$.

learning are drawn uniformly from the origin-centered unit sphere in $\Re^n$. The proof of correctness in Chapter 6 depends crucially on the symmetry of the uniform distribution which ensures that the average vector will tend to point in the exact same direction as the normal vector to the separating hyperplane.

A natural question is the following: what can be said about the performance of the `Average` algorithm if it is run under a nonuniform distribution? A moment's thought shows that the average vector under a nonuniform distribution need not point in the same direction as the vector corresponding to the target halfspace; moreover, even a slight deviation from this target vector can cause the resulting hypothesis to have a high error rate. Figure 7.1 gives an example of a distribution over $\Re^2$ for which the average vector points in nearly the same direction as the target vector, but the error rate of the average vector hypothesis is almost 50%.

7.1.2 The `Average` Algorithm Redeemed

The example of Figure 7.1 suggests that the `Average` algorithm can perform quite poorly for nonuniform distributions. However, we will see that in fact the `Average` algorithm *can* be used as the key component in an algorithm for learning linear threshold functions under nonuniform distributions. The example given in Figure 7.1 actually represents a worst-case scenario for the `Average` algorithm: as we show in Section 7.3, under suitable conditions the `Average` algorithm has the crucial property that the error rate of its hypothesis will be less than 1/2 for *any* probability distribution.

Algorithms which have this guarantee of outperforming a "random guess" are known as *weak learning algorithms.* Such algorithms were first discussed by Kearns and Valiant (Kearns and Valiant, 1994) and have since been studied intensively in learning theory. We show in Section 7.4 that by applying methods from boosting, it is possible to convert the `Average` algorithm into an effective, efficient algorithm for learning linear threshold functions to arbitrary accuracy.

7.1.3 Boosting-Based Linear Threshold Learning Algorithms

While this approach of boosting the `Average` algorithm is conceptually and algorithmically very different from the `Perceptron` algorithm, we establish performance bounds for the new algorithm which are remarkably similar to the bounds given by the Perceptron Convergence Theorem. We thus refer to our new boosting-based algorithm as a *PAC analogue* of Perceptron. Furthermore, we show that a simple variant of the `Average` algorithm can also be used as the weak learning component of a boosting-based algorithm for linear threshold functions, and we prove that the performance of this new algorithm is remarkably similar to that of Littlestone's Winnow algorithm. These similarities suggest a close relationship between boosting techniques and the online Perceptron and Winnow algorithms.

We give a unified analysis of our Perceptron and Winnow analogues which includes many other algorithms as well. Grove, Littlestone and Schuurmans (Grove *et al.*, 1997) have shown that Perceptron and (a version of) Winnow can be viewed as the $p = 2$ and $p \to \infty$ cases of a general online p-norm linear threshold learning algorithm, where $p \geq 2$ is any real number. We present PAC-model boosting-based analogues of these online p-norm algorithms for any value $2 \leq p \leq \infty$. The PAC-model Perceptron and Winnow analogues mentioned above are respectively the $p = 2$ and $p = \infty$ cases of this general algorithm.

The $p = \infty$ case of our algorithm can also be viewed as a generalization of Jackson and Craven's PAC-model algorithm for learning "sparse perceptrons" (Jackson and Craven, 1996). Their algorithm boosts using weak hypotheses which are single Boolean literals; this is similar to what the $p = \infty$ case of our algorithm does. Our analysis of the $p = \infty$ case generalizes their algorithm to deal with real-valued rather than Boolean input variables, thus achieving a goal stated by Jackson and Craven, and also yields a substantially stronger sample complexity bound than was established by Jackson and Craven.

7.1.4 Related Work

Several authors have previously studied linear threshold learning algorithms which work by combining weak predictors. Freund and Schapire have studied an algorithm which predicts using a weighted vote of the hypotheses which the Perceptron algorithm generates during its training phase (Freund and Schapire, 1998). The weight of each hypothesis in this vote is proportional to its survival time, i.e. the number of examples which elapse before it classifies an example incorrectly and causes the Perceptron algorithm to generate a new hypothesis. Freund and Schapire prove generalization error bounds on the resulting classifier which are similar to Vapnik's generalization error bounds for the "maximal margin" hyperplane (Vapnik, 1998). The Freund-Schapire algorithm differs from our approach in several ways: for one thing, their algorithm is unlike ours in that it does not use boosting to combine the weak predictors. Additionally, whereas our algorithm's final hypothesis is a single linear threshold function, their algorithm's final hypothesis is a depth-2 threshold circuit (a weighted vote over Perceptron hypotheses which are themselves linear threshold functions).

Ji and Ma have suggested that a random-search-and-test approach can be used to find weak classifier linear threshold functions for certain restricted halfspace learning problems (Ji and Ma, 1997). They propose combining these weak classifier linear threshold functions with a simple majority vote; thus, their approach also results in a final hypothesis which is a depth 2 threshold circuit.

Our approach is closest to that of Jackson and Craven, who use boosting to combine single literals into a strong hypothesis linear threshold function. As we show in Section 7.5, the $p = \infty$ case of our algorithm strengthens and generalizes their results.

The close similarity in performance bounds between our boosting-based algorithms and the online p-norm algorithms suggests a relationship between boosting and online learning. Freund and Schapire (Freund and Schapire, 1996) and Schapire (Schapire, 1999a) have investigated this relationship in the context of game theory.

7.2 Preliminaries

7.2.1 Geometric Preliminaries

For a point $x = (x_1, \ldots, x_n) \in \Re^n$ and $p \geq 1$ we write $\|x\|_p$ to denote the p-norm of x, namely

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

The ∞-norm of x is $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$. For $p, q \geq 1$ the q-norm is *dual* to the p-norm if $\frac{1}{p} + \frac{1}{q} = 1$; hence the 1-norm and the ∞-norm are dual to each other and the 2-norm is dual to itself. We use p and q to denote dual norms. The following facts are well known (e.g. (Taylor and Mann, 1972) pp. 203-204):

Hölder Inequality: $|u \cdot v| \leq \|u\|_p \|v\|_q$ for all $u, v \in \Re^n$ and $1 \leq p \leq \infty$.

Minkowski Inequality: $\|u + v\|_p \leq \|u\|_p + \|v\|_p$ for all $u, v \in \Re^n$ and $1 \leq p \leq \infty$.

Throughout this chapter the example space X is a subset of $\Re^n$. In this chapter we consider only origin-centered linear threshold functions, i.e. $f(x) = \text{sign}(u \cdot x)$ for some $u \in \Re^n$. While the standard definition of a linear threshold function allows a nonzero threshold (i.e. $f(x) = \text{sign}(u \cdot x - \theta)$ where θ can be any real number), it is easy to see that any linear threshold function of this more general form over n variables is equivalent to a linear threshold function with threshold 0 over $n + 1$ variables, so our new definition incurs no real loss of generality.

7.2.2 PAC Learning Linear Threshold Functions with Separation

The mistake bounds for both the Perceptron and Winnow algorithms depend on a "separation parameter" which gives a lower bound on the distance between any example and the hyperplane which defines the target halfspace. Similarly, our new algorithms also require that the examples used for learning must not lie too close to the separating hyperplane. Thus we will consider a slightly restricted version of the PAC learning model in which the domain of possible examples depends on the target concept being learned.

For $p \geq 2$ and $X \subset \Re^n$ we write $\|X\|_p$ to denote $\sup_{x \in X} \|x\|_p$. For a target vector $u \in \Re^n$ we use the symbol $\delta_{u,X}$ to denote the quantity

$$\delta_{u,X} \stackrel{\text{def}}{=} \inf_{x \in X} (u \cdot x)(\text{sign}(u \cdot x)),$$

which is a measure of the separation between examples in X and the hyperplane whose normal vector is u. We assume that $\|X\|_p < \infty$, i.e. the set X is bounded, and that $\delta_{u,X} > 0$, i.e. there is some nonzero lower bound on the separation between the hyperplane defined by u and the examples in X.

Let $EX(u, \mathcal{D})$ denote an example oracle which, when queried, provides a labeled example $\langle x, \text{sign}(u \cdot x) \rangle$ where x is drawn according to the distribution $\mathcal{D}$ over X. We say that an algorithm A is a *strong learning algorithm for u on X* if it satisfies the following condition: there is a function $m(\epsilon, \delta, u, X)$ such that for any distribution $\mathcal{D}$ over X, for all $0 < \epsilon, \delta < 1$, algorithm A makes at most $m(\epsilon, \delta, u, X)$ calls to $EX(u, \mathcal{D})$, and with probability at least $1 - \delta$ algorithm A outputs a hypothesis $h : X \to \{-1, 1\}$ such that $\Pr_{x \in \mathcal{D}}[h(x) \neq \text{sign}(u \cdot x)] \leq \epsilon$. We say that such a hypothesis h is an *ϵ-accurate hypothesis for u under $\mathcal{D}$* and that the function $m(\epsilon, \delta, u, X)$ is the sample complexity of algorithm A.

As our main result in this chapter we will describe a strong learning algorithm and carefully analyze its sample complexity. To do this we must consider algorithms which do not satisfy the strong learning property but are still capable of generating hypotheses that have some slight advantage over random guessing; such algorithms are known as *weak learning algorithms.* Let

$$S = \langle x^1, \mathrm{sign}(u \cdot x^1)\rangle, \ldots, \langle x^m, \mathrm{sign}(u \cdot x^m)\rangle$$

be a finite sequence of labeled examples from X and let $\mathcal{D}$ be a distribution over S. For $0 < \gamma < 1/2$, we say that $h : X \to [-1, 1]$ is a $(1/2 - \gamma)$-*approximator for u under $\mathcal{D}$* if

$$\frac{1}{2}\sum_{i=1}^{m} \mathcal{D}(x^i) \cdot |h(x^i) - \ \mathrm{sign}(u \cdot x^i)| \leq \frac{1}{2} - \gamma. \tag{7.1}$$

We say that an algorithm A is a $(1/2 - \gamma)$-*weak learning algorithm for u under $\mathcal{D}$* if the following condition holds: for any finite set S as described above and any distribution $\mathcal{D}$ on S, if A is given $\mathcal{D}$ and S as input then A outputs a hypothesis $h : X \to [-1, 1]$ which is a $(1/2 - \gamma)$-approximator for u under $\mathcal{D}$. Thus for our purposes a weak learning algorithm is one which can always find a hypothesis that outperforms random guessing on a fixed sample.

7.2.3 The Online p-norm Algorithms

In the online learning model, learning takes place over a sequence of trials. Throughout the learning process the learner maintains a hypothesis h which maps X to $\{-1, 1\}$. Each trial proceeds as follows: upon receiving an example $x \in X$ the learning algorithm outputs its prediction $\hat{y} = h(x)$ of the associated label y. The learning algorithm is then given the true label $y \in \{-1, 1\}$ and the algorithm can update its hypothesis h based on this new information before the next trial begins. The performance of an online learning algorithm on an example sequence is measured by the number of prediction mistakes which the algorithm makes.

Grove, Littlestone and Schuurmans (Grove *et al.*, 1997) and Gentile and Littlestone (Gentile and Littlestone, 1999) have studied a family of online algorithms for learning linear threshold functions (see Figure 1). We refer to this algorithm, which is parameterized by a real value $p \geq 2$, as the *online p-norm algorithm.* Like the well-known Perceptron algorithm, the online p-norm algorithm updates its hypothesis by making an additive change to a weight vector z. However, as shown in steps 4-5 of Figure 1, the p-norm algorithm does not use the z vector directly for prediction but

Input: real number $p \geq 2$
initial weight vector $z^0 = (z_1^0, \ldots, z_n^0) \in \Re^n$
real number $a > 0$

Output: hypothesis $h(x)$

set $t = 0$
while examples are available **do**
  get unlabeled example x^t
  for all $i = 1, \ldots, n$ **set** $w_i^t = \text{sign}(z_i^t)|z_i^t|^{p-1}$
  predict $\hat{y}_t = \text{sign}(w^t \cdot x^t)$
  get label $y_t \in \{-1, +1\}$
  for all $i = 1, \ldots, n$ **set** $z_i^{t+1} = z_i^t + a(y_t - \hat{y}_t)x_i^t$
  set $t = t + 1$
enddo

Figure 7.2: The online p-norm algorithm.

rather predicts using a vector w which is a transformed version of the z vector, namely $w_i = \text{sign}(z_i)|z_i|^{p-1}$ for all $i = 1, \ldots, n$. Note that when $p = 2$ we have $z = w$ and hence the online 2-norm algorithm is identical to the Perceptron algorithm. Grove *et al.* show that as $p \to \infty$ the online p-norm algorithm approaches a version of the Winnow algorithm. More precisely, the following theorem gives mistake bounds for the online p-norm algorithms:

Theorem 7.2 (Grove *et al.*, 1997) *Let $S = \langle x^1, y_1 \rangle, \ldots, \langle x^m, y_m \rangle$ be a sequence of labeled examples where $x \in X$ and $y = sign(u \cdot x)$ for every example $\langle x, y \rangle \in S$.*

(a) For any $2 \leq p < \infty$ and any $a > 0$, if the online p-norm algorithm is invoked with input parameters $(p, z^0 = (0, \ldots, 0), a)$, then the mistake bound on the example sequence S is at most

$$\frac{(p-1)\|u\|_q^2 \|X\|_p^2}{\delta_{u,X}^2}.$$

(b) For any $2 \leq p < \infty$, if z^0 satisfies $u \cdot z^0 > 0$ and $a = \frac{\delta_{u,X}\|z^0\|_p^2}{(p-1)u \cdot z^0 \|X\|_p^2}$, then the

mistake bound on S is at most

$$\frac{(p-1)\|u\|_q^2\|X\|_p^2}{\delta_{u,X}^2}\left(1-\left(\frac{u\cdot z^0}{\|u\|_q\|z^0\|_p}\right)^2\right).$$

(c) *Let $z^0=(1,\ldots,1)$ and suppose that $u_i>0$ for $i=1,\ldots,n$. If $p\to\infty$ and a is as described in part (b), then the mistake bound given in (b) converges to*

$$\frac{2\|u\|_1^2\|X\|_\infty^2}{\delta_{u,X}^2}\left(\log n+\sum_{i=1}^{n}\frac{u_i}{\|u\|_1}\log\frac{u_i}{\|u\|_1}\right).$$

When $p=2$ part (a) of this theorem corresponds exactly to the Perceptron Convergence Theorem (Theorem 5.2 of Chapter 5). Furthermore, part (c) corresponds closely to the mistake bound proved by Littlestone for the Winnow algorithm (Theorem 5.3 of Chapter 5).

7.2.4 From Online to PAC Learning

As described in Chapter 2, several generic procedures are known for automatically converting online learning algorithms into PAC-model algorithms. We recall Theorem 2.2, due to Littlestone, which gives a strong bound on one such conversion:

Theorem 2.2 *Let A be an online learning algorithm which changes its hypothesis only when it makes a mistake and which has a mistake bound of M for concept class C. Then there is a PAC-model learning algorithm A' for C which has sample complexity*

$$O\left(\frac{1}{\epsilon}\left(\log\frac{1}{\delta}+M\right)\right).$$

By applying Theorem 2.2 to Theorem 7.2 one can obtain sample complexity bounds on a generic PAC-model conversion of the online p-norm algorithm. We now describe a completely different PAC-model learning algorithm which has remarkably similar sample complexity bounds.

Input: real number $p \geq 2$
sequence $S = \langle x^1, y_1 \rangle, \ldots, \langle x^m, y_m \rangle$ of labeled examples with $x^i \in \Re^n$, $y_i \in \{-1, 1\}$
distribution $\mathcal{D}$ over S

Output: real valued hypothesis $h(x)$

1. **set** $z = \sum_{j=1}^{m} \mathcal{D}(x^j) y_j x^j$
2. **for all** $i = 1, \ldots, n$ **set** $w_i = \text{sign}(z_i)|z_i|^{p-1}$
3. **return** hypothesis $h(x) \equiv \frac{w \cdot x}{\|w\|_q \|X\|_p}$

Figure 7.3: The p-norm weak learning algorithm `WLA`.

7.3 A PAC Model p-norm Weak Learning Algorithm

Our p-norm weak learning algorithm is motivated by the following simple idea: Suppose that $S = \langle x^1, y_1 \rangle, \ldots, \langle x^m, y_m \rangle$ is a collection of labeled examples where $y_i = \text{sign}(u \cdot x^i)$ for each $i = 1, \ldots, m$. Now imagine replacing each negative example $\langle x^i, -1 \rangle$ in S by the equivalent positive example $\langle -x^i, 1 \rangle$ to obtain a new collection S' of normalized examples. Let $z \in \Re^n$ be the average location of an example in S', i.e. z is the "center of mass" of the point cloud S'. Since each example in S' is positive, each example in S' must lie on the same side of the hyperplane $u \cdot x = 0$ as the vector u, so clearly z must also lie on this side of the hyperplane. One might even hope that z, or some related vector, points in approximately the same direction as the vector u.

Our p-norm weak learning algorithm, which we call `WLA`, is presented in Figure 7.2. The vector z is the "center of mass" of the normalized points with respect to the probability distribution $\mathcal{D}$ which is part of the input to `WLA` (so running `WLA` repeatedly on the same data set S but with different distributions $\mathcal{D}$ can yield different values for z). Like the online p-norm algorithm, the `WLA` algorithm transforms the vector z to a vector w using the mapping $w_i = \text{sign}(z_i)|z_i|^{p-1}$. The real-valued `WLA` hypothesis is a scaled version of the linear functional defined by the vector w. The following theorem establishes that this simple algorithm is in fact a weak learner:

Theorem 7.3 `WLA` *is a* $(1/2-\gamma)$*-weak learning algorithm for* u *under* $\mathcal{D}$ *for* $\gamma = \frac{\delta_{u,X}}{2\|X\|_p\|u\|_q}$.

Proof: Let $S = \langle x^1, y_1\rangle, \ldots, \langle x^m, y_m\rangle$ be a sequence of labeled examples where $x \in X$ and $y = \text{sign}(u \cdot x)$ for every pair $\langle x, y\rangle \in S$, and let $\mathcal{D}$ be a distribution over S. We will show that the hypothesis h which `WLA`$(p, S, \mathcal{D})$ returns is a $(1/2-\gamma)$-approximator for u under $\mathcal{D}$.

To see that h maps X into $[-1,1]$, note that for any $x \in X$ Hölder's inequality implies

$$|h(x)| = \frac{|w \cdot x|}{\|w\|_q\|X\|_p} \leq \frac{\|w\|_q\|x\|_p}{\|w\|_q\|X\|_p} \leq \frac{\|w\|_q\|X\|_p}{\|w\|_q\|X\|_p} = 1.$$

Now we show that inequality (7.1) from Section 7.2.2 holds. Since $h(x^j) \in [-1,1]$ and $y_j \in \{-1,1\}$ we have that

$$|h(x^j) - y_j| = 1 - y_j h(x^j),$$

and thus

$$\begin{aligned}
\frac{1}{2}\sum_{j=1}^{m} \mathcal{D}(x^j)|h(x^j) - y_j| &= \frac{1}{2}\sum_{j=1}^{m} \mathcal{D}(x^j)(1 - y_j h(x^j)) \\
&= \frac{1}{2} - \frac{1}{2\|X\|_p}\left(\frac{\sum_{j=1}^{m} \mathcal{D}(x^j) y_j (w \cdot x^j)}{\|w\|_q}\right).
\end{aligned}$$

Thus it suffices to show that

$$\frac{\sum_{j=1}^{m} \mathcal{D}(x^j) y_j (w \cdot x^j)}{\|w\|_q} \geq \frac{\delta_{u,X}}{\|u\|_q}.$$

We first note that

$$\begin{aligned}
\sum_{j=1}^{m} \mathcal{D}(x^j) y_j (w \cdot x^j) &= w \cdot \left(\sum_{j=1}^{m} \mathcal{D}(x^j) y_j x^j\right) \\
&= w \cdot z \\
&= \sum_{j=1}^{m} |z_j|^p \\
&= \|z\|_p^p
\end{aligned}$$

and hence the left-hand side of the desired inequality equals $\|z\|_p^p/\|w\|_q$. We also have

$$\begin{aligned}\|w\|_q &= \left(\sum_{i=1}^n \left(|z_i|^{p-1}\right)^q\right)^{1/q} \\ &= \left(\sum_{i=1}^n |z_i|^p\right)^{1/q} \\ &= \|z\|_p^{p/q},\end{aligned}$$

where in the second equality we used the fact that $(p-1)q = p$. Consequently the left-hand side can be further simplified to $\|z\|_p^p/\|w\|_q = \|z\|_p^{p-p/q} = \|z\|_p$, and thus our goal is to show that $\|z\|_p \geq \delta_{u,X}/\|u\|_q$. Since $\delta_{u,X} \leq u \cdot (y_j x^j)$ for $j = 1, \ldots, m$, we have

$$\begin{aligned}\delta_{u,X} \leq \sum_{j=1}^m \mathcal{D}(x^j) u \cdot (y_j x^j) &= u \cdot \left(\sum_{j=1}^m \mathcal{D}(x^j) y_j x^j\right) \\ &= u \cdot z \\ &\leq \|u\|_q \|z\|_p,\end{aligned}$$

where the last line follows from the Hölder inequality, and the theorem is proved. ■

Thus, the simple `WLA` algorithm can serve as a weak learning algorithm for the halfspace learning problem. In the next section we use techniques from boosting and large margin classification to obtain a strong learning algorithm which has small sample complexity.

7.4 From Weak to Strong Learning

7.4.1 Boosting to Achieve High Accuracy

In (Kearns and Valiant, 1994) the following question was posed: is every concept class which is efficiently weakly learnable also efficiently strongly learnable? In an important result this question was answered in the affirmative by Schapire (Schapire, 1990), who gave a *boosting* algorithm which can be used to efficiently transform any weak learning algorithm into a strong learning algorithm. In several subsequent papers various improved boosting algorithms were developed (Freund, 1990; Freund, 1992; Freund, 1995; Freund and Schapire, 1997); we use the `AdaBoost` algorithm from

Input: real numbers $0 < \gamma, \mu < \frac{1}{2}$
sequence $S = \langle x^1, y_1 \rangle, \ldots, \langle x^m, y_m \rangle$ of labeled examples
weak learning algorithm `WL`: $S \to [-1, 1]$

Output: hypothesis $h(x)$

1. **set** $T = \frac{1}{2\gamma^2} \log \frac{1}{\mu}$
2. **for all** $i = 1, \ldots, m$ **set** $\mathcal{D}^1(x^i) = \frac{1}{m}$
3. **for** $t = 1, \ldots, T$ **do**
4. let h_t be the output of $\mathtt{WL}(\mathcal{D}^t, S)$
5. **set** $\epsilon_t = \frac{1}{2} \sum_{i=1}^m \mathcal{D}^t(x^i) |h_t(x^i) - y_i|$
6. **set** $\alpha_t = \frac{1}{2} \ln((1 - \epsilon_t)/\epsilon_t)$
7. **for all** $i = 1, \ldots, m$ **set**

$$\mathcal{D}^{t+1}(x^i) = \frac{\mathcal{D}^t(x^i) \exp(-y_i \alpha_t h_t(x^i))}{Z_t}$$

where $Z_t = \sum_{i=1}^m \mathcal{D}^t(x^i) \exp(-y_i \alpha_t h_t(x^i))$ is a normalizing factor which ensures that $\mathcal{D}^{t+1}$ is a distribution

9. **enddo**
10. **output** as final hypothesis $h(x) \equiv \text{sign}(f(x))$ where

$$f(x) = \frac{\sum_{t=1}^T \alpha_t h_t(x)}{\sum_{t=1}^T \alpha_t}.$$

Figure 7.4: The `AdaBoost` algorithm.

(Freund and Schapire, 1997) which is shown in Figure 3.

Our notation for the algorithm is similar to that of (Schapire *et al.*, 1998; Schapire and Singer, 1998). The input to `AdaBoost` is a sequence $S = \langle x^1, y_1 \rangle, \ldots, \langle x^m, y_m \rangle$ of m labeled examples, a weak learning algorithm `WL`, and two parameters $0 < \gamma, \mu < 1/2$. Given a distribution $\mathcal{D}^t$ over a data set S, algorithm `WL` outputs a hypothesis h_t which maps S to $[-1, 1]$. `AdaBoost` works in a sequence of stages, where in stage t it generates a distribution $\mathcal{D}^t$ and runs `WL` to obtain a hypothesis h_t. The final `AdaBoost` hypothesis is a linear threshold function over the hypotheses $h_1, \ldots, h_T$.

Freund and Schapire have shown that if the algorithm `WL` is a $(1/2 - \gamma)$-weak learning algorithm, i.e. each call of `WL` in `AdaBoost` generates a hypothesis h_t such that ϵ_t (as defined in line 5) is at most $1/2 - \gamma$, then the fraction of examples in S which are misclassified by the final hypothesis h is at most μ. Given this result, one straightforward way to obtain a strong learning algorithm for our halfspace learning problem is to draw a sufficiently large (as specified below) sample S from the example oracle $EX(u, \mathcal{D})$ and run `AdaBoost` on S using `WLA` as the weak learning algorithm, γ as given in Theorem 7.3, and $\mu < 1/|S|$. This choice of μ ensures that `AdaBoost`'s final hypothesis makes no errors on S; moreover, since each hypothesis generated by `WLA` is of the form $h_t(x) = v^t \cdot x$ for some $v^t \in \Re^n$, `AdaBoost`'s final hypothesis will be of the form $h(x) = \text{sign}(v \cdot x)$ for some $v \in \Re^n$. Using the fact that the Vapnik-Chervonenkis dimension of the class of zero-threshold linear threshold functions over $\Re^n$ is n, the well-known theorem of Blumer, Ehrenfeucht, Haussler and Warmuth (Blumer *et al.*, 1989) implies that with probability at least $1 - \delta$ the final hypothesis h is an ϵ-accurate hypothesis for u under $\mathcal{D}$ provided that $|S| \geq c(\epsilon^{-1}(n \log(\epsilon^{-1}) + \log(\delta^{-1})))$ for some constant $c > 0$.

This analysis, though attractively simple, yields a rather crude bound on sample complexity which does not depend on the particulars of the learning problem (i.e. u and X). In the rest of this section we use recent results on `AdaBoost`'s ability to generate a large-margin classifier and the generalization ability of large-margin classifiers to give a much tighter bound on sample complexity for this learning algorithm.

7.4.2 Boosting Real Valued Hypotheses to Achieve a Large Margin

Suppose that $h : X \rightarrow \{-1, 1\}$ is a classifier of the form $h(x) = \text{sign}(f(x))$ where f maps X into $[-1, 1]$. We say that the *margin* of h on a labeled example $\langle x, y \rangle$ is $yf(x)$; note that this quantity is nonnegative if and only if h correctly predicts the label y associated with x. The magnitude of the margin can be viewed as a measure of the confidence with which the classifier makes its prediction on x.

The following theorem, which is an extension of Theorem 5 from (Schapire *et al.*, 1998), shows that `AdaBoost` can be used in conjunction with a real-valued weak learner to obtain large-margin hypotheses.

Theorem 7.4 *Suppose that* `AdaBoost` *is run on an example sequence* $S = \langle x^1, y_1 \rangle, \ldots,$ $\langle x^m, y_m \rangle$ *using a weak learning algorithm* `WL`: $S \rightarrow [-1, 1]$. *Then for any value* $\theta \geq 0$ *we have*

$$\frac{|\{i \in \{1, 2, \ldots, m\} : y_i f(x^i) \leq \theta\}|}{m} \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}}.$$

Proof: The proof combines ideas from (Schapire *et al.*, 1998), where it is shown that `AdaBoost` with binary valued hypotheses generates a large margin classifier, and (Schapire and Singer, 1998), where an analysis is given for `AdaBoost`'s classification error with real valued hypotheses. As in Theorem 5 of (Schapire *et al.*, 1998), if $y_i f(x^i) \leq \theta$ then

$$y_i \sum_{t=1}^{T} \alpha_t h_t(x^i) \leq \theta \sum_{t=1}^{T} \alpha_t$$

which implies that

$$\exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x^i) + \theta \sum_{t=1}^{T} \alpha_t\right) \geq 1.$$

Following (Schapire *et al.*, 1998), we thus have

$$\begin{aligned}
\frac{|\{i \in \{1, 2, \ldots, m\} : y_i f(x^i) \leq \theta\}|}{m} &\leq \sum_{i=1}^{m} \frac{1}{m} \cdot \left[\exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x^i) + \theta \sum_{t=1}^{T} \alpha_t\right)\right] \\
&= \frac{\exp\left(\theta \sum_{t=1}^{T} \alpha_t\right)}{m} \sum_{i=1}^{m} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x^i)\right)
\end{aligned}$$

$$
\begin{aligned}
&= \exp\left(\theta \sum_{t=1}^{T} \alpha_t\right) \left(\prod_{t=1}^{T} Z_t\right) \sum_{i=1}^{m} \mathcal{D}^{T+1}(x^i) \\
&= \exp\left(\theta \sum_{t=1}^{T} \alpha_t\right) \left(\prod_{t=1}^{T} Z_t\right) \qquad (7.2)
\end{aligned}
$$

where the second equality follows from the definition of $\mathcal{D}^{t+1}$ and the final equality is because $\mathcal{D}^{T+1}$ is a distribution and hence sums to 1. Our goal is thus to bound the right side of (7.2).

If we let

$$r_t = \sum_{i=1}^{m} \mathcal{D}^t(x^i) y_i h_t(x^i)$$

then using the fact that

$$|h(x^j) - y_j| = 1 - y_j h(x^j)$$

we find that $\epsilon_t = \frac{1-r_t}{2}$. Substituting into the definition of α_t we obtain

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1+r_t}{1-r_t}\right).$$

Following (Schapire and Singer, 1998) for simplicity of notation we now fix t and let $u_i = y_i h_t(x^i)$, $Z = Z_t$, $\mathcal{D} = \mathcal{D}^t$, $\epsilon = \epsilon_t$, $r = r_t$, and $\alpha = \alpha_t$. A simple convexity argument shows that

$$e^{-\alpha u} \leq \frac{1+u}{2} e^{-\alpha} + \frac{1-u}{2} e^{\alpha}$$

for any $\alpha \in \Re$ and any $u \in [-1, 1]$. Since u_i always lies in the interval $[-1, 1]$, we can apply this inequality to obtain

$$
\begin{aligned}
Z &= \sum_{i=1}^{m} \mathcal{D}(x^i) e^{-\alpha u_i} \\
&\leq \sum_{i=1}^{m} \mathcal{D}(x^i) \left(\frac{1+u_i}{2} e^{-\alpha} + \frac{1-u_i}{2} e^{\alpha}\right). \qquad (7.3)
\end{aligned}
$$

As in Section 3.5 of (Schapire and Singer, 1998), substituting α into inequality (7.3) yields

$$
\begin{aligned}
Z_t &\leq \sqrt{1 - r_t^2} \\
&= \sqrt{1 - (1 - 2\epsilon_t)^2} \\
&= 2\sqrt{\epsilon_t(1 - \epsilon_t)}. \qquad (7.4)
\end{aligned}
$$

Substituting inequality (7.4) into inequality (7.2) and using the definition of α_t yields the desired bound of the theorem. ■

The results of Section 7.3 imply that if `WLA` is used as the weak learning algorithm in `AdaBoost`, then the value ϵ_t will always be at most $1/2 - \gamma$, and the upper bound of Theorem 7.4 becomes $((1-2\gamma)^{1-\theta}(1+2\gamma)^{1+\theta})^{T/2}$. The following technical lemma is useful:

Lemma 7.5 $(1-4x)^{1-x}(1+4x)^{1+x} \le 1-4x^2$ *for* $0 \le x \le 1/4$.

Proof: Using a simple convexity argument it can be verified that $\alpha^r \le 1-(1-\alpha)r$ for any $\alpha \ge 0$ and any $0 \le r \le 1$. This inequality implies that $(1-4x)^{1-x} \le 1-4x+4x^2$ and $(1+4x)^x \le 1+4x^2$, so consequently

$$(1-4x)^{1-x}(1+4x)^{1+x} \le (1-4x+4x^2)(1+4x)(1+4x^2),$$

which is at most $1-4x^2$ for $0 \le x \le 1/4$. ■

If we set $\theta = \gamma/2$ and apply Lemma 7.5 with $x = \theta$, the upper bound of Theorem 7.4 becomes $(1-\gamma^2)^{T/2}$ and we obtain the following:

Corollary 7.6 *If* `AdaBoost` *is run on a sequence S of labeled examples drawn from $EX(u, \mathcal{D})$ using* `WLA` *as the weak learner, γ as defined in Theorem 7.3 and $\mu < 1/|S|^4$, then the hypothesis h which* `AdaBoost` *generates will have margin at least $\gamma/2$ on every example in S.*

Proof: The bound on μ causes T to be greater than $\frac{2}{\gamma^2}\log\frac{1}{|S|}$, and consequently the upper bound of Theorem 7.4 is less than $1/|S|$. ■

Corollary 7.6 shows that a judicious choice of parameters for `AdaBoost` enables our boosting-based algorithm to both run efficiently and generate a final hypothesis which has a margin of at least $\gamma/2$ on every example in the training set. In the next subsection we use Corollary 7.6 and techniques from the theory of large margin classification to establish a bound on the generalization error of this hypothesis in terms of the sample size m.

7.4.3 Large Margins and Generalization Error

Let $\mathcal{F}$ be a collection of real-valued functions on a set X. A finite set $\{x^1, \ldots, x^k\} \subseteq X$ is said to be *ξ-shattered by $\mathcal{F}$* if there are real numbers $r_1, \ldots, r_k$ such that for all $b = (b_1, \ldots, b_k) \in \{-1, 1\}^k$, there is a function $f_b \in \mathcal{F}$ such that for all $i = 1, \ldots, k$

$$f_b(x^i) \begin{cases} \geq r_i + \xi & \text{if } \ b_i = 1 \\ \leq r_i - \xi & \text{if } \ b_i = -1. \end{cases}$$

For $\xi \geq 0$, the *fat-shattering dimension of $\mathcal{F}$ at scale ξ*, denoted $\mathrm{fat}_{\mathcal{F}}(\xi)$, is the size of the largest set which is ξ-shattered by $\mathcal{F}$, if this is finite, and infinity otherwise. The fat-shattering dimension is useful for us because of the following theorem from (Bartlett and Shawe-Taylor, 1999):

Theorem 7.7 *Let $\mathcal{F}$ be a collection of real-valued functions on X and let $\mathcal{D}$ be a distribution over $X \times \{-1, 1\}$. Let $S = \langle x^1, y_1\rangle, \ldots, \langle x^m, y_m\rangle$ be a sequence of labeled examples drawn from $\mathcal{D}$. With probability at least $1 - \delta$ over the choice of S, if a classifier $h(x) \equiv sign(f(x))$ with $f \in \mathcal{F}$ has margin at least $\xi > 0$ on every example in S, then*

$$\Pr_{(x,y)\in\mathcal{D}}[h(x) \neq y] \leq \frac{2}{m}\left(d \log \frac{8em}{d} \log(32m) + \log \frac{8m}{\delta}\right),$$

where $d = fat_{\mathcal{F}}(\xi/16)$.

As noted in Section 7.4.1, the final hypothesis h which `AdaBoost` outputs will be of the form $h(x) = \mathrm{sign}(f(x))$ with $f(x) = v \cdot x$ for some $v \in \Re^n$. Furthermore, since each invocation of `WLA` generates a hypothesis of the form $h_t(x) = v^t \cdot x$ with $\|v^t\|_q \leq \frac{1}{\|X\|_p}$, Minkowski's inequality implies that the vector v must satisfy $\|v\|_q \leq \frac{1}{\|X\|_p}$. We thus consider the class of functions

$$\mathcal{F} = \left\{x \mapsto v \cdot x : \|v\|_q \leq \frac{1}{\|X\|_p}, \|x\|_p \leq \|X\|_p\right\}. \tag{7.5}$$

If we can bound $\mathrm{fat}_{\mathcal{F}}(\xi)$, then given any sample size m, Theorem 7.7 immediately yields a corresponding bound on $\Pr_{x\in\mathcal{D}}[h(x) \neq \mathrm{sign}(u \cdot x)]$ for our halfspace learning problem. The following theorem, which is an extension of Theorem 1.6 from (Bartlett and Shawe-Taylor, 1999), gives the desired bound on $\mathrm{fat}_{\mathcal{F}}(\xi)$.

Theorem 7.8 *Let X be a bounded region in $\Re^n$ and let $\mathcal{F}$ be the class of functions on X defined in (7.5) above. Then $fat_{\mathcal{F}}(\xi) \leq \frac{2\log 4n}{\xi^2}$.*

Theorem 7.8 follows immediately upon combining the inequalities proved in the following two lemmas.

Lemma 7.9 *If the set $\{x^1, \ldots, x^d\}$ is ξ-shattered by $\mathcal{F}$ then every $b = (b_1, \ldots, b_d) \in \{-1, 1\}^d$ satisfies*

$$\left\| \sum_{i=1}^{d} b_i x^i \right\|_p \geq \xi d \|X\|_p.$$

Proof: Suppose that $\{x^1, \ldots, x^d\}$ is ξ-shattered by $\mathcal{F}$ as witnessed by the real numbers $r_1, \ldots, r_d$. Then for every $b = (b_1, \ldots, b_d) \in \{-1, 1\}^d$, there is a vector $v_b \in \Re^n$ with $\|v_b\|_q \leq \frac{1}{\|X\|_p}$ such that $b_i(v_b \cdot x^i - r_i) \geq \xi$ for $i = 1, \ldots, d$. Summing these d inequalities and rearranging, we obtain

$$v_b \cdot \left(\sum_{i=1}^{d} b_i x^i \right) \geq \xi d + \sum_{i=1}^{d} b_i r_i. \tag{7.6}$$

There are two cases to consider. Case 1 is if $\sum_{i=1}^{d} b_i r_i \geq 0$; if this is true, we have

$$\begin{aligned}
\frac{1}{\|X\|_p} \cdot \left\| \sum_{i=1}^{d} b_i x^i \right\|_p &\geq \|v_b\|_q \left\| \sum_{i=1}^{d} b_i x^i \right\|_p \\
&\geq v_b \cdot \left(\sum_{i=1}^{d} b_i x^i \right) \\
&\geq \xi d.
\end{aligned}$$

Here the first inequality is by the definition of $\mathcal{F}$, the second inequality is Hölder's, and the third is from inequality (7.6). This yields the desired inequality $\| \sum_{i=1}^{d} b_i x^i \|_p \geq \xi d \|X\|_p$.

In the second case, $\sum_{i=1}^{d} b_i r_i < 0$. If this is the case then let $c = (c_1, \ldots, c_d) = (-b_1, \ldots, -b_d)$. We then have $\sum_{i=1}^{d} c_i r_i > 0$, so Case 1 implies that $\| \sum_{i=1}^{d} c_i x^i \|_p \geq \xi d \|X\|_p$, and the lemma follows since

$$\left\| \sum_{i=1}^{d} c_i x^i \right\|_p = \left\| - \sum_{i=1}^{d} b_i x^i \right\|_p = \left\| \sum_{i=1}^{d} b_i x^i \right\|_p.$$

■

Lemma 7.10 *For any set $\{x^1, \ldots, x^d\}$ with each $\|x^i\|_p \leq \|X\|_p$, if $p \geq 2$ then there is some $b = (b_1, \ldots, b_d) \in \{-1, 1\}^d$ such that $\left\| \sum_{i=1}^{d} b_i x^i \right\|_p \leq \sqrt{2d \log 4n} \cdot \|X\|_p$.*

Proof: The proof uses the probabilistic method. We consider the random variable $z = \sum_{i=1}^{d} b_i x^i$ where $(b_1, \ldots, b_d)$ is uniformly distributed over $\{-1, 1\}^d$. For any coordinate $j \in \{1, \ldots, n\}$ we have $z_j = \sum_{i=1}^{d} b_i x_j^i$ and hence $E[z_j] = 0$. Let $Y_j = |x_j^1|^2 + \cdots + |x_j^d|^2$; Hoeffding's bound on sums of independent random variables implies that for any $t > 0$ we have

$$\Pr[|z_j| > t] \leq 2 \exp\left(\frac{-t^2}{2Y_j}\right).$$

As a consequence, taking $t = \sqrt{2Y_j \log 4n}$ we have that $\Pr[|z_j| \geq t] \leq 1/2n$. Using the union bound across $j = 1, 2, \ldots, n$, we have that with probability at least $1/2$ every coordinate z_j of z satisfies $|z_j| < \sqrt{2Y_j \log 4n}$, and hence

$$\begin{aligned}
\|z\|_p &= \left(\sum_{j=1}^{n} |z_j|^p\right)^{1/p} \\
&\leq \left(\sum_{j=1}^{n} \left(\sqrt{2Y_j \log 4n}\right)^p\right)^{1/p} \\
&= \sqrt{2 \log 4n} \cdot \left(\left(\sum_{j=1}^{n} \left[|x_j^1|^2 + \cdots + |x_j^d|^2\right]^{p/2}\right)^{2/p}\right)^{1/2} \qquad (7.7)
\end{aligned}$$

Since $p \geq 2$, we have $p/2 \geq 1$ and hence Minkowski's inequality implies that

$$\begin{aligned}
\left(\sum_{j=1}^{n} \left[|x_j^1|^2 + \cdots + |x_j^d|^2\right]^{p/2}\right)^{2/p} &\leq \left[\sum_{j=1}^{n} |x_j^1|^{2p/2}\right]^{2/p} + \cdots + \left[\sum_{j=1}^{n} |x_j^d|^{2p/2}\right]^{2/p} \\
&= \|x^1\|_p^2 + \cdots + \|x^d\|_p^2 \\
&\leq d\|X\|_p^2. \qquad (7.8)
\end{aligned}$$

The lemma follows by combining inequalities (7.7) and (7.8). ■

7.4.4 Putting it All Together

Combining Theorem 7.3, Corollary 7.6, and Theorems 7.7 and 7.8, it follows that if our algorithm uses a sample of size $|S| = m$, then with probability at least $1 - \delta$ the hypothesis h which is generated will satisfy

$$\Pr_{x \in \mathcal{D}}[h(x) \neq \text{ sign}(u \cdot x)] = O\left(\frac{1}{m}\left(\frac{\|u\|_q^2\|X\|_p^2}{\delta_{u,X}^2}\log n \log^2 m + \log\frac{m}{\delta}\right)\right).$$

Thus we have established the following (where the $\tilde{O}$-notation hides log factors):

Theorem 7.11 *The algorithm obtained by applying* `AdaBoost` *to* `WLA` *using the parameter settings described in Corollary 7.6 is a strong learning algorithm for u on X with sample complexity $m(\epsilon, \delta, u, X) = \tilde{O}\left(\frac{1}{\epsilon} \cdot \frac{\|u\|_q^2\|X\|_p^2}{\delta_{u,X}^2}\right)$.*

7.5 Relationship with the Online p-norm Algorithms

The sample complexity of our boosting-based p-norm PAC learning algorithm is remarkably similar to that of the PAC-transformed online p-norm algorithms of Section 7.2.2. Up to log factors both sets of bounds depend linearly on ϵ^{-1} and quadratically on $\|u\|_q\|X\|_p/\delta_{u,X}$. Comparing the bounds in more detail, we see that while our bounds contain some logarithmic factors which are not present in the bounds of the PAC-transformed online p-norm algorithms, the online variant described in part (a) of Theorem 7.2 has an extra factor of $p-1$ in its bound which is not present in the sample complexity of our algorithm. Variant (a) offers the advantage, though, that the user does not need to know the values of any quantities such as $\|X\|_p$ or $\|u\|_q$ in advance in order to run the algorithm. Turning to part (b) of Theorem 7.2, we see that if the parameter a is set appropriately in the online algorithm then the online bound differs from our PAC algorithm bound only by an extra factor of

$$(p-1)\left(1 - \left(\frac{u \cdot z^0}{\|u\|_q\|z^0\|_p}\right)^2\right)$$

(again ignoring log factors). Part (c) of Theorem 7.2 shows that as $p \to \infty$ this factor becomes quite small even when z^0 is chosen to be $(1, \ldots, 1)$. It should be noted, though, that the bound in part (c) can be applied only if each coordinate of the target vector u is positive, while our algorithm has no such restriction on u.

We also note that when $p = \Omega(\log n)$ Gentile and Littlestone have given alternative expressions for the online p-norm bounds in terms of $\|X\|_\infty$ and $\|u\|_1$. For these values of p, using an entirely similar analysis the bounds of our algorithm can be analogously rephrased in terms of $\|X\|_\infty$ and $\|u\|_1$ as well.

7.5.1 $p = 2$ and the Perceptron Algorithm

Since the $p = 2$ case of the online p-norm algorithm is precisely the Perceptron algorithm, the $p = 2$ case of our algorithm can be viewed as a natural PAC-model analogue of the online Perceptron algorithm. We note that when $p = 2$ the upper bound given in Lemma 7.10 can be strengthened to $\sqrt{d} \cdot \|X\|_2$ (see Lemma 1.3 of (Bartlett and Shawe-Taylor, 1999) or Theorem 4.1 of (Alon *et al.*, 1992) for a proof). This means that the fat-shattering dimension upper bound of Theorem 7.8 can be improved to $\frac{1}{\xi^2}$, which removes a log factor from the bound of Theorem 7.11; however this bound will still contain various log factors because of the log terms in Theorem 7.7.

7.5.2 $p = \infty$ and the Jackson-Craven Algorithm

At the other extreme, we can also define a natural $p = \infty$ version of our algorithm. Consider the vectors z and w which are computed by the weak learning algorithm `WLA`. If we let r be the number of coordinates z_i of z such that $|z_i| = \|z\|_\infty$, then for any i we have

$$\begin{aligned}
\lim_{p\to\infty}\left(\frac{w_i}{\|w\|_q}\right) &= \lim_{p\to\infty}\left(\frac{\operatorname{sign}(z_i)|z_i|^{p-1}}{(\sum_{i=1}^n |z_i|^{(p-1)q})^{1/q}}\right) \\
&= \begin{cases} \operatorname{sign}(z_i)/r & \text{if } |z_i| = \|z\|_\infty \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

Hence it is natural to consider a $p = \infty$ version of `WLA`, which we denote `WLA'`, in which the vector w is defined by taking $w_i = \operatorname{sign}(z_i)$ if $|z_i| = \|z\|_\infty$ and $w_i = 0$ otherwise. All of our analysis continues to hold for the `WLA'` algorithm (with minor modifications as sketched below) and we obtain a $p = \infty$ strong learning algorithm:

Claim 7.12 *Theorem 7.11 holds for $p = \infty$ with `WLA'` in place of `WLA`.*

Proof: The proof of Theorem 7.3 (with $\mathtt{WLA}'$ in place of $\mathtt{WLA}$) is unchanged up through the point where we must show that

$$\frac{\sum_{i=1}^{m} \mathcal{D}(x^i) y_i (w \cdot x^i)}{\|w\|_1} \geq \frac{\delta_{u,X}}{\|u\|_1}.$$

The left-hand side of this inequality can be rewritten as

$$\begin{aligned} \frac{w \cdot z}{\|w\|_1} &= \frac{\sum_{|z_i| = \|z\|_\infty} \operatorname{sign}(z_i) z_i}{\sum_{|z_i| = \|z\|_\infty} 1} \\ &= \frac{\sum_{|z_i| = \|z\|_\infty} \|z\|_\infty}{\sum_{|z_i| = \|z\|_\infty} 1} \\ &= \|z\|_\infty, \end{aligned}$$

and hence it suffices to prove that $\|z\|_\infty \geq \delta_{u,X}/\|u\|_1$. This is established at the end of the proof of Theorem 7.3, so Theorem 7.3 holds with $p = \infty$ and $\mathtt{WLA}'$ substituted for $\mathtt{WLA}$.

The rest of the analysis goes through unchanged except for inequalities (7.7) and (7.8) of Lemma 7.10. Since $\|X\|_\infty = \sup_{x \in X} \max_{j=1,\ldots,n} |x_j|$, we have that $Y_j \leq d\|X\|_\infty^2$ for all j, and hence in place of inequalities (7.7) and (7.8) we have

$$\begin{aligned} \|z\|_\infty &= \max_j |z_j| \\ &\leq \max_j \sqrt{2Y_j \log 4n} \\ &\leq \sqrt{2d \log 4n} \cdot \|X\|_\infty, \end{aligned}$$

which proves lemma 7.10. ■

There is a close relationship between this $p = \infty$ algorithm and the work of Jackson and Craven on learning sparse perceptrons (Jackson and Craven, 1996). Note that if $r = 1$, i.e. only one coordinate of z has $|z_i| = \|z\|_\infty$, then the $\mathtt{WLA}'$ hypothesis is $h(x) = \frac{\ell}{\|X\|_\infty}$ where ℓ is the signed variable from $\{x_1, \ldots, x_n, -x_1, \ldots, -x_n\}$ which is most strongly correlated under distribution $\mathcal{D}$ with the value of $\operatorname{sign}(u \cdot x)$. This is very similar to the weak learning algorithm used by Jackson and Craven, which takes the single best-correlated literal as its hypothesis (breaking ties arbitrarily).

The proof that this "best-single-literal" algorithm used in (Jackson and Craven, 1996) is a weak learning algorithm is due to Goldmann, Hastad and Razborov (Goldmann *et al.*, 1992). However, their proof assumes that the example space X is $\{0,1\}^n$ and that the target vector u has all integer coefficients; thus, as noted by Jackson and Craven, their algorithm for learning sparse perceptrons only applies to learning problems which are defined over Boolean input domains. In contrast, our $p = \infty$ algorithm can be applied on continuous input domains – the only restrictions required by our algorithm are that the example space X and the target vector u satisfy $\|X\|_\infty < \infty$ and $\delta_{u,X} > 0$.

We also observe that Theorem 7.11 establishes a tighter sample complexity bound for our $p = \infty$ strong learning algorithm than was given by Jackson and Craven. To see this, let $X = \{0,1\}^n$ and suppose that the target vector $u \in \Re^n$ has all integer coefficients so the Jackson-Craven algorithm can be applied. For this learning problem we have $\delta_{u,X} = \Omega(1)$ and $\|X\|_\infty = 1$; letting $s = \|u\|_1$, Theorem 7.11 implies that our $p = \infty$ strong learning algorithm has sample complexity roughly s^2/ϵ (ignoring log factors). This is a substantial improvement over the roughly s^4/ϵ sample complexity bound given by Jackson and Craven. More generally, the sample complexity bound given by Jackson and Craven for learning "s-sparse k-perceptrons" is roughly ks^4/ϵ; the analysis of this chapter can easily be extended to establish a sample complexity bound of roughly ks^2/ϵ for learning s-sparse k-perceptrons.

Chapter 8

Boosting and Hard-Core Set Construction

In the last chapter we saw how `AdaBoost` can be applied to obtain PAC learning algorithms which are remarkably similar to Perceptron and Winnow. We now explore a different application of boosting, this time in the area of computational complexity theory. We show that a close connection exists between boosting and *hard-core set construction*, a type of hardness amplification from complexity theory. By employing boosting techniques from learning theory, we use this connection to give an improved bound for hard-core set construction which matches known lower bounds from boosting and thus is optimal within this class of techniques. We also show how to apply ideas from hard-core set construction to give a new version of Jackson's celebrated boosting-based Harmonic Sieve algorithm for learning DNF formulae under the uniform distribution using membership queries. Our new version of the Sieve has a significant asymptotic improvement in running time. Critical to our arguments is a careful analysis of the distributions which are employed in both boosting and hard-core set constructions.

8.1 Introduction

8.1.1 Boosting and Hard-Core Sets

We refer to a *hardness amplification* as a result of the following form: given a Boolean function f that is mildly inapproximable by circuits of some bounded size g, construct

Reference:	Set size parameter:	Circuit size parameter:
Impagliazzo (Impagliazzo, 1995)	ϵ	$O(\gamma^2\epsilon^2)\cdot g$
Nisan (Impagliazzo, 1995)	ϵ	$O(\gamma^2(\log(1/\gamma\epsilon))^{-1})\cdot g$
Our Results	$\epsilon/2$	$O(\gamma^2(\log(1/\epsilon))^{-1})\cdot g$

Table 8.1: Comparison of known hard-core set constructions.

from f a new function f' that is highly inapproximable by all circuits of size closely related to g. Here "mildly inapproximable" means roughly that no circuit can agree with f on a fraction of inputs very close to 1, while "highly inapproximable" means that no circuit can agree with f on a fraction of inputs significantly greater than 1/2. Hardness amplification results are a crucial component of recent attempts to derandomize BPP (Babai *et al.*, 1993; Impagliazzo and Widgerson, 1997; Nisan and Wigderson, 1994). Perhaps the most famous hardness amplification result is Yao's XOR-lemma (Goldreich *et al.*, 1995), which states that if a Boolean function f is mildly inapproximable by circuits of size g then the XOR of several independent copies of f is highly inapproximable for circuits of size closely related to g.

Superficially, boosting and hardness amplification seem to have opposite goals. Recall that a boosting algorithm takes as input a *weak learning algorithm* which can generate hypotheses which have accuracy only slightly better than 1/2, and outputs a final hypothesis which is a highly accurate predictor for the target function. Thus boosting constructs a hypothesis which closely approximates a function f while hardness amplification results prove that certain functions are hard to approximate. The proof techniques employed in both areas, however, have a similar structure. All known hardness amplification results go by contradiction: assuming there exists a circuit C capable of mildly approximating f', one proves the existence of a slightly larger circuit which closely approximates f. From this perspective, a hardness amplification proof resembles a type of boosting procedure: circuits which mildly approximate a function f' (these correspond to the hypotheses output by the weak learner) are combined to form a new circuit computing f on a large fraction of inputs.

In an important paper, Impagliazzo (Impagliazzo, 1995) reduces the problem of amplifying the hardness of a function f to the problem of constructing a distribution $\mathcal{D}$ such that f is highly inapproximable by small circuits for inputs chosen according to $\mathcal{D}$. He then constructs such a distribution and uses it to prove an XOR lemma.

Impagliazzo also shows that the existence of such a distribution implies the existence of a "hard-core set" as defined in Section 8.2; we thus refer to Impagliazzo's method of constructing such a distribution as a *hard-core set construction*. Schapire (Schapire, 1990) was the first to point out that the existence of a boosting algorithm implies the existence of such a distribution.

8.1.2 Our Results

We give an explicit correspondence between the distributions that arise in Impagliazzo's hard-core set construction and the distributions constructed by boosting algorithms. This observation allows us to prove that the hard-core set construction of Impagliazzo *is* a boosting algorithm when the initial distribution is uniform. As we will show, there are two important parameters which boosting and hard-core set constructions share: the number of "stages" required and the "smoothness" of the distributions which are constructed. Interestingly, the procedures which have been used for hard-core set construction have better smoothness and can be used to improve algorithms in computational learning theory, while boosting algorithms require fewer stages and can be used to improve hard-core set construction.

We first show how to use known boosting algorithms to obtain new hard-core set constructions. Impagliazzo proves the following theorem: given a function f such that no circuit of size less than g correctly computes f on more than $(1-\epsilon)2^n$ inputs, then for any $\gamma < 1/2$ there exists a set S of size $\epsilon 2^n$ such that no circuit of size $O(\gamma^2\epsilon^2)g$ can correctly compute f on more than a $(1/2+\gamma)$ fraction of the inputs in S. By letting known boosting algorithms dictate the construction of the distributions in Impagliazzo's proof, we improve on previous results with respect to the circuit size parameter with only a small constant factor loss in the set size parameter. As explained in Section 8.4.6, we believe our parameters to be optimal up to constant factors with respect to this class of techniques. Table 1 summarizes our hard-core set construction results.

We then show how to use Impagliazzo's hard-core set construction to obtain a more efficient version of Jackson's breakthrough Harmonic Sieve algorithm (Jackson, 1997) for learning DNF formulae under the uniform distribution using membership queries. Jackson's original algorithm learns using the hypothesis class of threshold-of-parity functions and runs in time essentially $\tilde{O}(ns^8/\epsilon^{12})$, where n is the number of variables in the DNF formula, s is the number of terms, and ϵ is the accuracy parameter. Our variant uses the same hypothesis class and runs in time $\tilde{O}(ns^8/\epsilon^8)$. We can further improve the running time to $\tilde{O}(ns^8/\epsilon^6)$ at the cost of learning using a more complex class of hypotheses.

In recent work Bshouty, Jackson and Tamon (Bshouty *et al.*, 1999) have improved the running time of the Harmonic Sieve to $\tilde{O}(ns^4/\epsilon^4)$, where r is the number of distinct variables which appear in the minimal DNF representation of the target formula. Our results improve the running time of their new algorithm to $\tilde{O}(ns^4/\epsilon^2)$ time steps, which is the fastest known algorithm for PAC learning DNF with membership queries under the uniform distribution.

Our main technical contribution is a careful analysis of the distributions constructed during the boosting process. We show that boosting procedures which construct distributions with high minimum entropy are desirable for good hard-core set constructions.

8.1.3 Related Work

Boneh and Lipton (Boneh and Lipton, 1993) have applied Yao's XOR-lemma to prove the equivalence of weak and strong learnability for certain types of concept classes under the uniform distribution. Their result applies to concept classes closed under a polynomial number of XOR operations.

8.2 Hard-Core Set Construction Overview

Our first definition, taken from (Impagliazzo, 1995), formalizes the notion of a function which is hard to approximate. Readers who are familiar with the notation of (Impagliazzo, 1995) will notice that we are using different variables; the reasons for this will become clear in Section 8.4.

Definition 8.2 *Let f be a Boolean function on $\{0,1\}^n$ and $\mathcal{D}$ a distribution on $\{0,1\}^n$. Let $0 < \epsilon < 1/2$ and let $n \leq g \leq 2^n/n$. We say that f is* ϵ-hard for size g under $\mathcal{D}$ *if for any Boolean circuit C with at most g gates, we have* $\Pr_{\mathcal{D}}[f(x) = C(x)] \leq 1-\epsilon$.

In other words, any circuit of size at most g must disagree with f with probability at least ϵ for x drawn according to $\mathcal{D}$. We write $\mathcal{U}$ to denote the uniform distribution

Input: real numbers $0 < \epsilon < 1$, $0 < \gamma < 1/2$
Boolean function f

Output: circuit h such that $\Pr_{\mathcal{U}}[h(x) = f(x)] > 1 - \epsilon$

set $i \leftarrow 0$
set $M_0(x) \equiv 1$
until $\mu(M_i) < \epsilon$ **do**
  let C_i be a circuit of size at most g' with $\Pr_{\mathcal{D}_{M_i}}[C(x) = f(x)] \geq 1/2 + \gamma$
  set $R_{C_i}(x) \equiv 1$ if $f(x) = C_i(x)$, $R_{C_i}(x) \equiv -1$ otherwise
  set $N_i(x) \equiv \sum_{0 \leq j \leq i} R_{C_j}(x)$
  set $M_{i+1}(x) \equiv \begin{cases} 1 & \text{if } N_i(x) < 0 \\ 1 - \epsilon\gamma N_i(x) & \text{if } 0 \leq N_i(x) \leq 1/\epsilon\gamma \\ 0 & \text{if } N_i(x) > 1/\epsilon\gamma \end{cases}$
  set $i \leftarrow i + 1$
return $h \equiv MAJ(C_0, C_1, \ldots, C_{i-1})$

Figure 8.1: The `IHA` algorithm.

over $\{0,1\}^n$.

Definition 8.3 *A* measure *on $\{0,1\}^n$ is a function $M : \{0,1\}^n \to [0,1]$. The* absolute size *of a measure M is denoted by $|M|$ and equals $\sum_x M(x)$; the* relative size *of M is denoted $\mu(M)$ and equals $|M|/2^n$.*

Definition 8.4 *For any real valued function ξ, $L_\infty(\xi)$ denotes* $\max_x |\xi(x)|$.

The quantity $\log(L_\infty(\mathcal{D})^{-1})$ is often referred to as the *minimum entropy* of $\mathcal{D}$. There is a natural correspondence between measures and distributions: the distribution $\mathcal{D}_M$ induced by a measure M is defined by $\mathcal{D}_M(x) = M(x)/|M|$. Conversely, if $\mathcal{D}$ is a distribution then the measure $M_{\mathcal{D}}$ induced by $\mathcal{D}$ is defined by $M_{\mathcal{D}}(x) = \mathcal{D}(x)/L_\infty(\mathcal{D})$. Thus $M_{\mathcal{D}}$ is the largest measure which is a constant-multiple rescaling of $\mathcal{D}$ (note that $\mathcal{D}$ itself is a measure, though typically one which has much smaller size than $M_{\mathcal{D}}$). It is clear that $|M_{\mathcal{D}}| = 1/L_\infty(\mathcal{D})$ and $\mu(M_{\mathcal{D}}) = 1/L_\infty(2^n\mathcal{D})$. Thus, large measures correspond to distributions which do not assign large weight to any point (i.e., have high minimum entropy).

The next definition is also from (Impagliazzo, 1995):

Definition 8.5 *We say that f is* γ-hard-core on M for size g *if* $\Pr_{\mathcal{D}_M}[f(x) = C(x)] \leq 1/2 + \gamma$ *for every circuit C of size at most g. For $S \subseteq \{0,1\}^n$ we say that f* is γ-hard-core on S for size g *if f is γ-hard-core on M_S for size g, where $M_S(x)$ is the characteristic function of S.*

8.2.1 Existence of Hard-Core Measures

The following theorem due to Impagliazzo is the starting point of all our results:

Theorem 8.6 (Impagliazzo, 1995) *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ and let $0 < \gamma < 1/2$. Then there is a measure M with $\mu(M) \geq \epsilon$ such that f is γ-hard-core on M for size $g' = O(\epsilon^2\gamma^2)g$.*

Proof Sketch: Assume by way of contradiction that for every measure M with $\mu(M) \geq \epsilon$ there is a circuit C_M of size at most g' such that $\Pr_{\mathcal{D}_M}[f(x) = C_M(x)] > 1/2 + \gamma$. Now consider the algorithm `IHA` which is given in Figure 8.2. This algorithm iteratively modifies M until its relative size is less than ϵ. After each modification we obtain a circuit C_M as above. Once the relative size of M becomes less than ϵ we combine the circuits obtained during the process to contradict the original assumption. The following easily verifiable claims are useful for understanding how `IHA` works:

- $N_i(x)$ is the margin by which the majority vote of $C_0, \ldots, C_i$ correctly predicts the value of $f(x)$.
- The measure M_{i+1} assigns weight 0 to points where the margin of correctness is large, weight 1 to points where the margin is negative, and intermediate weight to points where the margin is positive but small.

Impagliazzo proves that after at most $i_0 = O(1/(\epsilon^2\gamma^2))$ cycles through the loop, $\mu(M_i)$ must be less than ϵ. Once this happens and we exit the loop, it is easy to see that $h \equiv MAJ(C_0, \ldots, C_{i-1})$ agrees with f on all inputs except those which have $N_i(x) \leq 0$ and hence $M_i(x) = 1$. Since $\mu(M_i) < \epsilon$, this implies that $\Pr_{\mathcal{U}}[f(x) = h(x)] \geq 1 - \mu(M_i) > 1 - \epsilon$. But h is a majority circuit over at most i_0 circuits each of size at most g', and majority over i_0 inputs can be computed by a circuit of size $O(i_0)$. It follows that h has at most $g'i_0 + O(i_0) \leq g$ gates, which contradicts the original assumption that f is ϵ-hard for circuits of size g under $\mathcal{U}$. ∎

Using a non-constructive proof technique, Nisan has established a similar result which is reported in (Impagliazzo, 1995). In Nisan's theorem the circuit size parameter is slightly worse as a function of γ but substantially better as a function of ϵ :

Theorem 8.7 (Impagliazzo, 1995) *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ and let $0 < \gamma < 1/2$. Then there is a measure M with $\mu(M) \geq \epsilon$ such that f is γ-hard-core on M for size $g' = O(\gamma^2(\log(2/\gamma\epsilon))^{-1})g$.*

In Section 8.4.2 we will establish results of this type which have a better circuit size parameter than either Theorem 8.6 or Theorem 8.7.

We note that Theorems 8.6 and 8.7 assert the existence of a large measure, not a large set as was promised in Section 1. Using a probabilistic argument which we give in Section 8.4.4 Impagliazzo has shown that the existence of a large measure M on which f is hard-core implies the existence of a large set S on which f is also hard-core (with slightly different parameters).

8.3 Boosting Overview

In this section we review the notions of weak and strong learning in the PAC learning model and give a general framework for boosting algorithms which convert weak learners into strong ones.

Recall that a a concept class over $\{0,1\}^n$ is a collection $C = \cup_{n\geq 1} C_n$ of Boolean functions where each $f \in C_n$ is a Boolean function on $\{0,1\}^n$. If f and h are two Boolean functions on $\{0,1\}^n$ and $\mathcal{D}$ is a distribution on $\{0,1\}^n$, we say that h is an *ϵ-approximator for f under $\mathcal{D}$* if $\Pr_{\mathcal{D}}[f(x) = h(x)] \geq 1 - \epsilon$. The learning algorithm has access to an *example oracle* $\mathrm{EX}(f, \mathcal{D})$ which, when queried, provides a labeled example $\langle x, f(x)\rangle$ where x is drawn from $\{0,1\}^n$ according to the distribution $\mathcal{D}$ and $f \in C_n$ is the unknown target concept which the algorithm is trying to learn. We have the following formal definition:

Definition 8.8 *An algorithm* `A` *is a* strong PAC learning algorithm for a concept class C *if the following condition holds: for any $n \geq 1$, any $f \in C_n$, any distribution $\mathcal{D}$ on $\{0,1\}^n$, and any $0 < \epsilon, \delta < 1$, if* `A` *is given access to n, ϵ, δ and $EX(f, \mathcal{D})$, then*

$\mathtt{A}$ runs in time polynomial in n, ϵ^{-1}, δ^{-1}, and $size(f)$, and with probability at least $1-\delta$ algorithm $\mathtt{A}$ outputs an ϵ-approximator for f under $\mathcal{D}$.

Here $size(f)$ measures the complexity of the function f under some fixed reasonable encoding scheme. For the concept class DNF which we will consider in Section 8.5, $size(f)$ is the minimum number of terms in any disjunctive normal form representation of f.

If the algorithm $\mathtt{A}$ is only guaranteed to find a $(1/2-\gamma)$-approximator for some $\gamma > 0$, then we say that $\mathtt{A}$ is a $(1/2-\gamma)$-*approximate* learning algorithm. If $\gamma = \Omega(1/p(n, size(f)))$ for some polynomial p then we say that $\mathtt{A}$ is a *weak* learning algorithm. We will abuse notation and say that $\mathtt{A}$ is a $(1/2-\gamma)$-approximate learning algorithm for f if $\mathtt{A}$ is a $(1/2-\gamma)$-approximate learning algorithm for the concept class C which consists of the single function f.

Schapire (Schapire, 1990) and subsequently Freund (Freund, 1990; Freund, 1992) have given explicit *boosting* algorithms which efficiently transform weak learning algorithms into strong ones. We now formally define boosting algorithms (a related definition can be found in (Freund, 1995)):

Definition 8.9 *An algorithm $\mathtt{B}$ is said to be a* boosting algorithm *if it satisfies the following condition: for any Boolean function f and any distribution $\mathcal{D}$, if $\mathtt{B}$ is given $0 < \epsilon, \delta < 1$, $0 < \gamma \leq 1/2$, an example oracle $EX(f, \mathcal{D})$, and a $(1/2-\gamma)$-approximate learning algorithm $\mathtt{WL}$ for f which runs in time T, then algorithm $\mathtt{B}$ runs in time polynomial in $T, \gamma^{-1}, \epsilon^{-1}$, and δ^{-1}, and with probability at least $1-\delta$ algorithm $\mathtt{B}$ outputs an ϵ-approximator for f under $\mathcal{D}$.*

8.3.1 Structure of Boosting Algorithms

All known boosting algorithms rely crucially on the fact that the weak learning algorithm $\mathtt{WL}$ can find a $(1/2-\gamma)$-approximator for f under $\mathcal{D}'$ for *any* distribution $\mathcal{D}'$, as long as $\mathtt{WL}$ is given access to the example oracle $\mathrm{EX}(f, \mathcal{D}')$. We give the following high-level definition:

Definition 8.10 *A* canonical boosting algorithm *is a boosting algorithm which has the following iterative structure:*

- *At stage 0 the algorithm starts with $\mathcal{D}_0 = \mathcal{D}$ and uses* WL *to generate a* $(1/2-\gamma)$*-approximator h_0 for f under $\mathcal{D}_0$.*
- *At stage i the boosting algorithm does two things: (1) defines a distribution $\mathcal{D}_i$ which favors points where the previous hypotheses $h_0, \ldots, h_{i-1}$ do poorly at predicting the value of f, and (2) simulates the example oracle $EX(f, \mathcal{D}_i)$ and lets* WL *access this simulated example oracle to produce a hypothesis h_i which is a* $(1/2-\gamma)$*-approximator for f under $\mathcal{D}_i$.*
- *After doing this repeatedly for several stages, the boosting algorithm combines the hypotheses $h_0, \ldots, h_{i-1}$ in some way to obtain a final hypothesis h which is an ϵ-approximator for f under $\mathcal{D}$.*

8.4 Hard-Core Set Construction from Boosting

8.4.1 A Structural Similarity

From the descriptions of the hard-core set construction of Section 8.2 and the canonical boosting algorithm of Section 8.3, one can see a close structural resemblance between the IHA algorithm and the canonical boosting algorithm outlined above. To be more specific, just as IHA assumes that at each stage there is a circuit C_i for which $\Pr_{\mathcal{D}_{M_i}}[f(x) \neq C_i(x)] \leq 1/2 - \gamma$, the canonical boosting algorithm assumes that WL can generate at each stage a hypothesis h_i for which $\Pr_{\mathcal{D}_i}[f(x) \neq h_i(x)] \leq 1/2 - \gamma$. The induced distributions $\mathcal{D}_{M_i}$ of IHA correspond precisely to the distributions $\mathcal{D}_i$ of the canonical boosting algorithm (note that IHA starts off with the measure $M_0 = 1$ which corresponds to the uniform distribution $\mathcal{U} = \mathcal{D}_0$). Finally, just as the canonical boosting algorithm combines the hypotheses $h_0, \ldots, h_{i-1}$ in some fashion to obtain a final hypothesis h which has $\Pr_{\mathcal{U}}[f(x) = h(x)] \geq 1 - \epsilon$, the IHA algorithm combines the circuits $C_0, \ldots, C_{i-1}$ by taking majority to obtain a circuit h such that $\Pr_{\mathcal{U}}[f(x) = h(x)] \geq 1 - \epsilon$.

We conclude that IHA is an algorithm which succeeds in boosting *provided that the starting distribution is the uniform distribution $\mathcal{U}$.* Since boosting algorithms from computational learning theory will work for *any* starting distribution, *a priori* it seems as if it should be possible to use any boosting algorithm in place of IHA and obtain a hard-core set construction. In the next section we prove a theorem which formalizes this idea and emphasizes the parameters which are important to obtain a good hard-core set construction.

8.4.2 A General Hard-Core Set Construction

Definition 8.11 *Let $\mathcal{D}$ be a distribution over $\{0,1\}^n$. For $d \geq 1$ we say that $\mathcal{D}$ is d-*smooth *if $L_\infty(2^n\mathcal{D}) \leq d$.*

As an immediate consequence of Definitions 8.3 and 8.11, we have

Observation 8.12 *If distribution $\mathcal{D}$ is d-smooth then $\mu(M_{\mathcal{D}}) \geq 1/d$.*

Definition 8.13 *Let* `B` *be a canonical boosting algorithm which takes as input ϵ, δ, γ, an example oracle $EX(f, \mathcal{D})$, and a $(1/2 - \gamma)$-approximate learning algorithm* `WL` *for f.*

1. *We say that* `B` *is a $k(\epsilon, \gamma)$-*stage *boosting algorithm if the following holds: For all example oracles $EX(f, \mathcal{D})$ and $(1/2-\gamma)$-approximate learners* `WL` *for f, algorithm* `B` *simulates at most $k = k(\epsilon, \gamma)$ distributions $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{k-1}$ for* `WL` *and uses* `WL` *to generate at most k hypotheses $h_0, \ldots, h_{k-1}$.*

2. *We say that* `B` *is a $d(\epsilon, \gamma)$-*smooth *boosting algorithm if the following holds: For all functions f and $(1/2 - \gamma)$-approximate learners* `WL`*, when* `B` *is given $EX(f, \mathcal{U})$ and* `WL`*, with nonzero probability both of the following events occur: (i) the simulated distributions $\mathcal{D}_0, \ldots, \mathcal{D}_{k-1}$ are each $d(\epsilon, \gamma)$-smooth, and (ii) the hypothesis h which* `B` *outputs satisfies $\Pr_{\mathcal{U}}[f(x) = h(x)] \geq 1 - \epsilon$.*

The property of the distributions $\mathcal{D}_i$ described in part 2 of the above definition is similar to Levin's notion of "dominated" distributions (Levin, 1986).

Now we can state the following theorem which generalizes Impagliazzo's hard-core set construction.

Theorem 8.14 *Let* `B` *be a $k(\epsilon, \gamma)$-stage, $d(\epsilon, \gamma)$-smooth boosting algorithm which outputs as its final hypothesis a circuit of size r over inputs $h_0, \ldots, h_{k-1}$. Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ and let $0 < \gamma < 1/2$. Then there is a measure M on $\{0,1\}^n$ with $\mu(M) \geq 1/d(\epsilon, \gamma)$ such that f is γ-hard-core on M for size $g' = (g - r)/k(\epsilon, \gamma)$.*

Proof: The proof is analogous to the proof of Theorem 8.6. Assume by way of contradiction that for every measure M with $\mu(M) \geq 1/d(\epsilon,\gamma)$ there is a circuit C_M of size at most g' such that $\Pr_{\mathcal{D}_M}[f(x) = C_M(x)] \geq 1/2 + \gamma$. By Observation 8.12, this implies that for every $d(\epsilon,\gamma)$-smooth distribution $\mathcal{D}$ there is a circuit $C_{\mathcal{D}}$ of size at most g' such that $\Pr_{\mathcal{D}}[f(x) = C_{\mathcal{D}}(x)] \geq 1/2 + \gamma$.

Now run the boosting algorithm `B` on inputs ϵ, δ, γ, and $\mathrm{EX}(f, \mathcal{U})$. Since `B` is $d(\epsilon,\gamma)$-smooth, with nonzero probability we have that (i) every distribution $\mathcal{D}_i$ which `B` simulates will be $d(\epsilon,\gamma)$-smooth, and (ii) the final hypothesis which `B` outputs is an ϵ-approximator to f under the original distribution $\mathcal{U}$. By (i), there must exist a circuit C_i of at most g' gates which is a $(1/2-\gamma)$-approximator for f under $\mathcal{D}_i$. Give `B` this circuit when it calls `WL` on distribution $\mathcal{D}_i$. Now by (ii), the final hypothesis which `B` outputs must be an ϵ-approximator to f under the original distribution $\mathcal{U}$. But since `B` is $k(\epsilon,\gamma)$-stage, this final hypothesis is a circuit of size at most $r + g'k(\epsilon,\gamma) \leq g$, which contradicts the original assumption that f is ϵ-hard for circuits of size g under $\mathcal{U}$. ■

8.4.3 New Hard-Core Set Constructions

Here we apply Theorem 8.14 to obtain new hard-core set constructions from known boosting algorithms. We proceed in stages. First, we show how two different boosting algorithms yield different hard-core set constructions. Next, we combine these boosting algorithms to achieve a new hard-core set construction which improves on results of Impagliazzo and Nisan in the circuit size parameter but has a set size parameter worse than their results by a logarithmic factor. In Section 8.4.5 we improve the set size parameter to within a constant factor of the Impagliazzo/Nisan results.

We first consider Freund's boost-by-majority algorithm from (Freund, 1990) which, following (Jackson, 1995), we refer to as `F1`. Algorithm `F1` is a $k = O(\gamma^{-2}\log(1/\epsilon))$-stage boosting algorithm which combines its k hypotheses using the majority function. Jackson's analysis ((Jackson, 1995), pp. 57–59) yields the following fact about `F1`:

Fact 8.15 *If* `F1` *is given inputs* ϵ, δ, γ, *EX*$(f, \mathcal{D})$ *and a* $(1/2-\gamma)$*-approximate weak learner* `WL` *for* f, *then with high probability each distribution* $\mathcal{D}'$ *which* `F1` *simulates for* `WL` *satisfies*

$$L_\infty(\mathcal{D}') = O(1/\epsilon^3) \cdot L_\infty(\mathcal{D}).$$

This immediately implies that F1 is $O(1/\epsilon^3)$-smooth.[1] We thus obtain the following hard-core set construction:

Theorem 8.16 *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ and let $0 < \gamma < 1/2$. Then there is a measure M on $\{0,1\}^n$ with $\mu(M) = \Omega(\epsilon^3)$ such that f is γ-hard-core on M for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$.*

Next, we consider Freund's later $\mathtt{B}_{\mathrm{Filt}}$ algorithm from (Freund, 1995) (the name comes from the fact that the algorithm "filters" examples from the original distribution to simulate new distributions). Like F1, algorithm $\mathtt{B}_{\mathrm{Filt}}$ is a k-stage boosting algorithm for $k = O(\gamma^{-2}(\log 1/\epsilon))$. $\mathtt{B}_{\mathrm{Filt}}$ combines its $(1/2-\gamma)$-approximators to obtain an ϵ-approximator for f by using a majority function on k inputs which may have some random inputs. A straightforward argument shows that some circuit of size $O(k)$ is a ϵ-approximator for f. To analyze the smoothness of $\mathtt{B}_{\mathrm{Filt}}$, we use the following fact which follows from Lemma 3.4 and Lemma 3.9 of (Freund, 1995):

Fact 8.17 *If $\mathtt{B}_{Filt}$ is given inputs ϵ, δ, γ, $EX(f, \mathcal{D})$ and a $(1/2-\gamma)$-approximate weak learner WL for f, then with high probability each distribution $\mathcal{D}'$ which $\mathtt{B}_{Filt}$ simulates for WL satisfies*

$$L_\infty(\mathcal{D}') = O(\log(1/\epsilon)/(\epsilon\gamma)) \cdot L_\infty(\mathcal{D}).$$

Since Fact 8.17 implies that $\mathtt{B}_{\mathrm{Filt}}$ is $O(\log(1/\epsilon)/(\epsilon\gamma))$-smooth, we obtain

Theorem 8.18 *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ and let $0 < \gamma < 1/2$. Then there is a measure M on $\{0,1\}^n$ with $\mu(M) = \Omega(\epsilon\gamma(\log(1/\epsilon))^{-1})$ such that f is γ-hard-core on M for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$.*

Finally we establish a stronger hard-core set construction by combining the previous two approaches. In (Freund, 1992) Freund describes a two-level boosting algorithm which works as follows: algorithm F1 is used to boost from accuracy $(1/2-\gamma)$ to accuracy $1/4$, and algorithm $\mathtt{B}_{\mathrm{Filt}}$ boosts from accuracy $1/4$ to accuracy ϵ by taking F1 as its weak learner. We call this combined algorithm $\mathtt{B}_{\mathrm{Comb}}$.

[1] In (Freund, 1990) Freund states without proof that the F1 algorithm can be shown to be $O(1/\epsilon^2)$-smooth. Jackson proves that F1 is $O(1/\epsilon^3)$-smooth and states that variants of F1 can be shown to be $O(1/\epsilon^{2+\rho})$-smooth for arbitrarily small values $\rho > 0$.

Lemma 8.19 *The boosting algorithm* $\mathtt{B}_{Comb}$ *is an* $O(\gamma^{-2}\log(1/\epsilon))$*-stage boosting algorithm.*

Proof: The top level of $\mathtt{B}_{\mathrm{Comb}}$, which uses algorithm $\mathtt{B}_{\mathrm{Filt}}$, takes $O(\log(1/\epsilon))$ stages since the weak learner which it uses is `F1` which provides $(1/2-\gamma')$-accurate hypotheses with $\gamma' = 1/4$. The bottom level, which uses algorithm `F1`, takes $O(\gamma^{-2})$ stages since it boosts a $(1/2-\gamma)$-approximate learner to accuracy $1/4$. Consequently the combined algorithm $\mathtt{B}_{\mathrm{Comb}}$ uses the claimed number of stages. ■

Lemma 8.20 $\mathtt{B}_{Comb}$ *is an* $O(\log(1/\epsilon)/\epsilon)$*-smooth boosting algorithm.*

Proof: Since $\mathtt{B}_{\mathrm{Filt}}$ is boosting from accuracy $1/4$ to accuracy ϵ using `F1` as its weak learner, Fact 8.17 implies that each distribution $\mathcal{D}'$ which $\mathtt{B}_{\mathrm{Filt}}$ passes to `F1` satisfies

$$L_\infty(\mathcal{D}') = O(\log(1/\epsilon)/\epsilon) \cdot L_\infty(\mathcal{D}).$$

Since `F1` is boosting from accuracy $(1/2-\gamma)$ to accuracy $1/4$, Fact 8.15 implies that if $\mathcal{D}''$ is the distribution which `F1` passes to `WL`, then

$$L_\infty(\mathcal{D}'') = O(1) \cdot L_\infty(\mathcal{D}').$$

Combining these two equations, we find that

$$L_\infty(\mathcal{D}'') = O(\log(1/\epsilon)/\epsilon) \cdot L_\infty(\mathcal{D}).$$

■

Finally, we note that the final hypothesis which $\mathtt{B}_{\mathrm{Comb}}$ outputs is a depth 2 majority circuit over the weak hypotheses h_i, since both `F1` and $\mathtt{B}_{\mathrm{Filt}}$ combine their hypotheses using the majority function. A straighforward bound on the size of this majority circuit yields the following hard-core set construction:

Theorem 8.21 *Let* f *be* ϵ*-hard for circuits of size* g *under* $\mathcal{U}$ *and let* $0 < \gamma < 1/2$. *Then there is a measure* M *on* $\{0,1\}^n$ *with* $\mu(M) = \Omega(\epsilon(\log(1/\epsilon))^{-1})$ *such that* f *is* γ*-hard-core on* M *for size* $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$.

8.4.4 From Hard-Core Measures to Hard-Core Sets

While we have been referring to our results thus far as hard-core set constructions, in fact Theorems 8.16, 8.18 and 8.21 establish the existence of hard-core measures rather than hard-core sets. We now show how hard-core measure constructions such as Theorems 8.16, 8.18 and 8.21 imply corresponding hard-core set constructions. This conversion from hard-core measures to hard-core sets is based on an argument from (Impagliazzo, 1995).

We will use the following crude lemma:

Lemma 8.22 *The number of Boolean circuits of size g is at most $((n+5)g^2)^g$.*

Proof: To specify a circuit it suffices to specify, for each of g gates, the two inputs and the label of the gate. For a given gate there are at most g^2 choices for the inputs and at most $n+5$ choices $(x_1, \ldots, x_n, \neg, \vee, \wedge, TRUE, FALSE)$ for the label. ■

The following easy fact follows from the definition of $|M|$.

Fact 8.23 *Let C be a circuit and view C and f as taking values in $\{-1,1\}$. Then $\Pr_{\mathcal{D}_M}[C(x)=f(x)] = \frac{1}{2}+\rho$ if and only if $\sum_{x\in\{0,1\}^n} M(x)C(x)f(x) = 2\rho|M|$.*

Now we state and prove our conversion from hard-core measures to hard-core sets. Constant factors have not been optimized in the following lemma.

Lemma 8.24 *Let f be a Boolean function on $\{0,1\}^n$ and M a measure such that (i) $\mu(M) \geq \tau$ and (ii) f is γ-hard-core on M for size g where $g \leq \frac{1}{2n}\cdot 2^n\gamma^2\tau^2$. Then there exists a set S with $|S| \geq \frac{\tau}{2}2^n$ such that f is 4γ-hard-core on S for size g.*

Proof: Let C be any circuit of size at most g. Consider the following randomized construction of a set $S \subseteq \{0,1\}^n$: for each $x \in \{0,1\}^n$ put x in S with probability $M(x)$. Let M_S be the characteristic function of S. For each $x \in \{0,1\}^n$ the expected value of $M_S(x)$ is $M(x)$, and thus by linearity of expectation we have

$$E\left[\sum_{x\in\{0,1\}^n} M_S(x)C(x)f(x)\right] = \sum_{x\in\{0,1\}^n} M(x)C(x)f(x) \leq 2\gamma|M|$$

where the inequality follows from Fact 8.23 and f being γ-hard-core on M for size g. For each value of x the quantity $M_S(x)C(x)f(x)$ is a random variable in the interval

$[-1,1]$. Hoeffding's tail bound now implies that

$$\begin{aligned}\Pr\left[\frac{1}{2^n}\sum_{x\in\{0,1\}^n} M_S(x)C(x)f(x) \geq 4\gamma\mu(M)\right] &\leq \exp\left(\frac{-2\cdot 2^n(2\gamma\mu(M))^2}{4}\right) \\ &\leq \exp(-2\cdot 2^n\gamma^2\tau^2)\end{aligned}$$

where the second inequality is because $\mu(M) \geq \tau$. By Lemma 8.22 and the bound on g we have that the number of circuits of size at most g is at most $((n+5)g^2)^g \ll \frac{1}{10}\exp(2\cdot 2^n\gamma^2\tau^2)$. Thus the probability that there exists a C with $|C| \leq g$ such that $\sum_{x\in\{0,1\}^n} M_S(x)C(x)f(x) \geq 4\gamma|M|$ is less than $\frac{1}{10}$.

Meanwhile, we also have that $E[|S|] = |M|$ and $|S|$ is a sum of 2^n independent random variables each with range $\{0,1\}$. Thus Hoeffding's bound implies that

$$\begin{aligned}\Pr\left[\frac{|S|}{2^n} < \frac{\mu(M)}{2}\right] &\leq \exp(-2\cdot 2^n(\mu(M)/2)^2) \\ &\leq \exp(-2^n\tau^2/2).\end{aligned}$$

If $\tau < \frac{1}{2^{n/2}}$ then the bound on g in the statement of the lemma is less than $\frac{1}{2n}$ and the lemma is trivially true. Thus we assume that $\tau \geq \frac{1}{2^{n/2}}$ which by the above inequality implies that $\Pr[|S| \geq \frac{|M|}{2}] > \frac{1}{10}$.

These two probability bounds together imply that there exists some S with $|S| \geq \frac{|M|}{2} \geq \frac{\tau}{2}2^n$ such that for every circuit C with at most g gates

$$\sum_{x\in\{0,1\}^n} M_S(x)C(x)f(x) \leq 4\gamma|M| \leq 8\gamma|S| = 8\gamma|M_S|.$$

By Fact 8.23 we have that $\Pr_{x\in S}[C(x) = f(x)] \leq \frac{1}{2} + 4\gamma$ for every such circuit C, and thus f is 4γ-hard-core on S. ■

Combining this lemma with Theorem 8.21 we obtain the following hard-core set construction:

Theorem 8.25 *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ where $g = O\left(\frac{2^n}{n}\cdot\frac{\epsilon^2}{\log 1/\epsilon}\right)$ and let $0 < \gamma < 1/2$. Then there is a set $S \subseteq \{0,1\}^n$ with $|S| = \Omega(\epsilon(\log 1/\epsilon)^{-1})2^n$ such that f is 4γ-hard-core on S for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1}g)$.*

8.4.5 Improving the Set Size Parameter

In this section we show how the set size parameter of Theorem 8.25 can be improved from $\Omega(\epsilon(\log 1/\epsilon)^{-1})$ to $\Omega(\epsilon)$. The basic idea is quite simple and was suggested by Avi Wigderson (Wigderson, 1999). If the hard-core set S is much smaller than $\epsilon 2^n$ then we can take away the points in this hard-core set and f must still be almost ϵ-hard for circuits of size g under the uniform distribution on the remaining points. We can then reapply Theorem 8.25 on the remaining points using "almost ϵ" in place of ϵ to obtain a new hard-core set. This procedure can be repeated until the total size of all the hard-core sets is $\Omega(\epsilon 2^n)$.

To make this argument precise we will need to consider measures which are defined over proper subsets of $\{0,1\}^n$. If $X \subseteq \{0,1\}^n$ and $M : X \to [0,1]$ is a measure we write $|M|_X$ to denote $\sum_{x \in X} M(x)$ and $\mu_X(M)$ to denote $\frac{|M|_X}{|X|}$. We write $\mathcal{U}_X$ to denote the uniform distribution on X.

The following generalization of Theorem 8.25 is easily seen to follow from the arguments of Section 8.4.3.

Theorem 8.26 *Fix $X \subseteq \{0,1\}^n$ such that $|X| \geq 2^n/2$. Let f be ϵ-hard for circuits of size g under $\mathcal{U}_X$ where $g = O\left(\frac{2^n}{n} \cdot \frac{\epsilon^2}{\log 1/\epsilon}\right)$ and let $0 < \gamma < 1/2$. Then there is a set $S \subset X$ with $|S| = \Omega(\epsilon(\log(1/\epsilon))^{-1})|X|$ such that f is γ-hard-core on S for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1} g)$.*

Proof: The only place where the domain $\{0,1\}^n$ is used in the proof of Theorem 8.25 is in the value 2^n which occurs in the Hoeffding tail bounds of Lemma 8.24. It is straightforward to verify that there is enough slack in these bounds for them to still go through with X in place of $\{0,1\}^n$ as long as $|X| \geq 2^n/2$. ∎

We need the following easy lemma:

Lemma 8.27 *If $X \subseteq \{0,1\}^n$ satisfies $|X| \geq (1 - \frac{\epsilon}{2})2^n$ and f is ϵ-hard for circuits of size g under $\mathcal{U}$ then f is $\frac{\epsilon}{2}$-hard for circuits of size g under $\mathcal{U}_X$.*

Proof: For any circuit C with $|C| \leq g$ we have that $|\{x : C(x) = f(x)\}| \leq (1-\epsilon)2^n$. Thus $\Pr_{\mathcal{U}_X}[C(x) = f(x)] \leq \frac{1-\epsilon}{1-\epsilon/2} < 1 - \frac{\epsilon}{2}$. ∎

Now we prove our strongest hard-core set construction.

Theorem 8.28 *Let f be ϵ-hard for circuits of size g under $\mathcal{U}$ where $g = O\left(\frac{2^n}{n} \cdot \frac{\epsilon^2}{\log 1/\epsilon}\right)$ and let $0 < \gamma < 1/2$. Then there is a set $S \subseteq \{0,1\}^n$ with $|S| \geq \frac{\epsilon}{2} 2^n$ such that f is γ-hard-core on S for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1} g)$.*

Proof: Theorem 8.25 implies that there is a set S_0 with $|S_0| = \Omega(\epsilon(\log 1/\epsilon)^{-1})2^n$ such that f is γ-hard-core on S_0 for size $g' = O(\gamma^2(\log 1/\epsilon)^{-1} g)$. If $|S_0| \geq \frac{\epsilon}{2}2^n$ we are done, so we assume that $|S_0| < \frac{\epsilon}{2}$ and let $X_1 = \{0,1\}^n \setminus S_0$. Lemma 8.27 implies that f is $\frac{\epsilon}{2}$-hard for circuits of size g under $\mathcal{U}_{X_1}$. Theorem 8.26 now implies the existence of a set $S_1 \subset X_1$ with $|S_1| = \Omega(\epsilon(\log 1/\epsilon)^{-1})|X_1|$ such that f is γ-hard-core on S_1 for size $g' = O(\gamma^2(\log(1/\epsilon))^{-1} g)$. Now let $X_2 = X_1 \setminus S_1$. Continue in this fashion, obtaining S_i from X_i as above, until $\sum |S_i| \geq \frac{\epsilon}{2}2^n$.

To see that this works, notice that until $\sum |S_i| \geq \frac{\epsilon}{2}2^n$ each set X_i satisfies $|X_i| > (1 - \frac{\epsilon}{2})2^n$. Thus by Lemma 8.27 at each stage we have that f is $\frac{\epsilon}{2}$-hard for size g under $\mathcal{U}_{X_i}$, so we can apply Theorem 8.26 each time with hardness parameter $\epsilon/2$. Furthermore, we have that each $|S_i| = \Omega(\epsilon(\log(1/\epsilon))^{-1})|X_i|$ where $|X_i| > (1 - \frac{\epsilon}{2})2^n$, so we must achieve $\sum |S_i| > \frac{\epsilon}{2}2^n$ after at most $O(\log 1/\epsilon)$ stages.

Now let $S = \cup_i S_i$. Since the S_i are disjoint we have $|S| \geq \frac{\epsilon}{2}2^n$. Moreover, since f is γ-hard-core on each $|S_i|$ for size g', for any circuit C of size at most g' we have

$$\Pr_{\mathcal{U}_S}[C(x) = f(x)] \; = \; \sum_i \left(\frac{|S_i|}{|S|} \Pr_{\mathcal{U}_{S_i}}[C(x) = f(x)] \right) \leq \frac{1}{2} + \gamma.$$

Thus f is γ-hard-core on S for size g' and the theorem is proved. ■

The reader may have noticed that the two arguments used to go from a hard-core measure to a hard-core set and to increase the size of the hard-core set do not depend in any essential way on the fact that the initial hard-core measure was of size $\mu(M) = \Omega(\epsilon(\log 1/\epsilon)^{-1})$. We could also have obtained the same final result – a hard-core set construction with $\epsilon/2$ as the set size parameter and $O(\gamma^2(\log 1/\epsilon)^{-1})$ as the circuit size parameter – by using the boosting algorithm `F1` which implies the existence of a measure of size $\Omega(\epsilon^3)$ (Theorem 8.16). We introduced and analyzed the $\mathtt{B}_{\text{Comb}}$ algorithm in part because this boosting algorithm will play an important role in the learning theory results of Section 8.5.

8.4.6 Optimal Hard-Core Set Construction

Freund has shown that any algorithm which boosts a $(1/2-\gamma)$-approximate weak learner to accuracy $1-\epsilon$ must combine at least $\Omega(\gamma^{-2}\log(1/\epsilon))$ weak hypotheses in the worst case (Freund, 1995). Thus, for any hard-core set construction falling within this framework our circuit size parameter is optimal up to constant factors. Furthermore, it is easy to see that any general hard-core set construction such as those we have given must have a set size parameter of at most $O(\epsilon)$, since the original ϵ-hard function f might be very easy to compute (e.g. constant) on a $1-O(\epsilon)$ fraction of inputs. Thus we believe that the hard-core set construction given by Theorem 8.28 is optimal up to constant factors with respect to both the circuit size and set size parameters.

8.4.7 A Boosting Algorithm from IHA

We have not yet described just how boosting algorithms manage to simulate the different distributions $\mathcal{D}_i$ for the example oracles $\mathrm{EX}(f,\mathcal{D}_i)$ which are required by the weak learning algorithm at each boosting stage. There are two different types of boosting algorithms, known as *boosting-by-filtering* and *boosting-by-sampling*, which handle this issue in different ways. In boosting-by-filtering the distribution $\mathcal{D}_i$ is simulated from $\mathcal{D}$ by filtering examples received from $\mathrm{EX}(f,\mathcal{D})$. If the filtering process accepts example x with probability $\alpha(x)$ and rejects x (i.e. discards x and makes another call to $\mathrm{EX}(f,\mathcal{D})$) with probability $1-\alpha(x)$, then it is easy to see that this filtering process defines a new distribution $\mathcal{D}'$ where

$$\mathcal{D}'(x) = \frac{\alpha(x)\mathcal{D}(x)}{\sum_y \alpha(y)\mathcal{D}(y)}. \tag{8.1}$$

The boosting algorithms $\mathtt{F1}$, $\mathtt{B}_{\mathrm{Filt}}$ and $\mathtt{B}_{\mathrm{Comb}}$ all are examples of boosting-by-filtering algorithms.

In boosting-by-sampling, on the other hand, a set S of examples is drawn from $\mathrm{EX}(f,\mathcal{D})$ once and for all at the beginning of the boosting process and the initial distribution $\mathcal{D}_0$ is taken to be the uniform distribution over S. Subsequent distributions $\mathcal{D}_i$ are nonzero only on the points of S, and the final hypothesis h generated by boosting has high accuracy with respect to the uniform distribution over S. Well-known results on generalization error (Blumer *et al.*, 1989) imply that if h belongs to a concept class with bounded Vapnik-Chervonenkis dimension, then for sufficiently large S any hypothesis h which is correct on all of S will with high probability have low error under $\mathcal{D}$. The `AdaBoost` algorithm of (Freund and Schapire, 1997) is an example of a boosting-by-sampling algorithm.

Impagliazzo's proof shows that it is possible to use `IHA` directly as a $O(1/\epsilon^2\gamma^2)$-stage, $1/\epsilon$-smooth boosting-by-sampling algorithm. In Chapter 9 we will give an improved boosting-by-sampling algorithm which is also $1/\epsilon$-smooth but uses only $O(1/\epsilon\gamma^2)$ stages. In the next section we discuss using `IHA` as a boosting-by-filtering algorithm in the case where the initial distribution $\mathcal{D}$ is uniform over $\{0,1\}^n$.

8.5 Faster Algorithms for Learning DNF

We have seen that boosting algorithms can be used to improve on previous complexity-theoretic hard-core set constructions. Now we go in the opposite direction and use ideas from hard-core set construction to establish new results in learning theory. We show that the uniform distribution boosting algorithm which is implicit in `IHA` can be used to improve the running time of Jackson's Harmonic Sieve algorithm for learning DNF under the uniform distribution using membership queries. This algorithm is widely viewed as one of the most important results in computational learning theory. We also show how a different modification inspired by our analysis in Section 8.4.3 can improve the running time even further at the cost of learning using more complex hypotheses.

Bshouty, Jackson and Tamon (Bshouty *et al.*, 1999) have recently given a variant of the Harmonic Sieve which runs substantially faster than the original algorithm. Their improvement is obtained by speeding up a weak learning algorithm which is a component of the Harmonic Sieve, and is orthogonal to our improvements of the boosting component of the Sieve. As described below, by combining our techniques with their improvements we obtain the fastest known algorithm for learning DNF under the uniform distribution with membership queries.

8.5.1 The DNF Learning Problem

Recall that a *disjunctive normal form* (DNF) expression is a disjunction of terms where each term is a conjunction of Boolean literals. Since every Boolean function can be expressed as a DNF, the concept class DNF is the class of all Boolean functions over $\{0,1\}^n$. The *DNF-size* of a function f is the minimum number of terms in any DNF expression for f. Thus an efficient learning algorithm for the concept class DNF must be able to learn any Boolean function in time polynomial in the number of terms in its smallest DNF representation.

In his seminal paper Valiant posed the question of whether there is an efficient PAC learning algorithm for DNF (Valiant, 1984). We saw in Chapter 3 that there is a PAC learning algorithm which learns to accuracy ϵ and runs in time $2^{O(n^{1/3}\log n \log s)}/\epsilon$ for target concepts of DNF-size s, but it is not yet known whether there is an algorithm which runs in time $\mathrm{poly}(n, s, 1/\epsilon)$. If the learning scenario is suitably modified, though, then efficient learning of DNF becomes possible. In a breakthrough result several years ago Jackson gave the Harmonic Sieve algorithm which uses *membership queries* to learn DNF to accuracy ϵ under the uniform distribution in roughly ns^8/ϵ^{12} time steps (Jackson, 1997). A membership query is an oracle query in which the learner specifies a point x and the membership oracle $\mathrm{MEM}(f)$ returns the value $f(x)$.

Although the Harmonic Sieve runs in polynomial time, it is not considered to be computationally practical due to the high degree of the polynomial time bound. We show how to substantially improve the algorithm's time dependence on the error parameter ϵ, thus making progress towards a more efficient implementation.

8.5.2 The Harmonic Sieve

The main result of (Jackson, 1997) is the following theorem:

Theorem 8.29 (Jackson, 1997) *Let f be a Boolean function on $\{0,1\}^n$ of DNF-size s and let $0 < \epsilon, \delta < 1$. For any constant $\rho > 0$, given access to a membership oracle $MEM(f)$ the Harmonic Sieve algorithm runs in time $\tilde{O}(ns^8/\epsilon^{12+\rho})$ and with probability $1-\delta$ outputs a hypothesis h such that $\Pr_{\mathcal{U}}[h(x) \neq f(x)] \leq \epsilon$.*

At the heart of Jackson's Harmonic Sieve algorithm is a procedure `WDNF` which was first studied in (Blum *et al.*, 1994). The `WDNF` algorithm takes as input an example oracle $\mathrm{EX}(f, \mathcal{D})$, a membership oracle $\mathrm{MEM}(f)$, a distribution oracle $\mathrm{DIST}(\mathcal{D})$ and a value $\delta > 0$. A distribution oracle $\mathrm{DIST}(\mathcal{D})$ is an oracle which, when queried with a point x in the domain of $\mathcal{D}$, returns the value of $\mathcal{D}(x)$. With probability at least $1-\delta$ the `WDNF` algorithm outputs a parity function which is a $(1/2 - \Omega(1/s))$-approximator for f under $\mathcal{D}$, where s is the DNF-size of f.

The Harmonic Sieve algorithm works by running Freund's boosting algorithm `F1` with `WDNF` as the weak learning algorithm. At the i-th stage of boosting the `F1` boosting algorithm simulates some distribution $\mathcal{D}_i$ and uses the simulated example oracle $EX(f, \mathcal{D}_i)$ for `WDNF`. Jackson shows that the `F1` boosting algorithm constructs each distribution $\mathcal{D}_i$ in such a way that after a "one-time" initial cost of $\tilde{O}(L_\infty(2^n\mathcal{D}_i)^6)$ time steps (to estimate the scaling factor in the denominator of Equation 8.1) for each $\mathcal{D}_i$, it is possible to simulate a constant-factor approximation $\mathrm{DIST}'(\mathcal{D}_i)$ of $\mathrm{DIST}(\mathcal{D}_i)$ in $O(n)$ time steps per call.[2] The following lemma is a direct consequence of Jackson's Lemma 9 and the analysis used in its proof:

Lemma 8.30 (Jackson, 1997) *Let f be any Boolean function of DNF-size s over $\{0,1\}^n$ and let $\mathcal{D}$ be any distribution over $\{0,1\}^n$. If* `WDNF` *is run using $EX(f,\mathcal{D})$, $MEM(f)$ and $DIST'(\mathcal{D})$ as its oracles, where each call to $EX(f,\mathcal{D})$ takes t time steps, each call to $MEM(f)$ takes 1 time step, and the time requirements of $DIST'(\mathcal{D})$ are as described above, then* `WDNF` *runs in time $\tilde{O}(s^4(L_\infty(2^n\mathcal{D}))^2 t + ns^6(L_\infty(2^n\mathcal{D}))^6)$ time steps and with probability at least $1-\delta$ outputs a parity function which is a $(1/2-\Omega(1/s))$-approximator to f under $\mathcal{D}$.*

Jackson proves that each call to $\mathrm{EX}(f, \mathcal{D}_i)$ made by `WDNF` can be simulated in time $\tilde{O}(nL_\infty(2^n\mathcal{D}_i)^3)$ with high probability. Consequently the time bound of Lemma 8.30 is dominated by the $ns^6(L_\infty(2^n\mathcal{D}))^6$ term, and thus the time required for the i-th execution of `WDNF` on distribution $\mathcal{D}_i$ is bounded by $\tilde{O}(ns^6(L_\infty(2^n\mathcal{D}))^6)$. As noted in Section 8.4.3, for any $\rho > 0$ algorithm `F1` can be shown to be an $O(\gamma^{-2}\log(1/\epsilon))$-stage, $O(1/\epsilon^{2+\rho})$-smooth boosting algorithm, so $L_\infty(2^n\mathcal{D}_i) = O(1/\epsilon^{2+\rho})$ for every distribution $\mathcal{D}_i$ which `WDNF` uses. Since $\gamma = \Omega(1/s)$ for the `WDNF` algorithm, all in all the Harmonic Sieve algorithm runs in time $\tilde{O}(ns^8/\epsilon^{12+\rho})$ for any $\rho > 0$. The hypotheses output by the Harmonic Sieve are majority-of-parity circuits since each

[2] To be more precise, the oracle $\mathrm{DIST}'(\mathcal{D}_i)$ is such that $\frac{1}{2}\mathrm{DIST}(\mathcal{D}_i)(x) \leq \mathrm{DIST}'(\mathcal{D}_i)(x) \leq \frac{3}{2}\mathrm{DIST}(\mathcal{D}_i)(x)$.

weak hypothesis is a parity circuit and F1's hypothesis is a majority circuit over weak hypotheses.

8.5.3 A Faster Version of the Harmonic Sieve

As described above, the Harmonic Sieve algorithm works by boosting under the uniform distribution and its running time strongly depends on the smoothness of the boosting algorithm. The following observation follows directly from the discussion of IHA in Section 8.2:

Observation 8.31 *For each measure M_i constructed in the execution of* IHA *the distribution $\mathcal{D}_{M_i}$ is $1/\epsilon$-smooth.*

This is substantially better than the distributions constructed by F1 which are guaranteed only to be $(1/\epsilon^{2+\rho})$-smooth. Thus, it appears that we can use the better boundedness of the IHA algorithm to obtain a faster version of the Harmonic Sieve, and indeed this turns out to be the case; however some details need to be addressed as described below.

One detail is that since the running time of WDNF depends on the quantity $L_\infty(2^n \mathcal{D}_i)$, we need a version of IHA which works efficiently over the entire domain $\{0,1\}^n$. Another way of saying this is that we need a boost-by-filtering version of IHA. Doing an exact computation of $\mu(M_i)$ in line 3 of IHA would take exponential time for the domain $\{0,1\}^n$, so our boost-by-sampling version of IHA instead estimates the value of $\mu(M_i)$ by using a sample average. More precisely, the algorithm draws a collection of uniformly distributed x's from $\{0,1\}^n$ and uses the observed average value of $M_i(x)$ on this sample as its estimate for $\mu(M_i)$. It is easy to see that $\mu(M_i)$ is the expected value of $M_i(x)$ for uniformly chosen x; standard bounds on sampling (e.g. Corollary 2.2 of (Jackson, 1995)) show that with very high probability an estimate $\mu'(M_i)$ satisfying $\frac{1}{2}\mu(M_i) \le \mu'(M_i) \le \frac{3}{2}\mu(M_i)$ can be found using $\tilde{O}(1/\mu(M_i)^2)$ samples. Thus in the boost-by-sampling version of IHA the test in line 3 can be approximately performed in $\tilde{O}(1/\epsilon^2)$ time steps, and the resulting algorithm will be $2/\epsilon$-smooth with high probability. We note that the $\tilde{O}(1/\mu(M_i)^2)$ time steps required to perform this estimation for each measure M_i corresponds exactly to the "one-time cost" of $\tilde{O}(L_\infty(2^n\mathcal{D}_i)^6)$ for estimating the denominator of Equation 8.1 which was required for each distribution $\mathcal{D}_i$ simulated by F1.

Another detail which must be taken into account is the slowdown incurred in the process of filtering examples from EX(f,$\mathcal{U}$) to simulate draws from $EX(f, \mathcal{D}_i)$. It is not difficult to see that if a measure M_i satisfies $\mu(M_i) \geq \epsilon/2$, then on average at most $2/\epsilon$ draws from$EX(f, \mathcal{U})$ are required to simulate a draw from EX($f, \mathcal{D}_{M_i}$), and with very high probability no more than $\tilde{O}(1/\epsilon)$ draws are required. Thus this slowdown, which is precisely the factor t in Lemma 8.30, is at most $\tilde{O}(1/\epsilon)$.

Thus `IHA` can be translated into a $O(1/\gamma^2\epsilon^2)$-stage, $O(1/\epsilon)$-smooth boost-by-filtering algorithm under the uniform distribution. This boosting algorithm can be used instead of `F1` in the top layer of the Harmonic Sieve; we call this modified algorithm `HS'`. Moreover, as discussed above Lemma 8.30 implies that each execution of `WDNF` in `HS'`will take at most $\tilde{O}(ns^6/\epsilon^6)$ time steps. Thus, although `HS'` requires a factor of $\tilde{\Omega}(1/\epsilon^2)$ more boosting stages than the original Sieve, this disadvantage is more than offset by the improved runtime of `WDNF`. We obtain the following:

Theorem 8.32 *There is a membership-query algorithm* `HS'` *for learning DNF under the uniform distribution which runs in time* $\tilde{O}(ns^8/\epsilon^8)$. *The algorithm outputs as its final hypothesis a majority-of-parity circuit.*

We can achieve an even faster variant of the Harmonic Sieve, at the price of using more complex hypotheses, by using the $\mathtt{B}_{\text{comb}}$ boosting algorithm instead of the boost-by-filtering `IHA` algorithm. As noted in Section 8.4.2, $\mathtt{B}_{\text{Comb}}$ is an $O(\gamma^{-2}\log(1/\epsilon))$-stage, $O(\log(1/\epsilon)/\epsilon)$-smooth boost-by-filtering algorithm. The smoothness of $\mathtt{B}_{\text{Comb}}$ implies that the "one-time cost" of computing the scaling factor for each distribution $\mathcal{D}_i$ is now $\tilde{O}(L_\infty(2^n\mathcal{D}_i)^2) = \tilde{O}(1/\epsilon^2)$. Furthermore, the time overhead factor for simulating each call to $EX(f, \mathcal{D}_i)$ is now $t = \tilde{O}(L_\infty(2^n\mathcal{D}_i)) = \tilde{O}(1/\epsilon)$. From Lemma 8.30 we find that if $\mathtt{B}_{\text{Comb}}$ is used as the boosting algorithm in the Sieve, then the running time of each boosting stage will be at most $\tilde{O}(ns^6/\epsilon^6)$. Since we boost for at most $O(s^2\log(1/\epsilon))$ stages under $\mathtt{B}_{\text{comb}}$, we have the following theorem:

Theorem 8.33 *There is a membership-query algorithm for learning s-term DNF formulae under the uniform distribution on* $\{0,1\}^n$ *which runs in time* $\tilde{O}(ns^8/\epsilon^6)$. *The algorithm outputs as its final hypothesis a majority-of-majority-of-parity circuit.*

The additional circuit complexity comes from the fact that the hypothesis output by $\mathtt{B}_{\text{Comb}}$ is a depth 2 majority circuit over its inputs.

8.5.4 Extensions

Throughout this section we have only discussed using the Harmonic Sieve to learn DNF formulae under the uniform distribution. Jackson generalizes the algorithm to several other concept classes including TOP (polynomial-size depth-2 circuits where the top gate computes majority and the bottom gates compute parities) and unions of axis-parallel rectangles over $\{0, 1, \ldots, b\}^n$ for constant b. In each case our approach can be used to improve the running time of Jackson's algorithms.

In recent work Bshouty, Jackson and Tamon (Bshouty *et al.*, 1999) have given a new version of the Harmonic Sieve for learning DNF under the uniform distribution. The new algorithm differs from the original Harmonic Sieve in that it uses a faster version of the `WDNF` algorithm. Implicit in their analysis is the following lemma:

Lemma 8.34 (Bshouty *et al.*, 1999) *Let f be any Boolean function of DNF-size s over $\{0,1\}^n$ and let $\mathcal{D}$ be any distribution over $\{0,1\}^n$. There is an algorithm* `WDNF′` *which takes as input an example oracle $EX(f, \mathcal{D})$, a membership oracle $MEM(f)$, a distribution oracle $DIST'(\mathcal{D})$, and a value $\delta > 0$. In the context of the Harmonic Sieve, each invocation of* `WDNF′` *with the i-th distribution $\mathcal{D}_i$ generated by the boosting algorithm takes $\tilde{O}(ns^2(L_\infty(2^n\mathcal{D}_i))^2)$ time steps and and with probability at least $1 - \delta$ outputs a parity function which is a $(1/2 - \Omega(1/s))$-approximator to f under $\mathcal{D}_i$.*

The Harmonic Sieve variant described in (Bshouty *et al.*, 1999) runs the original $O(1/\epsilon^{2+\rho})$-smooth `F1` boosting algorithm for $\tilde{O}(s^2)$ stages, using the new `WDNF′` algorithm as the weak learner, to obtain an overall running time of essentially $\tilde{O}(ns^4/\epsilon^4)$. By instead using the $O(\log(1/\epsilon)/\gamma^2)$-stage, $O(\log(1/\epsilon)/\epsilon)$-smooth boosting algorithm $\mathtt{B}_{\mathrm{comb}}$ as in Section 8.5.3, we obtain the following result, which is the fastest known algorithm for learning DNF under the uniform distribution using membership queries:

Theorem 8.35 *There is a membership-query algorithm for learning s-term DNF formulae over $\{0,1\}^n$ under the uniform distribution which runs in time $\tilde{O}(ns^4/\epsilon^2)$. The algorithm outputs as its final hypothesis a majority-of-majority-of-parity circuit.*

Chapter 9

Smooth Boosting and Learning with Malicious Noise

In this chapter we combine ideas from Chapters 7 and 8 to obtain noise-tolerant boosting-based algorithms for PAC learning linear threshold functions.

Motivated by our investigation of boosting algorithms in Chapter 8, we first describe a new boosting algorithm called `SmoothBoost`. Like Impagliazzo's algorithm for hard-core set construction, `Smoothboost` generates only smooth distributions which do not assign too much weight to any single example; however `SmoothBoost` is significantly faster and more flexible than the boosting algorithm which is implicit in Impagliazzo's algorithm.

We then use `SmoothBoost` to construct malicious noise tolerant versions of the PAC model p-norm linear threshold learning algorithms described in Chapter 7. In a noise-free setting the sample complexity bounds of the new algorithms match the bounds of the algorithms given in Chapter 7. However, the new algorithms can tolerate relatively high levels of malicious noise, a property which the algorithms of Chapter 7 do not appear to share. We show that the noise tolerance and sample complexity of these new PAC algorithms closely correspond to known bounds for the online p-norm algorithms of Grove, Littlestone and Schuurmans (Grove *et al.*, 1997) and Gentile and Littlestone (Gentile and Littlestone, 1999). As special cases of our new algorithms we obtain linear threshold learning algorithms which match the sample complexity *and* malicious noise tolerance of the online Perceptron and Winnow algorithms.

9.1 Introduction

Any realistic model of learning from examples must address the issue of noisy data. In 1985 Valiant extended the original PAC learning model to allow for the possibility of *malicious noise*. This is a worst-case model of errors in which some fraction of the labeled examples given to a learning algorithm may be corrupted by an adversary who can modify both example points and labels in an arbitrary fashion (a detailed description of the model is given in Section 9.3). The frequency of such corrupted examples is known as the *malicious noise rate*.

Learning in the presence of malicious noise is known to be quite difficult. Kearns and Li (Kearns and Li, 1993) have shown that for many concept classes it is information-theoretically impossible to learn to accuracy ϵ if the malicious noise rate exceeds $\frac{\epsilon}{1+\epsilon}$. In fact, for many interesting concept classes (such as the class of linear threshold functions), the best efficient algorithms known can only tolerate malicious noise rates which are significantly lower than this general upper bound. Despite these difficulties, the practical importance of being able to cope with noisy data has led many researchers to study PAC learning in the presence of malicious noise; see e.g. (Aslam and Decatur, 1998; Auer, 1997; Auer and Cesa-Bianchi, 1998; Cesa-Bianchi *et al.*, 1999; Decatur, 1993; Mansour and Parnas, 1998).

In this chapter we describe a new *smooth boosting* algorithm and use this smooth boosting algorithm to construct a family of PAC algorithms for learning linear threshold functions in the presence of malicious noise.

9.1.1 Motivation for Smooth Boosting

Our basic approach is quite simple, as illustrated by the following example. Consider a learning scenario in which we have a weak learning algorithm L which takes as input a finite sample S of m labeled examples. Algorithm L is known to have some tolerance to malicious noise; specifically, L is guaranteed to generate a hypothesis with nonnegligible advantage provided that the frequency of noisy examples in its sample is at most 10%. We would like to learn to high accuracy in the presence of malicious noise at a rate of 1%.

The obvious approach in this setting is to use a boosting algorithm, which will generate some sequence of distributions $\mathcal{D}_1, \mathcal{D}_2, \ldots$ over S. This approach can fail, though, if the boosting algorithm generates distributions which are very skewed from the uniform distribution on S; if distribution $\mathcal{D}_i$ assigns probabilities as large as $\frac{20}{m}$ to individual points in S, for instance, then the frequency of noisy examples for L in stage i could be as high as 20%. What we need instead is a *smooth* boosting algorithm which only constructs distributions $\mathcal{D}_i$ over S which never assign weight greater than $\frac{10}{m}$ to any single example. By using such a smooth booster we are assured that the weak learner will function successfully at each stage, so the overall boosting process will work correctly.

While the setting described above is artificial, we note here that indirect empirical evidence has been given supporting the smooth boosting approach for noisy settings. It is well known (Dietterich, 2000; Schapire, 1999b) that commonly used boosting algorithms such as `AdaBoost` can perform poorly on noisy data. Dietterich has suggested that this poor performance is due to `AdaBoost`'s tendency to generate very skewed distributions which put a great deal of weight on a few noisy examples. This overweighting of noisy examples cannot occur under a smooth boosting regimen.

In Section 9.2 we give a simple new boosting algorithm, `SmoothBoost`, which is guaranteed to generate only smooth distributions as described above. In fact, we show in Section 9.4.2 that the distributions generated by `SmoothBoost` are optimally smooth.

`SmoothBoost` is not the first boosting algorithm which attempts to avoid the skewed distributions of `AdaBoost`. In addition to the boosting algorithm implicit in Impagliazzo's hard-core set construction algorithm (Impagliazzo, 1995), a boosting algorithm with a smoothness guarantee similar to that of `SmoothBoost` has been given by Domingo and Watanabe (Domingo and Watanabe, 2000). We saw in Chapter 8 that Freund's $\mathtt{B}_{\mathrm{Comb}}$ boosting algorithm (Freund, 1992) generates relatively smooth distributions, and more recently Freund (Freund, 1999) has also described a new boosting algorithm which uses a more moderate weighting scheme than `AdaBoost`. In Section 9.2.3 we show that in addition to generating smooth distributions, our `SmoothBoost` algorithm has several other desirable properties, such as constructing a large margin final hypothesis and using real-valued weak hypotheses, which are essential for the noisy linear threshold learning application of Section 9.3. We discuss the relationship between `SmoothBoost` and the algorithms of Domingo and Watanabe, Freund, and Impagliazzo in Section 9.2.4.

9.1.2 Learning Linear Threshold Functions with Malicious Noise

We use the `SmoothBoost` algorithm in Section 9.3 to construct a family of PAC-model malicious noise tolerant algorithms for learning linear threshold functions. Our new family of algorithms is quite similar to the family constructed in Chapter 7, except that now we use `SmoothBoost` instead of `AdaBoost` as the boosting component. We showed in Chapter 7 that for linearly separable (noise-free) data, the `AdaBoost`-based algorithms have sample complexity bounds which are essentially identical to those of the online p-norm linear threshold learning algorithms of Grove, Littlestone and Schuurmans (Grove *et al.*, 1997). As described in Chapter 7, these online algorithms include as special cases ($p = 2$ and $p = \infty$) the well-known online Perceptron and Winnow algorithms.

Gentile and Littlestone (Gentile and Littlestone, 1999) have given mistake bounds for the online p-norm algorithms when run on examples which are not linearly separable, thus generalizing previous bounds on noise tolerance for Perceptron (Freund and Schapire, 1998) and Winnow (Littlestone, 1991). A drawback of our `AdaBoost`-based PAC-model p-norm algorithms of Chapter 7 is that they do not appear to succeed in the presence of malicious noise for the reasons described above. We show in Section 9.4 that for all values $2 \leq p \leq \infty$, our new PAC algorithms which use `SmoothBoost` match both the sample complexity *and* the malicious noise tolerance of the online p-norm algorithms. Our construction thus provides malicious noise tolerant PAC analogues of Perceptron and Winnow (and many other algorithms as well).

9.2 Smooth Boosting with `SmoothBoost`

In this section we describe and analyze the `SmoothBoost` algorithm. We show that `SmoothBoost` has several useful properties:

- it only constructs smooth distributions which do not put too much weight on any single example;
- it can be used to generate a large margin final hypothesis;
- it can be used with a weak learning algorithm which outputs real-valued hy-

potheses;

- it constructs its final hypothesis by thresholding a weighted sum of weak hypotheses.

All of these properties are essential for the noisy linear threshold learning problem we consider in Section 9.3.

9.2.1 Preliminaries

First we recall some useful notation from Chapters 7 and 8. A *measure* on a finite set is a function $M : S \to [0,1]$. We write $|M|$ to denote $\sum_{x \in S} M(x)$ and $\mu(M)$ to denote $|M|/|S|$. Given a measure M, there is a natural induced distribution $\mathcal{D}_M$ defined by $\mathcal{D}_M(x) = M(x)/|M|$. This definition yields

Observation 9.2 $L_\infty(\mathcal{D}_M) \leq \frac{1}{|M|} = \frac{1}{\mu(M)|S|}$.

Let $\mathcal{D}$ be a distribution over a set $S = \langle x^1, y_1 \rangle, \ldots, \langle x^m, y_m \rangle$ of labeled examples and let h be a real-valued function which maps $\{x^1, \ldots, x^m\}$ into $[-1,1]$. If

$$\frac{1}{2} \sum_{j=1}^{m} \mathcal{D}(j) |h(x^j) - y_j| \leq \frac{1}{2} - \gamma$$

then we say that the *advantage* of h under $\mathcal{D}$ is γ. As in Chapter 7 we say that an algorithm which takes S and $\mathcal{D}$ as input and outputs an h which has advantage at least $\gamma > 0$ is a *weak learning algorithm.*

Finally, let $g(x) = \text{sign}(f(x))$ where $f : X \to [-1,1]$ is a real-valued function. As in Section 7.4.2 we say that the *margin* of g on a labeled example $\langle x, y \rangle \in X \times \{-1,1\}$ is $yf(x)$; intuitively this is the amount by which g predicts y correctly. Note that the margin of g on $\langle x, y \rangle$ is nonnegative if and only if g predicts y correctly.

9.2.2 The `Smoothboost` Algorithm

The `Smoothboost` algorithm is given in Figure 9.2.1 The parameter κ is the desired error rate of the final hypothesis, the parameter γ is the guaranteed advantage of the hypotheses returned by the weak learner, and θ is the desired margin of the final hypothesis. `Smoothboost` runs the weak learning algorithm several times on

Input: real numbers $0 < \kappa < 1$, $0 \le \theta \le \gamma < \frac{1}{2}$
sample $S = \langle x^1, y_1 \rangle, \ldots, \langle x^m, y_m \rangle$ where each $y_i \in \{-1, 1\}$
weak learner `WL` which takes input $(S, \mathcal{D}_t)$ and outputs
$h_t : \{x^1, \ldots, x^m\} \to [-1, 1]$

Output: hypothesis $h(x) = \text{sign}(f(x))$

1. **forall** $j = 1, \ldots, m$ **set** $M_1(j) = 1$
2. **forall** $j = 1, \ldots, m$ **set** $N_0(j) = 0$
3. **set** $t = 1$
4. **until** $\mu(M) < \kappa$ **do**
5. **forall** $j = 1, \ldots, m$ **set** $\mathcal{D}_t(j) = M_t(j)/|M_t|$
6. run $\texttt{WL}(S, \mathcal{D}_t)$ to get h_t such that $\frac{1}{2}\sum_{j=1}^{m} \mathcal{D}_t(j)|h_t(x^j) - y_j| \le \frac{1}{2} - \gamma$
7. **forall** $j = 1, \ldots, m$ **set** $N_t(j) = N_{t-1}(j) + y_j h_t(x^j) - \theta$
8. **forall** $j = 1, \ldots, m$ **set** $M_{t+1}(j) = \begin{cases} 1 & \text{if } N_t(j) < 0 \\ (1-\gamma)^{N_t(j)/2} & \text{if } N_t(j) \ge 0 \end{cases}$
9. **set** $t = t + 1$
10. **set** $T = t - 1$
11. **return** $h = \text{sign}(f(x))$ where $f(x) = \frac{1}{T}\sum_{i=1}^{T} h_i(x)$

Figure 9.1: The `SmoothBoost` algorithm.

a sequence of carefully constructed distributions and outputs a thresholded sum of the hypotheses thus generated. The quantity $N_t(j)$ in line 7 may be viewed as the cumulative amount by which the hypotheses $h_1, \ldots, h_t$ exceed the desired margin θ on the labeled example $\langle x^j, y_j \rangle$. The measure M_{t+1} assigns exponentially less weight to examples where N_t is large, thus forcing the weak learner to focus in stage $t+1$ on examples where previous hypotheses have done poorly. Note that since any measure maps into $[0, 1]$ there is a strict bound on the amount of weight which can be assigned to any example.

9.2.3 Proof of Correctness of `SmoothBoost`

Several useful properties of the `Smoothboost` algorithm are easy to verify. The algorithm is called `Smoothboost` because each distribution it constructs is guaranteed to be "smooth" in the sense that no single point receives too much weight:

Claim 9.3 *Each distribution $\mathcal{D}_t$ defined in step 5 of* `Smoothboost` *has $L_\infty(\mathcal{D}_t) \leq \frac{1}{\kappa m}$.*

Proof: Follows directly from Observation 9.2 and the condition in line 4. ∎

Another useful property is that the final hypothesis h has margin at least θ on all but a κ fraction of the points in S :

Theorem 9.4 *If* `Smoothboost` *terminates then f satisfies $\frac{|\{j \,:\, y_j f(x^j) \leq \theta\}|}{m} < \kappa$.*

Proof: Since $N_T(j) = T(y_j f(x^j) - \theta)$, if $y_j f(x^j) \leq \theta$ then $N_T(j) \leq 0$ and hence $M_{T+1}(j) = 1$. Consequently we have

$$\frac{|\{j \;:\; y_j f(x^j) \leq \theta\}|}{m} \leq \frac{\sum_{j=1}^{m} M_{T+1}(j)}{m} = \mu(M_{T+1}) < \kappa$$

by the condition in line 4. ∎

Note that since $\theta \geq 0$ Theorem 9.4 implies that the final `Smoothboost` hypothesis is correct on all but a κ fraction of S.

Finally we must show that the algorithm terminates in a reasonable amount of time. The following theorem bounds the number of times that `SmoothBoost` will execute its main loop:

Theorem 9.5 *If each hypothesis h_t returned by* `WL` *in line 6 has advantage at least γ under $\mathcal{D}_t$ (i.e. satisfies the condition of line 6) and θ is set to $\frac{\gamma}{2+\gamma}$, then* `SmoothBoost` *terminates with $T < \frac{2}{\kappa\gamma^2\sqrt{1-\gamma}}$.*

As will be evident from the proof, slightly different bounds on T can be established by choosing different values of θ in the range $[0, \gamma]$. We take $\theta = \frac{\gamma}{2+\gamma}$ in the theorem above both to obtain a margin $\theta = \Omega(\gamma)$ and to obtain a clean bound in the theorem. Theorem 9.5 follows from the bounds established in the following two lemmas:

Lemma 9.6 $\sum_{j=1}^m \sum_{t=1}^T M_t(j) y_j h_t(x^j) \geq 2\gamma \sum_{t=1}^T |M_t|$.

Lemma 9.7 *If $\theta = \frac{\gamma}{2+\gamma}$ then $\sum_{j=1}^m \sum_{t=1}^T M_t(j) y_j h_t(x^j) < \frac{2m}{\gamma\sqrt{1-\gamma}} + \gamma \sum_{t=1}^T |M_t|$.*

Combining these bounds we obtain

$$\frac{2m}{\gamma\sqrt{1-\gamma}} > \gamma \sum_{t=1}^T |M_t| \geq \gamma\kappa m T$$

where the last inequality is because $|M_t| = \mu(M_t)m \geq \kappa m$ for $t = 1, \ldots, T$.

Proof of Lemma 9.6: Since $h_t(x^j) \in [-1, 1]$ and $y_j \in \{-1, 1\}$, we have $y_j h_t(x^j) = 1 - |h_t(x^j) - y_j|$, and thus

$$\sum_{j=1}^m \mathcal{D}_t(j) y_j h_t(x^j) = \sum_{j=1}^m \mathcal{D}_t(j)(1 - |h_t(x^j) - y_j|) \geq 2\gamma.$$

This implies that

$$\sum_{j=1}^m \sum_{t=1}^T M_t(j) y_j h_t(x^j) = \sum_{t=1}^T |M_t| \sum_{j=1}^m \mathcal{D}_t(j) y_j h_t(x^j) \geq \sum_{t=1}^T 2\gamma |M_t|.$$

■

Proof of Lemma 9.7: By the definition of $N_t(j)$ we have

$$\begin{aligned}
\sum_{j=1}^m \sum_{t=1}^T M_t(j) y_j h_t(x^j) &= \sum_{j=1}^m \sum_{t=1}^T M_t(j)(N_t(j) - N_{t-1}(j) + \theta) \\
&= \theta \sum_{t=1}^T |M_t| + \sum_{t=1}^T \sum_{j=1}^m M_t(j)(N_t(j) - N_{t-1}(j)). \qquad (9.1)
\end{aligned}$$

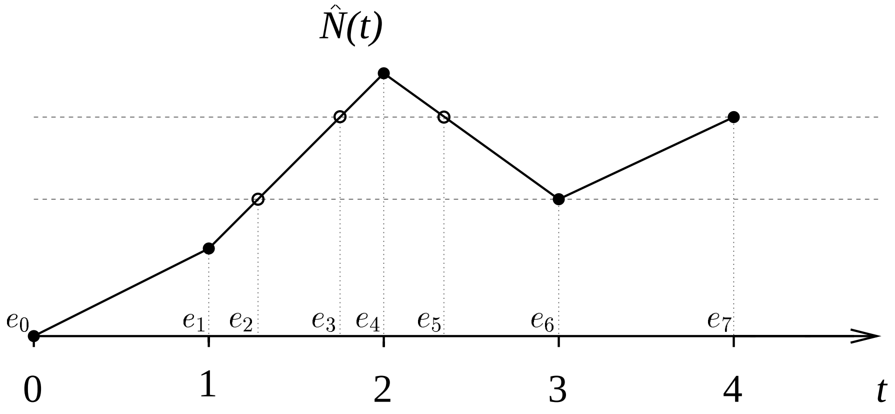


Figure 9.2: A plot of $\hat{N}$ with $T = 4$.

Note that $\hat{N}$ is piecewise linear with joins at integer values of t. A possible pairing of segments matches $[e_2, e_3]$ with $[e_5, e_6]$ and $[e_3, e_4]$ with $[e_4, e_5]$, leaving $[e_0, e_1]$, $[e_1, e_2]$ and $[e_6, e_7]$ unpaired. In this example $\hat{N}$ is increasing on each unpaired segment.

It thus suffices to show that if $\theta = \frac{\gamma}{2+\gamma}$, then for each $j = 1, \ldots, m$ we have

$$\sum_{t=1}^{T} M_t(j)(N_t(j) - N_{t-1}(j)) < \frac{2}{\gamma\sqrt{1-\gamma}} + (\gamma - \theta) \sum_{t=1}^{T} M_t(j) \tag{9.2}$$

since summing this inequality over $j = 1, \ldots, m$ and substituting into (9.1) proves the lemma. Fix any $j \in \{1, \ldots, m\}$; for ease of notation we write N_t and M_t in place of $N_t(j)$ and $M_t(j)$ for the rest of the proof.

If $N_t = N_{t-1}$ for some integer t then the term $M_t(N_t - N_{t-1})$ contributes 0 to the sum in (9.2), so without loss of generality we assume that $N_t \neq N_{t-1}$ for all integers t. We extend the sequence $(N_0, N_1, \ldots, N_T)$ to a continuous piecewise linear function $\hat{N}$ on $[0, T]$ in the obvious way, i.e. for t an integer and $\epsilon \in [0, 1]$ we let $\hat{N}(t + \epsilon) = N_t + \epsilon(N_{t+1} - N_t)$. Let

$$E = \{e \in [0, T] : \hat{N}(e) = N_t \text{ for some integer } t = 0, 1, \ldots, T\}.$$

The set E is finite so we have $0 = e_0 < e_1 \cdots < e_r = T$ with $E = \{e_0, \ldots, e_r\}$ (see Figure 9.2). Since for each integer $t \geq 1$ the interval $(t-1, t]$ must contain some e_i, we can reexpress the sum $\sum_{t=1}^{T} M_t(N_t - N_{t-1})$ as

$$\sum_{i=1}^{r} M_{\lceil e_i \rceil} \left(\hat{N}(e_i) - \hat{N}(e_{i-1}) \right). \tag{9.3}$$

We say that two segments $[e_{a-1}, e_a]$ and $[e_{b-1}, e_b]$ *match* if $\hat{N}(e_{a-1}) = \hat{N}(e_b)$ and $\hat{N}(e_{b-1}) = \hat{N}(e_a)$. For example, in Figure 9.2 the segment $[e_2, e_3]$ matches $[e_5, e_6]$ but does not match $[e_6, e_7]$. We pair up matching segments until no more pairs can be formed. Note that if any unpaired segments remain, it must be the case that either $\hat{N}$ is increasing on each unpaired segment (if $N_T > 0$) or $\hat{N}$ is decreasing on each unpaired segment (if $N_T < 0$). Now we separate the sum (9.3) into two pieces, i.e. $\sum_{i=1}^{r} M_{\lceil e_i \rceil}(\hat{N}(e_i) - \hat{N}(e_{i-1})) = P + U$, where P is the sum over all paired segments and U is the sum over all unpaired segments. We will show that $P < (\gamma - \theta)\sum_{t=1}^{T} M_t$ and $U < \frac{2}{\gamma\sqrt{1-\gamma}}$, thus proving the lemma.

First we bound P. Let $[e_{a-1}, e_a]$ and $[e_{b-1}, e_b]$ be a pair of matching segments where $\hat{N}$ is increasing on $[e_{a-1}, e_a]$ and decreasing on $[e_{b-1}, e_b]$. The contribution of these two segments to P is

$$\begin{aligned} M_{\lceil e_a \rceil}\left(\hat{N}(e_a) - \hat{N}(e_{a-1})\right) + M_{\lceil e_b \rceil}\left(\hat{N}(e_b) - \hat{N}(e_{b-1})\right) \\ = (M_{\lceil e_a \rceil} - M_{\lceil e_b \rceil})\left(\hat{N}(e_a) - \hat{N}(e_{a-1})\right). \end{aligned} \tag{9.4}$$

Since each segment $[e_{a-1}, e_a]$ is contained in $[t-1, t]$ for some integer t, we have that $\lceil e_a \rceil - 1 \le e_{a-1} < e_a \le \lceil e_a \rceil$. The linearity of $\hat{N}$ on $[\lceil e_a \rceil - 1, \lceil e_a \rceil]$ implies that

$$N_{\lceil e_a \rceil - 1} \le \hat{N}(e_{a-1}) < \hat{N}(e_a) \le N_{\lceil e_a \rceil} \le N_{\lceil e_a \rceil - 1} + 1 - \theta \tag{9.5}$$

where the last inequality is because $y_j h_t(x^j) \le 1$ in line 7 of `Smoothboost` . Similarly, we have that $\lceil e_b \rceil - 1 \le e_{b-1} < e_b \le \lceil e_b \rceil$, and hence

$$N_{\lceil e_b \rceil - 1} \ge \hat{N}(e_{b-1}) > \hat{N}(e_b) \ge N_{\lceil e_b \rceil} \ge N_{\lceil e_b \rceil - 1} - 1 - \theta. \tag{9.6}$$

Since $\hat{N}(e_a) = \hat{N}(e_{b-1})$ inequalities (9.5) and (9.6) imply that $N_{\lceil e_a \rceil - 1} \ge N_{\lceil e_b \rceil - 1} - 2$. The definition of M now implies that $M_{\lceil e_b \rceil} \ge (1-\gamma)M_{\lceil e_a \rceil}$. Since $\hat{N}(e_a) - \hat{N}(e_{a-1}) > 0$, we thus have that (9.4) is at most

$$\gamma M_{\lceil e_a \rceil}\left(\hat{N}(e_a) - \hat{N}(e_{a-1})\right) \le \gamma(1-\theta)M_{\lceil e_a \rceil}(e_a - e_{a-1}) \tag{9.7}$$

where the inequality follows from (9.5) and the linearity of $\hat{N}$ on $[e_{a-1}, e_a]$. Since $\hat{N}(e_a) - \hat{N}(e_{a-1}) = \hat{N}(e_{b-1}) - \hat{N}(e_b)$, we similarly have that (9.4) is at most

$$\begin{aligned} \gamma M_{\lceil e_a \rceil} \left(\hat{N}(e_{b-1}) - \hat{N}(e_b) \right) &\leq \frac{\gamma}{1-\gamma} M_{\lceil e_b \rceil} \left(\hat{N}(e_{b-1}) - \hat{N}(e_b) \right) \\ &\leq \frac{\gamma}{1-\gamma} (1+\theta) M_{\lceil e_b \rceil} (e_{b-1} - e_b). \end{aligned} \tag{9.8}$$

Using the fact that $\theta = \frac{\gamma}{2+\gamma}$ and some algebra, inequalities (9.7) and (9.8) imply that (9.4) is at most

$$\frac{\gamma(1+\gamma)}{2+\gamma} \left(M_{\lceil e_a \rceil}(e_a - e_{a-1}) + M_{\lceil e_b \rceil}(e_{b-1} - e_b) \right). \tag{9.9}$$

If we sum (9.9) over all pairs of matching segments the resulting quantity is an upper bound on P. In this sum, for each value of $t = 1, \ldots, T$, the coefficient of M_t will be at most $\frac{\gamma(1+\gamma)}{2+\gamma} = \gamma - \theta$. (This bound on the coefficient of M_t holds because for each t, the total length of all paired segments in $[t-1, t]$ is at most 1). Consequently we have $P < (\gamma - \theta) \sum_{t=1}^{T} M_t$ as desired.

Now we show that U, the sum over unpaired segments, is at most $\frac{2}{\gamma\sqrt{1-\gamma}}$. If $\hat{N}$ is decreasing on each unpaired segment then clearly $U < 0$, so we suppose that $\hat{N}$ is increasing on each unpaired segment. Let $[e_{c_1-1}, e_{c_1}], \ldots, [e_{c_d-1}, e_{c_d}]$ be all the unpaired segments. As in Figure 9.2 it must be the case that the intervals $[\hat{N}(e_{c_i-1}), \hat{N}(e_{c_i}))$ are all disjoint and their union is $[0, N_T)$. By the definition of M, we have

$$U = \sum_{i=1}^{d} (1-\gamma)^{(N_{\lceil e_{c_i} \rceil - 1})/2} \left(\hat{N}(e_{c_i}) - \hat{N}(e_{c_i-1}) \right).$$

As in the bound for P, we have

$$N_{\lceil e_{c_i} \rceil - 1} \leq \hat{N}(e_{c_i-1}) < \hat{N}(e_{c_i}) \leq N_{\lceil e_{c_i} \rceil} \leq N_{\lceil e_{c_i} \rceil - 1} + 1 - \theta < N_{\lceil e_{c_i} \rceil - 1} + 1$$

and hence

$$\begin{aligned} U &\leq \sum_{i=1}^{d} (1-\gamma)^{(\hat{N}(e_{c_i})-1)/2} \left(\hat{N}(e_{c_i}) - \hat{N}(e_{c_i-1}) \right) \\ &= (1-\gamma)^{-1/2} \sum_{i=1}^{d} (1-\gamma)^{\hat{N}(e_{c_i})/2} \left(\hat{N}(e_{c_i}) - \hat{N}(e_{c_i-1}) \right). \end{aligned}$$

Since $\hat{N}$ is increasing, for each i we have

$$(1-\gamma)^{\hat{N}(e_{c_i})/2}\left(\hat{N}(e_{c_i})-\hat{N}(e_{c_i-1})\right) < \int_{z=\hat{N}(e_{c_i-1})}^{\hat{N}(e_{c_i})}(1-\gamma)^{z/2}dz.$$

Since the disjoint intervals $[\hat{N}(e_{c_i-1}),\hat{N}(e_{c_i}))$ cover $[0,N_T)$ we thus have

$$\begin{aligned}
U &< (1-\gamma)^{-1/2}\int_{z=0}^{N_T}(1-\gamma)^{z/2}dz \\
&< (1-\gamma)^{-1/2}\int_{z=0}^{\infty}(1-\gamma)^{z/2}dz \\
&= \frac{-2}{\sqrt{1-\gamma}\ln(1-\gamma)} \quad < \quad \frac{2}{\gamma\sqrt{1-\gamma}} \qquad \text{for } 0<\gamma<1/2.
\end{aligned}$$

(Lemma 9.7) ■

9.2.4 Comparison with Other Boosting Algorithms

The `Smoothboost` algorithm was inspired by an algorithm given by Impagliazzo in the context of hard-core set constructions in complexity theory (Impagliazzo, 1995). We observed in Chapter 8 that Impagliazzo's algorithm can be reinterpreted as a boosting-by-sampling algorithm which generates distributions $\mathcal{D}_t$ which, like the distributions generated by `SmoothBoost`, satisfy $L_\infty(\mathcal{D}_t) \le \frac{1}{\kappa m}$. However, our `Smoothboost` algorithm differs from Impagliazzo's algorithm in several important ways. The algorithm in (Impagliazzo, 1995) uses additive rather than multiplicative updates for $M_t(j)$, and the bound on T which is given for the algorithm in (Impagliazzo, 1995) is $O(\frac{1}{\kappa^2\gamma^2})$ which is worse than our bound by a factor of $\frac{1}{\kappa}$. Another important difference is that the algorithm in (Impagliazzo, 1995) has no margin parameter θ and does not appear to output a large margin final hypothesis. Finally, the analysis in (Impagliazzo, 1995) only covers the case where the weak hypotheses are binary-valued rather than real-valued.

Freund and Schapire's well-known boosting algorithm `AdaBoost` is somewhat faster than `SmoothBoost`, requiring only $T = O(\frac{\log(1/\kappa)}{\gamma^2})$ stages (Freund and Schapire, 1997). Like `SmoothBoost`, `AdaBoost` can be used with real-valued weak hypotheses and can be used to output a large margin final hypothesis (Schapire *et al.*, 1998). However, `AdaBoost` is *not* guaranteed to generate only smooth distributions, and thus does not appear to be useful in a malicious noise context.

The results of Chapter 8 imply that Freund's $\texttt{B}_{\text{Comb}}$ algorithm will generate distributions which satisfy $L_\infty(\mathcal{D}_t) \leq O(\frac{\log(1/\kappa)}{\kappa} \cdot \frac{1}{m})$, which is a weaker smoothness guarantee than `SmoothBoost` by a logarithmic factor. Another drawback of $\texttt{B}_{\text{Comb}}$ for the linear threshold learning application which we will study is the structure of the final hypothesis which it generates. As we saw in Chapter 8, if the weak learning algorithm returns hypotheses $h_1, \ldots, h_T$ then the final hypothesis generated by $\texttt{B}_{\text{Comb}}$ will be a depth two majority circuit with the hypotheses h_i at the inputs. In contrast, the `SmoothBoost` final hypothesis is simply a thresholded weighted sum of the hypotheses h_i, i.e. it is a depth one majority circuit. This simple structure of the final `SmoothBoost` hypothesis will play a crucial role in the generalization error bound of the next section.

Freund has recently studied a sophisticated boosting algorithm called `BrownBoost` (Freund, 1999) which uses a gentler weighting scheme than `AdaBoost`. Freund suggests that `BrownBoost` should be well suited for dealing with noisy data; however it is not clear from the analysis in (Freund, 1999) whether `BrownBoost`-generated distributions satisfy a smoothness property such as the $L_\infty(\mathcal{D}_t) \leq \frac{1}{\kappa m}$ property of `SmoothBoost`, or whether `BrownBoost` can be used to generate a large margin final hypothesis. We note that the `BrownBoost` algorithm is much more complicated to run than `SmoothBoost`, as it involves solving a differential equation at each stage of boosting.

`Smoothboost` is perhaps most similar to the modified AdaBoost algorithm called `MadaBoost` which was recently given by Domingo and Watanabe (Domingo and Watanabe, 2000). Like `SmoothBoost`, `MadaBoost` uses multiplicative updates on weights and never allows weights to exceed 1 in value. Domingo and Watanabe proved that `MadaBoost` takes at most $T \leq \frac{2}{\kappa\gamma^2}$ stages, which is quite similar to our bound in Theorem 9.5. (If we set $\theta = 0$ in `SmoothBoost`, a slight modification of the proof of Theorem 9.5 gives a bound of roughly $\frac{4}{3\kappa\gamma^2}$, which improves the `Madaboost` bound by a constant factor.) However, the analysis for `MadaBoost` given in (Domingo and Watanabe, 2000) only covers the case of binary-valued weak hypotheses, and does not establish that `MadaBoost` generates a large margin final hypothesis. We also note that our proof technique of simultaneously upper and lower bounding $\sum_{j=1}^m \sum_{t=1}^T M_t(j) y_j h_t(x^j)$ is different from the approach used in (Domingo and Watanabe, 2000).

9.3 Learning Linear Threshold Functions with Malicious Noise

In this section we show how the `SmoothBoost` algorithm can be used in conjunction with a simple noise tolerant weak learning algorithm to obtain a PAC learning algorithm for learning linear threshold functions with malicious noise.

9.3.1 Geometric Preliminaries

We briefly review the geometric preliminaries from Section 7.2. For $x \in \Re^n$ and $p \geq 1$, $\|x\|_p$ denotes the p-norm of x, namely $\|x\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$. The ∞-norm of x is $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$. We write $B_p(R)$ to denote the p-norm ball of radius R, i.e. $B_p(R) = \{x \in \Re^n : \|x\|_p \leq R\}$.

For $p, q \geq 1$ the q-norm is *dual* to the p-norm if $\frac{1}{p} + \frac{1}{q} = 1$; for us p and q will always denote dual norms. The following facts are useful:

Hölder Inequality: $|u \cdot v| \leq \|u\|_p \|v\|_q$ for all $u, v \in \Re^n$ and $1 \leq p \leq \infty$.

Minkowski Inequality: $\|u + v\|_p \leq \|u\|_p + \|v\|_p$ for all $u, v \in \Re^n$ and $1 \leq p \leq \infty$.

Recall that a *linear threshold function* is a function $f : \Re^n \to \{-1, 1\}$ such that $f(x) = \text{sign}(u \cdot x)$ for some $u \in \Re^n$.

9.3.2 PAC Learning with Malicious Noise

Let $EX^\eta_{MAL}(u, \mathcal{D})$ be a *malicious example oracle with noise rate* η that behaves as follows when invoked: with probability $1 - \eta$ the oracle returns a *clean* example $\langle x, \text{sign}(u \cdot x)\rangle$ where x is drawn from the probability distribution $\mathcal{D}$ over $B_p(R)$. With probability η, though, $EX^\eta_{MAL}(u, \mathcal{D})$ returns a *dirty* example $\langle x, y\rangle \in B_p(R) \times \{-1, 1\}$ about which nothing can be assumed. Such a malicious example $\langle x, y\rangle$ may be chosen by a computationally unbounded adversary which has complete knowledge of u, $\mathcal{D}$, and the state of the learning algorithm when the oracle is invoked.

The goal of a learning algorithm in this model is to construct an approximation to the target concept $\text{sign}(u \cdot x)$. More formally, we say that a Boolean function $h : \Re^n \to \{-1, 1\}$ is an *ϵ-approximator for u under $\mathcal{D}$* if $\Pr_{x \in \mathcal{D}}[h(x) \neq \text{sign}(u \cdot x)] \leq \epsilon$. The learning algorithm is given an accuracy parameter ϵ and a confidence parameter δ, has access to $EX^{\eta}_{MAL}(u, \mathcal{D})$, and must output a hypothesis h which, with probability at least $1-\delta$, is an ϵ-approximator for u under $\mathcal{D}$. The *sample complexity* of a learning algorithm in this model is the number of times it queries the malicious example oracle.

(A slightly stronger model of PAC learning with malicious noise has also been proposed (Aslam and Decatur, 1998; Cesa-Bianchi *et al.*, 1999). In this model first a clean sample of the desired size is drawn from a noise-free oracle; then each point in the sample is independently selected with probability η; then an adversary replaces each selected point with a dirty example of its choice; and finally the corrupted sample is provided to the learning algorithm. This model is stronger than the original malicious noise model since each dirty example is chosen by the adversary with full knowledge of the entire sample rather than knowledge only of the earlier examples that the learner has drawn thus far in the sample sequence. All of our results also hold in this stronger model.)

A final note: like the Perceptron algorithm, the learning algorithms which we consider will require that the quantity $u \cdot x$ be bounded away from zero (at least most of the time). We thus say that a distribution $\mathcal{D}$ is *ξ-good for u* if $|u \cdot x| \geq \xi$ for all x which have nonzero weight under $\mathcal{D}$, and we restrict our attention to learning under ξ-good distributions. Of course, dirty examples drawn from $EX^{\eta}_{MAL}(u, \mathcal{D})$ need not satisfy $|u \cdot x| \geq \xi$.

9.3.3 A Noise Tolerant Weak Learning Algorithm

For ease of reference we repeat the p-norm weak learning algorithm `WLA` from Section 7.3 in Figure 9.3. Recall that `WLA` takes as input a data set S and a distribution $\mathcal{D}$ over S. The algorithm computes the vector z which is the average location of the (label-normalized) points in S under $\mathcal{D}$, transforms z to obtain a vector w, and predicts using the linear functional defined by w.

We showed in Chapter 7 that the `WLA` algorithm is a weak learning algorithm for linear threshold functions in a noise-free setting. The following theorem shows that if a small fraction of the examples in S are affected by malicious noise, `WLA` will still generates a hypothesis with nonnegligible advantage provided that the input distribution $\mathcal{D}$ is sufficiently smooth.

Theorem 9.8 *Fix $2 \leq p \leq \infty$ and let $S = \langle x^1, y_1\rangle, \ldots, \langle x^m, y_m\rangle$ be a set of labeled*

Input:	real number $p \geq 2$ sample $S = \langle x^1, y_1 \rangle, \ldots, \langle x^m, y_m \rangle$ of labeled examples with $x^i \in \Re^n$, $y_i \in \{-1, 1\}$ distribution $\mathcal{D}$ over S
Output:	real valued hypothesis $h(x)$

1. **set** $z = \sum_{j=1}^m \mathcal{D}(j) y_j x^j$
2. **for all** $i = 1, \ldots, n$ **set** $w_i = \mathrm{sign}(z_i)|z_i|^{p-1}$
3. **return** hypothesis $h(x) \equiv v \cdot x$ where $v = \frac{w}{\|w\|_q R}$

Figure 9.3: The p-norm weak learning algorithm `WLA`.

examples with each $x^j \in B_p(R)$. Let $\mathcal{D}$ be a distribution over S such that $L_\infty(\mathcal{D}) \leq \frac{1}{\kappa m}$. Suppose that $\xi > 0$ and $u \in \Re^n$ are such that $\xi \leq R\|u\|_q$ and at most $\eta' m$ examples in S do not satisfy $y_j(u \cdot x^j) \geq \xi$, where $\eta' \leq \frac{\kappa \xi}{4R\|u\|_q}$. Then `WLA`$(p, S, \mathcal{D})$ returns a hypothesis $h : B_p(R) \to [-1, 1]$ which has advantage at least $\frac{\xi}{4R\|u\|_q}$ under $\mathcal{D}$.

Proof: By Hölder's inequality, for any $x \in B_p(R)$ we have

$$|h(x)| = \frac{|w \cdot x|}{\|w\|_q R} \leq \frac{\|w\|_q \|x\|_p}{\|w\|_q R} \leq 1,$$

and thus h indeed maps $B_p(R)$ into $[-1, 1]$.

Now we show that h has the desired advantage. Since $h_t(x^j) \in [-1, 1]$ and $y_j \in \{-1, 1\}$, we have $|h(x^j) - y_j| = 1 - y_j h(x^j)$, so

$$\frac{1}{2} \sum_{j=1}^m \mathcal{D}(j)|h(x^j) - y_j| = \frac{1}{2} \sum_{j=1}^m \mathcal{D}(j)(1 - y_j h(x^j)) = \frac{1}{2} - \left(\frac{\sum_{j=1}^m \mathcal{D}(j) y_j (w \cdot x^j)}{2\|w\|_q R} \right).$$

To prove the theorem it thus suffices to show that $\frac{\sum_{j=1}^m \mathcal{D}(j) y_j (w \cdot x^j)}{\|w\|_q} \geq \frac{\xi}{2\|u\|_q}$. The numerator of the left side is $w \cdot \left(\sum_{j=1}^m \mathcal{D}(j) y_j x^j \right) = w \cdot z = \sum_{i=1}^n |z_i|^p = \|z\|_p^p$. Using the fact that $(p-1)q = p$, the denominator is

$$\|w\|_q = \left(\sum_{i=1}^n \left(|z_i|^{p-1} \right)^q \right)^{1/q} = \left(\sum_{i=1}^n |z_i|^p \right)^{1/q} = \|z\|_p^{p/q}.$$

We can therefore rewrite the left side as $\|z\|_p^p / \|z\|_p^{p/q} = \|z\|_p$, and thus our goal is to show that $\|z\|_p \geq \frac{\xi}{2\|u\|_q}$. By Hölder's inequality it suffices to show that $z \cdot u \geq \frac{\xi}{2}$, which we now prove.

Let $S_1 = \{\langle x^j, y_j\rangle \in S : y_j(u \cdot x^j) \geq \xi\}$ and let $S_2 = S \setminus S_1$. The definition of S_1 immediately yields $\sum_{j\in S_1} \mathcal{D}(j)y_j(u\cdot x^j) \geq \mathcal{D}(S_1)\xi$. Moreover, since each $\|x^j\|_p \leq R$, by Hölder's inequality we have $y_j(u\cdot x^j) \geq -\|x^j\|_p\cdot\|u\|_q \geq -R\|u\|_q$ for each $\langle x^j, y_j\rangle \in S_2$. Since each example in S_2 has weight at most $\frac{1}{\kappa m}$ under $\mathcal{D}$, we have $\mathcal{D}(S_2) \leq \frac{\eta'}{\kappa}$, and hence

$$\begin{aligned} z\cdot u = \sum_{j=1}^{m} \mathcal{D}(j)y_j(u\cdot x^j) &= \sum_{j\in S_1} \mathcal{D}(j)y_j(u\cdot x^j) + \sum_{j\in S_2} \mathcal{D}(j)y_j(u\cdot x^j) \\ &\geq \mathcal{D}(S_1)\xi - \mathcal{D}(S_2)R\|u\|_q \\ &\geq \left(1 - \frac{\eta'}{\kappa}\right)\xi - \frac{\eta' R\|u\|_q}{\kappa} \\ &\geq \frac{3\xi}{4} - \frac{\xi}{4} = \frac{\xi}{2}, \end{aligned}$$

where the inequality $(1 - \frac{\eta'}{\kappa}) \geq \frac{3}{4}$ follows from the bound on η' and the fact that $\xi \leq R\|u\|_q$. ■

9.3.4 Putting it All Together

In this section we show how `SmoothBoost` can be used with `WLA` to obtain an algorithm for learning linear threshold functions in the presence of malicious noise.

The algorithm for learning $\text{sign}(u\cdot x)$ with respect to a ξ-good distribution $\mathcal{D}$ over $B_p(R)$ is as follows:

- Draw a sample $S = \langle x^1, y_1\rangle, \ldots, \langle x^m, y_m\rangle$ of m labeled examples from $EX^{\eta}_{MAL}(u, \mathcal{D})$.
- Run `Smoothboost` on S with parameters $\kappa = \frac{\epsilon}{4}$, $\gamma = \frac{\xi}{4R\|u\|_q}$, $\theta = \frac{\gamma}{2+\gamma}$ using `WLA` as the weak learning algorithm.

We now determine constraints on the sample size m and the malicious noise rate η under which this is a successful and efficient learning algorithm.

We first note that since $\mathcal{D}$ is ξ-good for u, we have that $\xi \leq R\|u\|_q$. Furthermore, since $\kappa = \frac{\epsilon}{4}$, Claim 9.3 implies that each distribution $\mathcal{D}_t$ which is given to `WLA` by `SmoothBoost` has $L_\infty(\mathcal{D}_t) \leq \frac{4}{\epsilon m}$. Let $S_C \subseteq S$ be the clean examples and $S_D = S \setminus S_C$ the dirty examples in S. If $\eta \leq \frac{\epsilon\xi}{32R\|u\|_q}$ and $m \geq \frac{96R\|u\|_q}{\epsilon\xi}\log\frac{2}{\delta}$, then a simple Chernoff bound implies that with probability at least $1 - \frac{\delta}{2}$ we have $|S_D| \leq \frac{\epsilon\xi}{16R\|u\|_q}m$. Thus, we can apply Theorem 9.8 with $\eta' = \frac{\epsilon\xi}{16R\|u\|_q}$; so each weak hypothesis $h_t(x) = v^t \cdot x$ generated by `WLA` has advantage $\frac{\xi}{4R\|u\|_q}$ under $\mathcal{D}_t$. Consequently, by Theorems 9.4 and 9.5, `Smoothboost` efficiently outputs a final hypothesis $h(x) = \text{sign} f(x)$ which has margin less than θ on at most an $\frac{\epsilon}{4}$ fraction of S. Since $|S_C|$ is easily seen to be at least $\frac{m}{2}$, we have that the margin of h is less than θ on at most an $\frac{\epsilon}{2}$ fraction of S_C. As in Chapter 7, this means that we can apply methods from the theory of data-dependent structural risk minimization (Bartlett and Shawe-Taylor, 1999; Shawe-Taylor *et al.*, 1998) to bound the error of h under $\mathcal{D}$.

Recall that the final `Smoothboost` hypothesis is $h(x) = \text{sign}(f(x))$ where $f(x) = v \cdot x$ is a convex combination of hypotheses $h_t(x) = v^t \cdot x$. Since each vector v^t satisfies $\|v^t\|_q \leq \frac{1}{R}$, by Minkowski's inequality we have that $\|v\|_q \leq \frac{1}{R}$ as well. We proved the following theorem in Chapter 7 (Theorem 7.8):

Theorem 9.9 *Fix any value $2 \leq p \leq \infty$ and let $\mathcal{F}$ be the class of functions $\{x \mapsto v \cdot x : \|v\|_q \leq \frac{1}{R}, x \in B_p(R)\}$. Then $fat_{\mathcal{F}}(\mu) \leq \frac{2 \log 4n}{\mu^2}$, where $fat_{\mathcal{F}}(\mu)$ is the* fat-shattering dimension of $\mathcal{F}$ at scale μ *as defined in, e.g., (Bartlett* et al.*, 1996; Bartlett and Shawe-Taylor, 1999; Shawe-Taylor* et al.*, 1998).*

The following theorem is from (Bartlett and Shawe-Taylor, 1999):

Theorem 9.10 (Bartlett and Shawe-Taylor, 1999) *Let $\mathcal{F}$ be a collection of real-valued functions over some domain X, let $\mathcal{D}$ be a distribution over $X \times \{-1, 1\}$, let $S = \langle x^1, y_1\rangle, \ldots, \langle x^m, y_m\rangle$ be a sequence of labeled examples drawn from $\mathcal{D}$, and let $h(x) = sign(f(x))$ for some $f \in \mathcal{F}$. If h has margin less than θ on at most k examples in S, then with probability at least $1 - \delta$ we have that $\Pr_{\langle x,y\rangle \in \mathcal{D}}[h(x) \neq y]$ is at most*

$$\frac{k}{m} + \sqrt{\frac{2}{m}(d \ln(34e/m) \log(578m) + \ln(4/\delta))}, \tag{9.10}$$

where $d = fat_{\mathcal{F}}(\theta/16)$.

We have that h has margin less than θ on at most an $\frac{\epsilon}{2}$ fraction of the clean examples S_C, so we may take k/m to be $\frac{\epsilon}{2}$ in the above theorem. Now if we apply Theorem 9.9 and solve for m the inequality obtained by setting (1) to be at most ϵ, we obtain our main result:

Theorem 9.11 *Fix $2 \le p \le \infty$ and let $\mathcal{D}$ be a distribution over $B_p(R)$ which is ξ-good for u. The algorithm described above uses $m = \tilde{O}\left(\left(\frac{R\|u\|_q}{\xi\epsilon}\right)^2\right)$ examples and outputs an ϵ-approximator for u under $\mathcal{D}$ with probability $1-\delta$ in the presence of malicious noise at a rate $\eta = \Omega\left(\epsilon \cdot \frac{\xi}{R\|u\|_q}\right)$.*

9.4 Discussion

9.4.1 Comparison with Online Algorithms

The bounds on sample complexity and malicious noise tolerance of our `SmoothBoost`-based algorithms given by Theorem 9.11 are remarkably similar to the bounds which can be obtained through a natural PAC conversion of the online p-norm algorithms introduced by Grove, Littlestone and Schuurmans and studied by Gentile and Littlestone. As stated in Theorem 7.2 of Chapter 7, Grove, Littlestone and Schuurmans proved that the online p-norm algorithm makes at most $O\left(\left(\frac{R\|u\|_q}{\xi}\right)^2\right)$ mistakes on linearly separable data. Subsequently Gentile and Littlestone extended the analysis from (Grove *et al.*, 1997) and considered a setting in which the examples are not linearly separable. Their analysis (Theorem 6) implies that if an example sequence containing K malicious errors is provided to the online p-norm algorithm, then the algorithm will make at most

$$O\left(\left(\frac{R\|u\|_q}{\xi}\right)^2 + K \cdot \frac{R\|u\|_q}{\xi}\right)$$

mistakes. To obtain PAC-model bounds on the online p-norm algorithms in the presence of malicious noise, we use the following theorem due to Auer and Cesa-Bianchi (Auer and Cesa-Bianchi, 1998) (Theorem 6.2):

Theorem 9.12 *Fix a hypothesis class $\mathcal{H}$ of Vapnik-Chervonenkis dimension d. Let A be an online learning algorithm with the following properties: (1) A only uses hypotheses which belong to $\mathcal{H}$, (2) if A is given a noise-free example sequence then A makes at most m_0 mistakes, and (3) if A is given an example sequence with K malicious errors then A makes at most $m_0 + BK$ mistakes. Then there is a PAC algorithm A' which learns to accuracy ϵ and confidence δ, uses $\tilde{O}(\frac{B^2}{\epsilon^2} + \frac{m_0}{\epsilon} + \frac{d}{\epsilon})$ examples, and can tolerate a malicious noise rate $\eta = \frac{\epsilon}{2B}$.*

Applying this theorem, we find that these PAC conversions of the online p-norm algorithms have sample complexity and malicious noise tolerance bounds which are essentially identical to the bounds given for our `SmoothBoost`-based algorithm. We note that these algorithms have the best bounds on malicious noise tolerance of any known computationally efficient algorithms for learning linear threshold functions.

9.4.2 `SmoothBoost` is Optimally Smooth

We observed in Section 8.4.6 that any hard-core set construction theorem must have a set size parameter which is at most $O(\epsilon)$. Using the connection between hard-core set constructions and boosting algorithms described in Chapter 8, it follows that no algorithm which boosts to accuracy ϵ over a universe of m points can satisfy a stronger "smoothness guarantee" on its distributions than the bound of $L_\infty(\mathcal{D}_t) = O(\frac{1}{\epsilon m})$ which is achieved by Impagliazzo's algorithm. Since `SmoothBoost` also satisfies this bound, it too generates distributions which are optimally smooth up to a constant factor. We now give an alternative proof of `SmoothBoost`'s optimality which highlights an interesting connection between the limits of boosting algorithms and known limits on noise-tolerant learning.

It is evident from the proof of Theorem 9.11 that the smoothness of the distributions generated by `SmoothBoost` relates directly to the level of malicious noise which our linear threshold learning algorithm can tolerate. On the other hand, as mentioned in Section 9.1, Kearns and Li have shown that for a broad range of concept classes no algorithm can learn to accuracy ϵ in the presence of malicious noise at a rate $\eta > \frac{\epsilon}{1+\epsilon}$. Using the Kearns-Li upper bound on malicious noise tolerance, we can prove that `SmoothBoost` is optimal up to constant factors in terms of the smoothness of the distributions which it generates.

Recall that if `SmoothBoost` is run on a set of m examples with accuracy parameter κ, then each distribution $\mathcal{D}_t$ which `SmoothBoost` constructs will satisfy $L_\infty(\mathcal{D}_t) \leq \frac{1}{\kappa m}$. The proof is by contradiction; so suppose that there exists a boosting algorithm called `SuperSmoothBoost` which is similar to `SmoothBoost` but which has an even stronger smoothness guarantee on its distributions. More precisely we suppose that `SuperSmoothBoost` takes as input parameters κ, γ and a labeled sample S of size m, has access to a weak learning algorithm `WL`, generates a sequence $\mathcal{D}_1, \mathcal{D}_2, \ldots$ of distributions over S, and outputs a Boolean-valued final hypothesis h. As in Section

9.2.3, we suppose that if the weak learning algorithm `WL` always returns a hypothesis h_t which has advantage γ under $\mathcal{D}_t$, then `SuperSmoothboost` will eventually halt and the final hypothesis h will agree with at least a $1-\kappa$ fraction of the labeled examples in S. Finally, we suppose that each distribution $\mathcal{D}_t$ is guaranteed to satisfy $L_\infty(\mathcal{D}_t) \leq \frac{1}{64\kappa m}$.

Consider the following severely restricted linear threshold learning problem: the domain is $\{-1,1\}^2 \subset \Re^2$, so any distribution $\mathcal{D}$ can assign weight only to these four points. Moreover, we only allow two possibilities for the target concept $\text{sign}(u \cdot x)$: the vector u is either $(1,0)$ or $(0,1)$. The four points in $\{-1,1\}^2$ are classified in all four possible ways by these two concepts, and hence the concept class consisting of these two concepts is a *distinct* concept class as defined by Kearns and Li (Kearns and Li, 1993). It is clear that every example belongs to $B_\infty(1)$ (i.e. $R=1$), that $\|u\|_1 = 1$, and that any distribution $\mathcal{D}$ over $\{-1,1\}^2$ is 1-good for u (i.e. $\xi = 1$).

Consider the following algorithm for this restricted learning problem:

- Draw a sample $S = \langle x^1, y_1\rangle, \ldots, \langle x^m, y_m\rangle$ of m labeled examples from $EX^\eta_{MAL}(u, \mathcal{D})$;
- Run `SuperSmoothBoost` on S with parameters $\kappa = \frac{\epsilon}{4}$, $\gamma = \frac{\xi}{4R\|u\|_q} = \frac{1}{4}$ using `WLA` with $p = \infty$ as the weak learning algorithm.

Suppose that the malicious noise rate η is 2ϵ. As in Section 9.3.4, a Chernoff bound shows that for $m = O(\frac{1}{\epsilon}\log\frac{1}{\delta})$, with probability at least $1-\frac{\delta}{2}$ we have that the sample S contains at most $4\epsilon m$ dirty examples. By the `SuperSmoothBoost` smoothness property and our choice of κ, we have that $L_\infty(\mathcal{D}_t) \leq \frac{1}{16\epsilon m}$. Theorem 9.8 now implies that each `WLA` hypothesis h_t has advantage at least $\frac{\xi}{4R\|u\|_q} = \frac{1}{4}$ with respect to $\mathcal{D}_t$. As in Section 9.3.4, we have that with probability at least $1-\frac{\delta}{2}$ the final hypothesis h output by `SuperSmoothBoost` disagrees with at most an $\frac{\epsilon}{2}$ fraction of the clean examples S_C.

Since the domain is finite (in fact of size four) we can bound generalization error directly. A simple Chernoff bound argument shows that if m is sufficiently large, then with probability at least $1-\delta$ the hypothesis h will be an ϵ-approximator for $\text{sign}(u \cdot x)$ under $\mathcal{D}$. However, Kearns and Li have shown (Theorem 1 of (Kearns and Li, 1993)) that no learning algorithm for a distinct concept class can learn to accuracy ϵ with probability $1-\delta$ in the presence of malicious noise at rate $\eta \geq \frac{\epsilon}{1+\epsilon}$. This contradiction proves that the `SuperSmoothBoost` algorithm cannot exist, and hence the distributions generated by `SmoothBoost` are optimal up to constant factors.

Chapter 10

Learning Monotone DNF under Product Distributions

In this final chapter we again consider the DNF learning problem, but this time from a new perspective. While Chapter 3 gave a superpolynomial time algorithm for learning an arbitrary DNF under an arbitrary distribution, we now focus on restricted versions of the DNF learning problem which can be solved in polynomial time.

The main result of this chapter is an algorithm that PAC learns the class of monotone $2^{O(\sqrt{\log n})}$-term DNF in polynomial time under the uniform distribution, using random examples only. This is an exponential improvement over the best previous polynomial-time algorithms in this well-studied model, which can learn monotone $o(\log^2 n)$-term DNF, and is the first efficient algorithm for monotone $(\log n)^{\omega(1)}$-term DNF in any model of learning from random examples. (Recall from Chapter 8 that Jackson's Harmonic Sieve algorithm for learning DNF under the uniform distribution makes essential use of membership queries and thus does not fit into the paradigm of learning from random examples only.) We also show how to extend our algorithm so that it learns in polynomial time under any constant-bounded product distribution, and show that the algorithm can be used to learn various classes of constant-depth monotone circuits with few relevant variables.

10.1 Introduction

10.1.1 Learning DNF in Restricted Models

In his seminal 1984 paper (Valiant, 1984) Valiant introduced the distribution-free model of Probably Approximately Correct (PAC) learning from random examples and posed the following question: can polynomial size DNF be learned in polynomial time using random examples drawn from an arbitrary distribution? Over the subsequent years, the apparent difficulty of designing polynomial time algorithms for this general problem led many researchers to study restricted versions of the DNF learning problem which admit polynomial time algorithms. As described below, the restrictions which have been considered include

- allowing the learner to make *membership queries* for the value of the target function at points selected by the learner;
- requiring that the learner succeed only under restricted distributions on examples, such as the uniform distribution, rather than all distributions;
- requiring that the learner succeed only for restricted subclasses of DNF formulae such as monotone DNF with a bounded number of terms.

A SAT-k DNF is a DNF in which each truth assignment satisfies at most k terms. Khardon (Khardon, 1994) gave a polynomial time membership query algorithm for learning polynomial-size SAT-1 DNF under the uniform distribution; this result was later strengthened by Blum et al. (Blum *et al.*, 1994) to SAT-k DNF for any constant k. Bellare gave a polynomial time membership query algorithm for learning $O(\log n)$-term DNF under the uniform distribution (Bellare, 1992). This result was strengthened by Blum and Rudich gave a polynomial time membership query algorithm (Blum and Rudich, 1995) who gave a polynomial time algorithm for exact learning $O(\log n)$-term DNF using membership and equivalence queries; several other polynomial-time algorithms for $O(\log n)$-term DNF have since been given in this model (Beimel *et al.*, 1996; Bshouty, 1995; Bshouty, 1997; Kushilevitz, 1997). Using techniques from Fourier analysis Mansour gave an $n^{O(\log \log n)}$-time membership query algorithm which learns polynomial-size DNF under the uniform distribution (Mansour, 1995). As described in Chapter 8, in a celebrated result Jackson gave the Harmonic Sieve algorithm which runs in polynomial time and uses membership queries to learn polynomial size DNF in polynomial time under the uniform distribution (Jackson, 1997). In fact, Jackson proves that the Sieve can be modified to run in polynomial time under any constant-bounded product distribution.

In the standard PAC model without membership queries positive results are known for various subclasses of DNF under restricted distributions. A *read-k* DNF is one in which each variable appears at most k times. Kearns *et al.* showed that read-once DNF are PAC learnable under the uniform distribution in polynomial time (Kearns *et al.*, 1994). Hancock extended this result to read-k DNF for any constant k (Hancock, 1992). Verbeurgt gave an algorithm for learning arbitrary polynomial-size DNF under the uniform distribution in time $n^{O(\log n)}$ (Verbeurgt, 1990). In an important result Linial *et al.* gave an algorithm for learning any AC^0 circuit (constant depth, polynomial size, unbounded fanin AND/OR gates) under the uniform distribution in $n^{poly(\log n)}$ time (Linial *et al.*, 1993).

A *monotone* DNF is a DNF with no negated variables. It is well known that in the distribution-independent setting, learning monotone DNF is equivalent to learning general DNF (Kearns *et al.*, 1987b). However this equivalence does not hold for learning under restricted distributions such as the uniform distribution, and many researchers have studied the problem of learning monotone DNF under restricted distributions. Hancock and Mansour gave a polynomial time algorithm for learning monotone read-k DNF under constant-bounded product distributions (Hancock and Mansour, 1991). Verbeurgt gave a polynomial time uniform distribution algorithm for learning poly-disjoint one-read-once monotone DNF and read-once factorable monotone DNF (Verbeurgt, 1998). Kucera *et al.* gave a polynomial-time algorithm which learns monotone k-term DNF under the uniform distribution using hypotheses which are monotone k-term DNF (Kucera *et al.*, 1994). This was improved by Sakai and Maruoka who gave a polynomial-time algorithm for learning monotone $O(\log n)$-term DNF under the uniform distribution using hypotheses which are monotone $O(\log n)$-term DNF (Sakai and Maruoka, 2000). In (Bshouty, 1995) Bshouty gave a polynomial-time uniform-distribution algorithm for learning a class which includes monotone $O(\log n)$-term DNF. Later Bshouty and Tamon gave a polynomial-time algorithm for learning a class which includes monotone $O(\log^2 n/(\log\log n)^3)$-term DNF under constant-bounded product distributions (Bshouty and Tamon, 1996).

10.1.2 Learning Larger Monotone DNF

We give an algorithm for learning monotone DNF under the uniform distribution. If the desired accuracy level ϵ is constant as a function of n (the number of variables), then the algorithm learns $2^{O(\sqrt{\log n})}$-term monotone DNF over n variables in poly(n) time. (We note that the algorithm of (Bshouty and Tamon, 1996) for learning monotone DNF with $O((\log n)^2/(\log \log n)^3)$ terms also requires that ϵ be constant in order to achieve poly(n) runtime.) This is the first polynomial time algorithm which uses only random examples and successfully learns monotone DNF with more than a polylogarithmic number of terms. We also show that essentially the same algorithm learns various classes of small constant-depth circuits which compute monotone functions on few variables. These results extend to learning under any constant-bounded product distribution.

Our algorithm combines ideas from Linial *et al.*'s influential paper (Linial *et al.*, 1993) on learning AC^0 functions using the Fourier transform and Bshouty and Tamon's paper (Bshouty and Tamon, 1996) on learning monotone functions using the Fourier transform. By analyzing the Fourier transform of AC^0 functions, Linial *et al.* showed that almost all of the Fourier "power spectrum" of any AC^0 function is contained in "low" Fourier coefficients, i.e. coefficients which correspond to small subsets of variables. (We give a detailed overview of Fourier analysis on the Boolean cube in Section 10.2.2.) Their learning algorithm estimates each low Fourier coefficient by sampling and constructs an approximation to f using these estimated Fourier coefficients. If c is the size bound for low Fourier coefficients, then since there are $\binom{n}{c}$ Fourier coefficients corresponding to subsets of c variables the algorithm requires roughly n^c time steps. Linial *et al.* showed that for AC^0 circuits c is essentially poly($\log n$); this result was later sharpened for DNF formulae by Mansour (Mansour, 1995).

Our algorithm extends this approach in the following way: Let $C \subset AC^0$ be a class of Boolean functions which we would like to learn. Suppose that C has the following properties:

1. For every $f \in C$ there is a set S_f of "important" variables such that almost all of the power spectrum of f is contained in Fourier coefficients corresponding to subsets of S_f.

2. There is an efficient algorithm which identifies the set S_f from random examples.

(Such an algorithm, which we give in Section 10.3.1, is implicit in (Bshouty and Tamon, 1996) and requires only that f be monotone.) We can learn an unknown function f from such a class C by first identifying the set S_f, then estimating the low Fourier coefficients which correspond to small subsets of S_f and using these estimates to construct an approximation to f. To see why this works, note that since f is in AC^0 almost all of the power spectrum of f is in the low Fourier coefficients; moreover, property (1) implies that almost all of the power spectrum of f is in the Fourier coefficients which correspond to subsets of S_f. Consequently it must be the case that almost all of the power spectrum of f is in low Fourier coefficients which correspond to subsets of S_f. Thus in our setting we need only estimate the $\binom{|S_f|}{c}$ Fourier coefficients which correspond to "small" subsets of variables in S_f. If $|S_f| \ll n$ then this is much more efficient than estimating all $\binom{n}{c}$ low Fourier coefficients.

10.2 Preliminaries

We write $[n]$ to denote the set $\{1, \ldots, n\}$ and use capital letters A, B, C for subsets of $[n]$. We write $|A|$ to denote the number of elements in A. The lowercase letter x always denotes an n-bit string $x = x_1 \ldots x_n \in \{0,1\}^n$.

We view Boolean circuits as being composed of AND/OR/NOT gates where AND and OR gates have unbounded fanin and negations occur only on inputs. We view Boolean functions on n variables as real valued functions which map $\{0,1\}^n$ to $\{-1,1\}$. A Boolean function $f : \{0,1\}^n \to \{-1,1\}$ is *monotone* if changing the value of an input bit from 0 to 1 never causes the value of f to change from 1 to -1.

If $\mathcal{D}$ is a distribution and f is a Boolean function on $\{0,1\}^n$, then as in (Bshouty and Tamon, 1996; Hancock and Mansour, 1991; Kahn *et al.*, 1988) we say that the *influence of x_i on f with respect to $\mathcal{D}$* is the probability that $f(x)$ differs from $f(y)$, where y is x with the i-th bit flipped and x is drawn from $\mathcal{D}$. For ease of notation let $f_{i,0}$ denote the function obtained from f by fixing x_i to 0 and let $f_{i,1}$ be defined similarly. We thus have

$$I_{\mathcal{D},i}(f) = \Pr_{\mathcal{D}}[f_{i,0}(x) \neq f_{i,1}(x)] = \frac{1}{2} E_{\mathcal{D}}[|f_{i,1} - f_{i,0}|].$$

For monotone f this can be further simplified to

$$I_{\mathcal{D},i}(f) = \frac{1}{2}E_{\mathcal{D}}[f_{i,1} - f_{i,0}] = \frac{1}{2}\left(E_{\mathcal{D}}[f_{i,1}] - E_{\mathcal{D}}[f_{i,0}]\right). \tag{10.1}$$

10.2.1 Distribution-Specific PAC Learning

The learning model we consider in this chapter is a distribution-specific version of the PAC learning model which has been studied by many researchers, e.g. (Blum *et al.*, 1994; Bellare, 1992; Bshouty *et al.*, 1999; Bshouty and Tamon, 1996; Furst *et al.*, 1991; Hancock and Mansour, 1991; Jackson, 1997; Khardon, 1994; Kucera *et al.*, 1994; Linial *et al.*, 1993; Mansour, 1995; Verbeurgt, 1990; Verbeurgt, 1998). Let C be a class of Boolean concepts over $\{0,1\}^n$, let $\mathcal{D}$ be a probability distribution over $\{0,1\}^n$, and let $f \in C$ be an unknown target function. A learning algorithm A for C takes as input an accuracy parameter $0 < \epsilon < 1$ and a confidence parameter $0 < \delta < 1$. During its execution the algorithm has access to an *example oracle* $EX(f,\mathcal{D})$ which, when queried, generates a random labeled example $\langle x, f(x)\rangle$ where x is drawn according to $\mathcal{D}$. The learning algorithm outputs a hypothesis h which is a Boolean function over $\{0,1\}^n$; the error of this hypothesis is defined to be $\mathrm{error}(h) = \Pr_{\mathcal{D}}[h(x) \neq f(x)]$. We say that A *learns* C *under* $\mathcal{D}$ if for every $f \in C$ and $0 < \epsilon, \delta < 1$, with probability at least $1-\delta$ algorithm A outputs a hypothesis h which has $\mathrm{error}(h) \leq \epsilon$.

10.2.2 The Discrete Fourier Transform

Let $\mathcal{U}$ denote the uniform distribution over $\{0,1\}^n$. The set of all real valued functions on $\{0,1\}^n$ may be viewed as a 2^n-dimensional vector space with inner product defined by

$$\langle f, g\rangle = 2^{-n}\sum_{x\in\{0,1\}^n} f(x)g(x) = E_{\mathcal{U}}[fg]$$

and norm defined by $\|f\| = \sqrt{\langle f, f\rangle}$. Given any subset $A \subseteq [n]$, the Fourier basis function $\chi_A : \{0,1\}^n \to \{-1,1\}$ is defined by $\chi_A(x) = (-1)^{|A\cap X|}$, where X is the subset of $[n]$ defined by $i \in X$ iff $x_i = 1$. It is well known that the 2^n basis functions χ_A form an orthonormal basis for the vector space of real valued functions on $\{0,1\}^n$; we refer to this basis as *the* χ *basis.* In particular, any function f can be uniquely expressed as $f(x) = \sum_A \hat{f}(A)\chi_A(x)$, where the values $\hat{f}(A)$ are known as the Fourier coefficients of f with respect to the χ basis. Since the functions χ_A form an orthonormal basis, the value of $\hat{f}(A)$ is $\langle f, \chi_A \rangle$; also, by linearity we have that $f(x) + g(x) = \sum_A (\hat{f}(A) + \hat{g}(A))\chi_A(x)$. Another easy consequence of orthonormality is Parseval's identity

$$E_{\mathcal{U}}[f^2] = \|f\|^2 = \sum_{A \subseteq [n]} \hat{f}(A)^2.$$

If f is a Boolean function then this value is exactly 1. Finally, for any Boolean function f and real-valued function g we have the following easily verified inequality (Bshouty and Tamon, 1996; Linial *et al.*, 1993):

$$\Pr_{\mathcal{U}}[f \neq \text{sign}(g)] \leq E_{\mathcal{U}}[(f-g)^2] \tag{10.2}$$

where sign(g) takes value 1 if $g \geq 0$ and takes value -1 if $g < 0$.

10.3 Learning under Uniform Distributions

10.3.1 Identifying Relevant Variables

The following lemma, which is implicit in (Bshouty and Tamon, 1996), gives an efficient algorithm for identifying the important variables of a monotone Boolean function. We refer to this algorithm as `FindVariables`.

Lemma 10.2 *Let $f : \{0,1\}^n \rightarrow \{-1,1\}$ be a monotone Boolean function. There is an algorithm which has access to $EX(f, \mathcal{U})$, runs in poly$(n, 1/\epsilon, \log 1/\delta)$ time steps for all $\epsilon, \delta > 0$, and with probability at least $1-\delta$ outputs a set $S_f \subseteq [n]$ such that*

$$i \in S_f \text{ implies } \sum_{A: i \in A} \hat{f}(A)^2 \geq \epsilon/2 \qquad \text{and} \qquad i \notin S_f \text{ implies } \sum_{A: i \in A} \hat{f}(A)^2 \leq \epsilon.$$

Proof: Kahn *et al.* ((Kahn *et al.*, 1988) Section 3) have shown that

$$I_{\mathcal{U},i}(f) = \sum_{A: i \in A} \hat{f}(A)^2. \tag{10.3}$$

To prove the lemma it thus suffices to show that $I_{\mathcal{U},i}(f)$ can be estimated to within accuracy $\epsilon/4$ with high probability. By Equation (10.1) from Section 10.2 this can be done by estimating $E_{\mathcal{U}}[f_{i,1}]$ and $E_{\mathcal{U}}[f_{i,0}]$. Using Chernoff bounds it is easily verified that with probability at least $1-\delta/2n$ a sample of size $\text{poly}(n, 1/\epsilon, \log 1/\delta)$ drawn from $EX(f, \mathcal{U})$ will contain $\text{poly}(n, 1/\epsilon, \log 1/\delta)$ examples with $x_i = b$ for each value of $b = 0, 1$. A second application of Chernoff bounds shows that with probability at least $1-\delta/2n$ a sample of $\text{poly}(n, 1/\epsilon, \log 1/\delta)$ labeled examples drawn uniformly from $\{x : x_i = b\}$ yields an $\epsilon/4$-accurate estimate of $E_{\mathcal{U}}[f_{i,b}]$. ∎

10.3.2 The Learning Algorithm

Our learning algorithm, which we call `LearnMonotone`, is given below:

- Use `FindVariables` to identify a set S_f of important variables.
- Draw m labeled examples $\langle x^1, f(x^1)\rangle, \ldots, \langle x^m, f(x^m)\rangle$ from $EX(f, \mathcal{U})$. For every $A \subseteq S_f$ with $|A| \leq c$ set $\alpha_A = \frac{1}{m}\sum_{i=1}^m f(x^i)\chi_A(x^i)$. For every A such that $|A| > c$ or $A \not\subseteq S_f$ set $\alpha_A = 0$.
- Output the hypothesis $\text{sign}(g(x))$, where $g(x) = \sum_A \alpha_A \chi_A(x)$.

The algorithm thus estimates $\hat{f}(A)$ for $A \subseteq S_f, |A| \leq c$ by sampling and constructs a hypothesis using these approximate Fourier coefficients. The values of m and c and the parameter settings for `FindVariables` are specified below.

10.3.3 Learning Monotone $2^{O(\sqrt{\log n})}$-term DNF

Let $f : \{0,1\}^n \to \{-1,1\}$ be a monotone s-term DNF. The proof that algorithm `LearnMonotone` learns f uses one DNF which we call f_1 to show that `FindVariables` identifies a small set of variables S_f and uses another DNF which we call f_2 to show that f can be approximated by approximating Fourier coefficients which correspond to small subsets of S_f.

Let f_1 be the DNF which is obtained from f by removing every term which contains more than $\log \frac{32sn}{\epsilon}$ variables. Since there are at most s such terms each of which is satisfied by a random example with probability less than $\epsilon/32sn$, we have $\Pr_{\mathcal{U}}[f(x) \neq f_1(x)] < \frac{\epsilon}{32n}$ (this type of argument was first used in (Verbeurgt, 1990)). Let $R \subseteq [n]$ be the set of variables which f_1 depends on; it is clear that $|R| \leq s \log \frac{32sn}{\epsilon}$.

Moreover since $I_{\mathcal{U},i}(f_1) = 0$ for $i \notin R$, equation (10.3) from Section 10.3.1 implies that $\hat{f}_1(A) = 0$ for $A \not\subseteq R$.

Since f and f_1 are Boolean functions, $f - f_1$ is either 0 or 2, so $E_{\mathcal{U}}[(f - f_1)^2] = 4\,\Pr_{\mathcal{U}}[f \neq f_1] < \epsilon/8n$. By Parseval's identity we have

$$\begin{aligned}
E_{\mathcal{U}}[(f-f_1)^2] &= \sum_A (\hat{f}(A) - \hat{f}_1(A))^2 \\
&= \sum_{A \subseteq R} (\hat{f}(A) - \hat{f}_1(A))^2 + \sum_{A \not\subseteq R} (\hat{f}(A) - \hat{f}_1(A))^2 \\
&= \sum_{A \subseteq R} (\hat{f}(A) - \hat{f}_1(A))^2 + \sum_{A \not\subseteq R} (\hat{f}(A))^2 \\
&< \epsilon/8n.
\end{aligned}$$

Thus $\sum_{A \not\subseteq R} \hat{f}(A)^2 < \frac{\epsilon}{8n}$, and consequently we have

$$i \notin R \text{ implies } \sum_{A : i \in A} \hat{f}(A)^2 < \frac{\epsilon}{8n}. \tag{10.4}$$

We set the parameters of `FindVariables` so that with high probability

$$i \in S_f \quad \text{implies} \quad \sum_{A : i \in A} \hat{f}(A)^2 \geq \epsilon/8n \tag{10.5}$$

$$i \notin S_f \quad \text{implies} \quad \sum_{A : i \in A} \hat{f}(A)^2 \leq \epsilon/4n. \tag{10.6}$$

Inequalities (10.4) and (10.5) imply that $S_f \subseteq R$, so $|S_f| \leq s \log \frac{32sn}{\epsilon}$. Furthermore, since $A \not\subseteq S_f$ implies $i \in A$ for some $i \notin S_f$, inequality (10.6) implies

$$\sum_{A \not\subseteq S_f} \hat{f}(A)^2 \leq \epsilon/4. \tag{10.7}$$

The following lemma is due to Mansour:

Lemma 10.3 (Mansour, 1995) *Let f be a DNF with terms of size at most d. Then for all $\epsilon > 0$*

$$\sum_{|A| > 20d\log(2/\epsilon)} \hat{f}(A)^2 \leq \epsilon/2.$$

One approach at this point is to use Mansour's lemma to approximate f by approximating the Fourier coefficients of all subsets of S_f which are smaller than $20d\log(2/\epsilon)$, where $d = \log \frac{32sn}{\epsilon}$ is the maximum size of any term in f_1. However, this approach does not give a good overall running time because d is too large. Instead we consider another DNF with smaller terms than f_1 which also closely approximates f. By using this stronger bound on term size in Mansour's lemma we get a better final result.

More precisely, let f_2 be the DNF obtained from f by removing every term which contains at least $\log \frac{32s}{\epsilon}$ variables. Let $c = 20 \log \frac{128s}{\epsilon} \log \frac{8}{\epsilon}$. Mansour's lemma implies that

$$\sum_{|A|>c} \hat{f}_2(A)^2 \leq \epsilon/8. \tag{10.8}$$

Moreover, we have $\Pr_{\mathcal{U}}[f \neq f_2] \leq \epsilon/32$ and hence

$$4 \Pr_{\mathcal{U}}[f \neq f_2] = E_{\mathcal{U}}[(f - f_2)^2] = \sum_A (\hat{f}(A) - \hat{f}_2(A))^2 \leq \epsilon/8. \tag{10.9}$$

Let α_A and $g(x)$ be as defined in `LearnMonotone`. Using inequality (10.2) from Section 10.2.2, we have

$$\Pr[\text{sign}(g) \neq f] \leq E_{\mathcal{U}}[(g - f)^2] = \sum_A (\alpha_A - \hat{f}(A))^2 = X + Y + Z,$$

where

$$X = \sum_{|A| \leq c, A \not\subseteq S_f} (\alpha_A - \hat{f}(A))^2, \quad Y = \sum_{|A|>c} (\alpha_A - \hat{f}(A))^2, \quad Z = \sum_{|A| \leq c, A \subseteq S_f} (\alpha_A - \hat{f}(A))^2.$$

To bound X, we observe that $\alpha_A = 0$ for $A \not\subseteq S_f$, so by (10.7) we have

$$X = \sum_{|A| \leq c, A \not\subseteq S_f} \hat{f}(A)^2 \leq \sum_{A \not\subseteq S_f} \hat{f}(A)^2 \leq \epsilon/4.$$

To bound Y, we note that $\alpha_A = 0$ for $|A| > c$ and hence $Y = \sum_{|A|>c} \hat{f}(A)^2$. Since $\hat{f}(A)^2 \leq 2(\hat{f}(A) - \hat{f}_2(A))^2 + 2\hat{f}_2(A)^2$, we have

$$\begin{aligned} Y &\leq 2 \sum_{|A|>c} (\hat{f}(A) - \hat{f}_2(A))^2 + 2 \sum_{|A|>c} \hat{f}_2(A)^2 \\ &\leq 2 \sum_A (\hat{f}(A) - \hat{f}_2(A))^2 + \epsilon/4 \\ &\leq \epsilon/2 \end{aligned}$$

by inequalities (10.8) and (10.9) respectively.

It remains to bound $Z = \sum_{|A| \le c, A \subseteq S_f} (\alpha_A - \hat{f}(A))^2$. As in (Linial *et al.*, 1993) this sum can be made less than $\epsilon/4$ by taking m sufficiently large so that with high probability each estimate α_A differs from the true value $\hat{f}(A)$ by at most $\sqrt{\epsilon/4|S_f|^c}$. A straightforward use of Chernoff bounds shows that $m = \text{poly}(|S_f|^c, 1/\epsilon, \log(1/\delta))$ suffices to ensure that $Z \le \epsilon/4$.

Thus, we have $X+Y+Z \le \epsilon$. Recalling our bounds on $|S_f|$ and c, we have proved:

Theorem 10.4 *Under the uniform distribution on* $\{0,1\}^n$, *for any values* $\epsilon, \delta > 0$, *algorithm* `LearnMonotone` *learns* s*-term monotone DNF in time polynomial in* n, $(s \log \frac{sn}{\epsilon})^{\log \frac{s}{\epsilon} \log \frac{1}{\epsilon}}$ *and* $\log(1/\delta)$.

Taking $s = 2^{O(\sqrt{\log n})}$ we obtain the following corollary:

Corollary 10.5 *For any constant* ϵ *algorithm* `LearnMonotone` *learns* $2^{O(\sqrt{\log n})}$*-term monotone DNF in poly*$(n, \log(1/\delta))$ *time under the uniform distribution.*

As noted earlier, Bshouty and Tamon's algorithm for learning monotone DNF with $O((\log n)^2/(\log\log n)^3)$ terms also requires that ϵ be constant in order to achieve poly(n) runtime.

10.3.4 Learning Monotone Circuits

Let C be the class of depth d, size M circuits which compute monotone functions on r out of n variables. An analysis similar to that of the last section (but simpler since we do not need to introduce auxiliary functions f_1 and f_2) shows that algorithm `LearnMonotone` can be used to learn C. If f is the target function and g is the hypothesis generated by `LearnMonotone`, then as in the last section we have that $\Pr[\text{sign}(g) \neq f] \le X + Y + Z$ with

$$X = \sum_{|A| \le c, A \not\subseteq S_f} \hat{f}(A)^2, \quad Y = \sum_{|A| > c} \hat{f}(A)^2, \quad Z = \sum_{|A| \le c, A \subseteq S_f} (\alpha_A - \hat{f}(A))^2.$$

By using `FindVariables` to identify the "important" relevant variables S_f (of which there are now at most r) we can ensure that $\sum_{|A| \not\subseteq S_f} \hat{f}(A)^2 \le \epsilon/4$ and thus bound X. To bound Y, instead of using Mansour's lemma we we use the main lemma of (Linial *et al.*, 1993) which bounds the total weight of high-order Fourier coefficients for constant-depth circuits:

Lemma 10.6 (Linial *et al.*) *Let f be a Boolean function computed by a circuit of depth d and size M and let c be any integer. Then*

$$\sum_{|A|>c} \hat{f}(A)^2 \leq 2M2^{-c^{1/d}/20}.$$

Finally, as in Section 10.3.3 the bound on Z is obtained by taking a sufficiently large sample to estimate each Fourier coefficient $\hat{f}(A)$ with $A \subseteq S_f, |A| \leq c$ with error at most $\sqrt{\epsilon/4|S_f|^c}$. Taking $m = \text{poly}(r^c, 1/\epsilon, \log(1/\delta))$ and $c = \Theta((\log(M/\epsilon))^d)$ in `LearnMonotone` we obtain:

Theorem 10.7 *Fix $d \geq 1$ and let $C_{d,M,r}$ be the class of depth d, size M circuits which compute monotone functions with r relevant variables out of n. Under the uniform distribution, for any $\epsilon, \delta > 0$, algorithm* `LearnMonotone` *learns class $C_{d,M,r}$ in time polynomial in n, $r^{(\log(M/\epsilon))^d}$ and $\log(1/\delta)$.*

One interesting corollary is the following:

Corollary 10.8 *Fix $d \geq 1$ and let C_d be the class of depth d, size $2^{O((\log n)^{1/(d+1)})}$ circuits which compute monotone functions with $2^{O((\log n)^{1/(d+1)})}$ relevant variables out of n. Then for any constant ϵ algorithm* `LearnMonotone` *learns class C_d in poly$(n, \log(1/\delta))$ time.*

While this class C_d is rather limited from the perspective of Boolean circuit complexity, from a learning theory perspective it is fairly rich. We note that C_d strictly includes the class of depth d, size $2^{O((\log n)^{1/(d+1)})}$ circuits on $2^{O((\log n)^{1/(d+1)})}$ variables which contain only unbounded fanin AND and OR gates. This follows from results of Okol'nishnikova (Okol'nishnikova, 1982) and Ajtai and Gurevich (Ajtai and Gurevich, 1987) (see also (Boppana and Sipser, 1990) Section 3.6) which show that there are monotone functions which can be computed by AC^0 circuits but are not computable by AC^0 circuits which have no negations.

10.4 Product Distributions

A *product distribution* over $\{0,1\}^n$ is characterized by parameters $\mu_1, \ldots, \mu_n$ where $\mu_i = \Pr[x_i = 1]$. Such a distribution $\mathcal{D}$ assigns values independently to each variable, so for $a \in \{0,1\}^n$ we have $\mathcal{D}(a) = \left(\prod_{a_i=1} \mu_i\right)\left(\prod_{a_i=0}(1-\mu_i)\right)$. The uniform distribution is a product distribution with each $\mu_i = 1/2$. The standard deviation of x_i under a product distribution is $\sigma_i = \sqrt{\mu_i(1-\mu_i)}$. A product distribution $\mathcal{D}$ is *constant-bounded* if there is some constant $c \in (0,1)$ independent of n such that $\mu_i \in [c, 1-c]$ for all $i = 1,\ldots,n$. We let β denote $\max_{i=1,\ldots,n}(1/\mu_i, 1/(1-\mu_i))$. Throughout the rest of this paper $\mathcal{D}$ denotes a product distribution.

Given a product distribution $\mathcal{D}$ we define a new inner product over the vector space of real valued functions on $\{0,1\}^n$ as

$$\langle f, g\rangle_{\mathcal{D}} = \sum_{x\in\{0,1\}^n} \mathcal{D}(x)f(x)g(x) = E_{\mathcal{D}}[fg]$$

and a corresponding norm $\|f\|_{\mathcal{D}} = \sqrt{\langle f, f\rangle_{\mathcal{D}}}$. We refer to this norm as *the $\mathcal{D}$-norm.* For $i = 1,\ldots,n$ let $z_i = (x_i - \mu_i)/\sigma_i$. Given $A \subseteq [n]$, let ϕ_A be defined as $\phi_A(x) = \prod_{i\in A} z_i$. As noted by Bahadur (Bahadur, 1961) and Furst *et al.* (Furst *et al.*, 1991), the 2^n functions ϕ_A form an orthonormal basis for the vector space of real valued functions on $\{0,1\}^n$ with respect to the $\mathcal{D}$-norm, i.e. $\langle \phi_A, \phi_B\rangle_{\mathcal{D}}$ is 1 if $A = B$ and is 0 otherwise. We refer to this basis as *the ϕ basis.* The following fact is useful:

Fact 10.9 (Bahadur; Furst *et. al*) *The ϕ basis is the basis which would be obtained by Gram-Schmidt orthonormalization (with respect to the $\mathcal{D}$-norm) of the χ basis performed in order of increasing $|A|$.*

By the orthonormality of the ϕ basis, any real function on $\{0,1\}^n$ can be uniquely expressed as $f(x) = \sum_A \tilde{f}(A)\phi_A(x)$ where $\tilde{f}(A) = \langle f, \phi_A\rangle_{\mathcal{D}}$ is the Fourier coefficient of A with respect to the ϕ basis. Note that we write $\tilde{f}(A)$ for the ϕ basis Fourier coefficient and $\hat{f}(A)$ for the χ basis Fourier coefficient. Also by orthonormality we have Parseval's identity

$$E_{\mathcal{D}}[f^2] = \|f\|_{\mathcal{D}}^2 = \sum_{A\subseteq[n]} \tilde{f}(A)^2$$

which is 1 for Boolean f. Finally, for Boolean f and real-valued g we have ((Furst *et al.*, 1991) Lemma 10)

$$\Pr_{\mathcal{D}}[f \neq \mathrm{sign}(g)] \leq E_{\mathcal{D}}[(f-g)^2]. \tag{10.10}$$

Furst *et al.* (Furst *et al.*, 1991) analyzed the ϕ basis Fourier spectrum of AC^0 functions and gave product distribution analogues of Linial *et al.*'s results on learning AC^0 circuits under the uniform distribution. In Section 10.4.1 we sharpen and extend some results from (Furst *et al.*, 1991), and in Section 10.5 we use these sharpened results together with techniques from (Furst *et al.*, 1991) to obtain product distribution analogues of our algorithms from Section 3.

10.4.1 Some ϕ Basis Fourier Lemmas

A *random restriction* $\rho_{p,\mathcal{D}}$ is a mapping from $\{x_1, \ldots, x_n\}$ to $\{0, 1, *\}$ where x_i is mapped to $*$ with probability p, to 1 with probability $(1-p)\mu_i$, and to 0 with probability $(1-p)(1-\mu_i)$. If f is a Boolean function then $f\lceil\rho$ represents the function $f(\rho_{p,\mathcal{D}}(x))$ whose variables are those x_i which are mapped to $*$ and whose other x_i are instantiated as 0 or 1 according to $\rho_{p,\mathcal{D}}$.

The following is a variant of Hastad's well known switching lemma (Hastad, 1986):

Lemma 10.10 *Let $\mathcal{D}$ be a product distribution with parameters μ_i and β as defined above, let f be a CNF formula where each clause has at most d literals, and let $\rho_{p,\mathcal{D}}$ be a random restriction. Then with probability at least $1-(4\beta pd)^t$,*

1. *the function $f\lceil\rho$ can be expressed as a DNF formula where each term has at most t literals;*
2. *the terms of such a DNF all accept disjoint sets of inputs.*

Proof sketch: The proof is a minor modification of arguments given in Section 4 of (Beame, 1994). ■

The following corollary is a product distribution analogue of ((Linial *et al.*, 1993) Corollary 1):

Corollary 10.11 *Let $\mathcal{D}$ be a product distribution with parameters μ_i and β, let f be a CNF formula where each clause has at most d literals, and let $\rho_{p,\mathcal{D}}$ be a random restriction. Then with probability at least $1-(4\beta pd)^t$ we have that $\widetilde{f\lceil\rho}(A) = 0$ for all $|A| > t$.*

Proof: It is shown in (Linial *et al.*, 1993) that if $f\lceil\rho$ satisfies properties (1) and (2) of Lemma 10.10 then $\widehat{f\lceil\rho}(A) = 0$ for all $|A| > t$. Hence such a $f\lceil\rho$ is in the space spanned by $\{\chi_A : |A| \leq t\}$. By Fact 10.9 and the nature of Gram-Schmidt orthonormalization, this is the same space which is spanned by $\{\phi_A : |A| \leq t\}$, and the corollary follows. ■

Corollary 10.11 is a sharpened version of a similar lemma, implicit in (Furst *et al.*, 1991), which states that under the same conditions with probability at least $1 - (5\beta pd/2)^t$ we have $\widetilde{f\lceil\rho}(A) = 0$ for all $|A| > t^2$. Armed with the sharper Corollary 10.11, by retracing arguments from (Furst *et al.*, 1991) it is straightforward to prove

Lemma 10.12 *For any Boolean function f, for any integer ℓ,*

$$\sum_{|A|>\ell} \tilde{f}(A)^2 \leq 2 \Pr_{\rho_{p,D}}[\widetilde{f\lceil\rho}(A) \neq 0 \textit{ for some } |A| > \ell p/2].$$

Boolean duality implies that the conclusion of Corollary 10.11 also holds if f is a DNF with each term of length at most d. Taking $p = 1/8\beta d$ and $t = \log\frac{4}{\epsilon}$ in this DNF version of Corollary 10.11 and $\ell = 16\beta d \log\frac{4}{\epsilon}$ in Lemma 10.12, we obtain the following analogue of Mansour's lemma (Lemma 10.3) for the ϕ basis:

Lemma 10.13 *Let f be a DNF with terms of size at most d. Then for all $\epsilon > 0$*

$$\sum_{|A|>16\beta d\log(4/\epsilon)} \tilde{f}(A)^2 \leq \epsilon/2.$$

Again using arguments from (Furst *et al.*, 1991), Corollary 10.11 can also be used to prove the following version of the main lemma from (Furst *et al.*, 1991):

Lemma 10.14 *Let f be a Boolean function computed by a circuit of depth d and size M and let c be any integer. Then*

$$\sum_{|A|>c} \tilde{f}(A)^2 \leq 2M2^{-c^{1/d}/8\beta}.$$

The version of this lemma given in (Furst *et al.*, 1991) has $1/(d+2)$ instead of $1/d$ in the exponent of c. This new tighter bound will enable us to give stronger guarantees on our learning algorithm's performance under product distributions than we could have obtained by simply using the lemma from (Furst *et al.*, 1991).

10.5 Learning under Product Distributions

10.5.1 Identifying Relevant Variables

We have the following analogue to Lemma 10.2 for product distributions:

Lemma 10.15 *Let $f : \{0,1\}^n \to \{-1,1\}$ be a monotone Boolean function. There is an algorithm which has access to $EX(f, \mathcal{D})$, runs in poly$(n, \beta, 1/\epsilon, \log 1/\delta)$ time steps for all $\epsilon, \delta > 0$, and with probability at least $1-\delta$ outputs a set $S_f \subseteq [n]$ such that*

$$i \in S_f \text{ implies } \sum_{A: i \in A} \tilde{f}(A)^2 \geq \epsilon/2 \qquad \text{and} \qquad i \notin S_f \text{ implies } \sum_{A: i \in A} \tilde{f}(A)^2 \leq \epsilon.$$

Proof: We use the following fact:

Fact 10.16 ((Bshouty and Tamon, 1996) Lemma 4.1) *For any Boolean function f and any product distribution $\mathcal{D}$,*

$$4\sigma_i^2 I_{\mathcal{D},i}(f) = \sum_{A: i \in A} \tilde{f}(A)^2.$$

Using Chernoff bounds one can easily show that with poly$(n, \beta, 1/\epsilon, \log 1/\delta)$ examples it is possible to estimate each value μ_i (and thus σ_i) with high accuracy. As in Lemma 10.2, using Chernoff bounds it is possible to estimate $I_{\mathcal{D},i}(f)$ with high accuracy again using poly$(n, \beta, 1/\epsilon, \log 1/\delta)$ examples. Combining these estimates for σ_i and $I_{\mathcal{D},i}(f)$ proves the lemma. We call this algorithm `FindVariables2`. ■

10.5.2 The Learning Algorithm

We would like to modify `LearnMonotone` so that it uses the ϕ basis rather than the χ basis. However, as in (Furst *et al.*, 1991) the algorithm does not know the exact values of μ_i so it cannot use exactly the ϕ basis; instead it approximates each μ_i by a sample value μ_i' and uses the resulting basis, which we call the ϕ' basis. In more detail, the algorithm is as follows:

- Use `FindVariables2` to identify a set S_f of important variables.

- Draw m labeled examples $\langle x^1, f(x^1)\rangle, \ldots, \langle x^m, f(x^m)\rangle$ from $EX(f, \mathcal{D})$. Compute $\mu'_i = \frac{1}{m}\sum_{j=1}^{m} x_i^j$ for $1 \leq i \leq n$. Define $z'_i = (x_i - \mu'_i)/\sqrt{\mu'_i(1-\mu'_i)}$ and $\phi'_A = \prod_{i \in A} z'_i$.

- For every $A \subseteq S_f$ with $|A| \leq c$ set $\alpha'_A = \frac{1}{m}\sum_{j=1}^{m} f(x^j)\phi'_A(x^j)$. If $|\alpha'_A| > 1$ set $\alpha'_A = \text{sign}(\alpha'_A)$. For every A such that $|A| > c$ or $A \not\subseteq S_f$ set $\alpha'_A = 0$.

- Output the hypothesis $\text{sign}(g(x))$, where $g(x) = \sum_A \alpha'_A \chi_A(x)$.

We call this algorithm `LearnMonotone2`. As in (Furst *et al.*, 1991) we note that setting α'_A to ± 1 if $|\alpha'_A| > 1$ can only bring the estimated value closer to the true value of $\tilde{f}(A)$.

10.5.3 Learning Monotone $2^{O(\sqrt{\log n})}$-term DNF

For the most part only minor changes to the analysis of Section 10.3.3 are required. As before we suppose that the target concept is an s-term DNF. Since a term of size greater than d is satisfied by a random example from $\mathcal{D}$ with probability less than $(\frac{\beta-1}{\beta})^d$, we now take $\log_{\frac{\beta}{\beta-1}} \frac{32sn}{\epsilon} = \Theta(\beta \log \frac{sn}{\epsilon})$ as the term size bound for f_1. Proceeding as in Section 10.3.3 we obtain $|S_f| = O(\beta s \log \frac{sn}{\epsilon})$. We similarly set a term size bound of $\Theta(\beta \log \frac{s}{\epsilon})$ for f_2. We use the ϕ basis Parseval identity and inequality (10.10) in place of the χ basis identity and inequality (10.2) respectively. Lemma 10.13 provides the required analogue of Mansour's lemma for product distributions; using the new term size bound on f_2 we obtain $c = \Theta(\beta^2 \log \frac{s}{\epsilon} \log \frac{1}{\epsilon})$.

The one new ingredient in the analysis of `LearnMonotone2` comes in bounding the quantity $Z = \sum_{|A| \leq c, A \subseteq S_f} (\alpha'_A - \tilde{f}(A))^2$. In addition to the sampling error which would be present even if μ'_i were exactly μ_i, we must also deal with error due to the fact that α'_A is an estimate of the ϕ' basis coefficient rather than of the ϕ basis coefficient $\tilde{f}(A)$. An analysis entirely similar to that of Section 5.2 of (Furst *et al.*, 1991) shows that taking $m = \text{poly}(c, |S_f|^c, \beta^c, 1/\epsilon, \log(1/\delta))$ suffices to bound $Z \leq \epsilon/4$. We thus have

Theorem 10.17 *Under any product distribution $\mathcal{D}$ on $\{0,1\}^n$, for any $\epsilon, \delta > 0$, algorithm* `LearnMonotone2` *learns s-term monotone DNF in time polynomial in n, $(\beta s \log \frac{sn}{\epsilon})^{\beta^2 \log \frac{s}{\epsilon} \log \frac{1}{\epsilon}}$, and $\log(1/\delta)$.*

Since a constant-bounded product distribution $\mathcal{D}$ has $\beta = \Theta(1)$, we obtain

Corollary 10.18 *For any constant ϵ and any constant-bounded product distribution $\mathcal{D}$, algorithm* `LearnMonotone2` *learns $2^{O(\sqrt{\log n})}$-term monotone DNF in poly$(n, \log(1/\delta))$ time.*

10.5.4 Learning Monotone Circuits

Using Lemma 10.14 and an analysis similar to the above, we obtain the following results.

Theorem 10.19 *Fix $d \geq 1$ and let C be the class of depth d, size M circuits which compute monotone functions on r out of n variables. Under any product distribution $\mathcal{D}$, for any $\epsilon, \delta > 0$, algorithm* `LearnMonotone2` *learns class C in time polynomial in n, $r^{(\beta \log \frac{M}{\epsilon})^d}$ and $\log(1/\delta)$.*

Corollary 10.20 *Fix $d \geq 1$ and let C be the class of depth d, size $2^{O((\log n)^{1/(d+1)})}$ circuits which compute monotone functions on $2^{O((\log n)^{1/(d+1)})}$ variables. Then for any constant ϵ and any constant-bounded product distribution $\mathcal{D}$, algorithm* `LearnMonotone2` *learns class C in poly$(n, \log(1/\delta))$ time.*

Chapter 11

Future Directions

While our results shed new light on several important aspects of computationally efficient learning, many interesting questions remain to be answered.

DNF Learning: The $2^{\tilde{O}(n^{1/3})}$ time complexity of our DNF learning algorithm from Chapter 3 is a significant improvement over previous results, but this running time is still far from polynomial. Can polynomial time algorithms be obtained for the general problem of PAC learning DNF from random examples under an arbitrary distribution? Similarly, while our algorithm in Chapter 10 for learning monotone DNF under the uniform distribution runs in polynomial time for DNF with $2^{O(\sqrt{\log n})}$ terms, it remains an open question to design a polynomial time algorithm for learning arbitrary polynomial size monotone DNF under the uniform distribution.

Computational Sample Complexity: A goal for future research is to replace the nonstandard cryptographic hardness assumption of Section 4.5, which we used to prove that attribute efficient learning can be computationally hard, with a standard assumption. It would also be interesting to extend our cryptographic hardness result for attribute efficient learning to a more natural concept class such as the class of 1-decision lists of length k.

Boosting: The new `SmoothBoost` algorithm introduced in Chapter 9 is guaranteed to generate optimally smooth distributions, but it requires $1/\epsilon\gamma^2$ stages of boosting. As described in Chapters 8 and 9, other boosting algorithms such as Freund's boost-by-majority algorithm and Freund and Schapire's `AdaBoost` are known to run in an optimal $\log(1/\epsilon)/\gamma^2$ stages, but these algorithms do not appear to generate optimally smooth distributions. Is there a single boosting algorithm which is both optimally smooth and optimally fast?

Linear Threshold Learning: Our algorithms in Chapter 9 match the malicious noise tolerance of the best previous algorithms, but the level of malicious noise which they can handle is still significantly smaller than the information-theoretic upper bound. Can efficient linear threshold learning algorithms be designed which have improved tolerance to malicious noise?

References

M. Ajtai and Y. Gurevich. 1987. Monotone versus positive. *Journal of the ACM*, 34(4):1004–1015.

N. Alon, J. Spencer, and P. Erdos. 1992. *The Probabilistic Method.* Wiley-Interscience, New York.

D. Angluin and P. Laird. 1988. Learning from noisy examples. *Machine Learning*, 2:343–370.

D. Angluin. 1988. Queries and concept learning. *Machine Learning*, 2:319–342.

M. Anthony, G. Brightwell, and J. Shawe-Taylor. 1995. On specifying Boolean functions using labelled examples. *Discrete Applied Mathematics*, 61:1–25.

M. Anthony. 1995. Classification by polynomial surfaces. *Discrete Applied Mathematics*, 61:91–103.

E. Artin. 1964. *The Gamma Function.* Holt, Rinehart and Winston, New York.

J. Aslam and S. Decatur. 1998. Specification and simulation of statistical query algorithms for efficiency and noise tolerance. *Journal of Computer and System Sciences*, 56:191–208.

J. Aspnes, R. Beigel, M. Furst, and S. Rudich. 1994. The expressive power of voting polynomials. *Combinatorica*, 14(2):1–14.

P. Auer and N. Cesa-Bianchi. 1998. On-line learning with malicious noise and the closure algorithm. *Annals of Mathematics and Artificial Intelligence*, 23:83–99.

P. Auer and M. Warmuth. 1995. Tracking the best disjunction. In *Proceedings of the Thirty-Sixth Symposium on Foundations of Computer Science*, pages 312–321.

P. Auer. 1997. Learning nested differences in the presence of malicious noise. *Theoretical Computer Science*, 185(1):159–175.

L. Babai, L. Fortnow, N. Nisan, and A.Wigderson. 1993. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318.

R. Bahadur, 1961. *A representation of the joint distribution of responses to n dichotomous items*, In H. Solomon, editor, *Studies in Item Analysis and Prediction*, pages 158–168. Stanford University Press.

P. Bartlett and J. Shawe-Taylor, 1999. *Generalization performance of support vector machines and other pattern classifiers*, In B. Scholkopf, C.J.C. Burges, and A.J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 43–54. MIT Press.

P. Bartlett, P. Long, and R. Williamson. 1996. Fat-shattering and the learnability of real-valued functions. *Journal of Computer and System Sciences*, 52(3):434–452.

E. B. Baum and Y-D. Lyuu. 1991. The transition to perfect generalization in perceptrons. *Neural Computation*, 3:386–401.

E. Baum. 1990. The Perceptron algorithm is fast for nonmalicious distributions. *Neural Computation*, 2:248–260.

P. Beame. 1994. A switching lemma primer. Technical Report UW-CSE-95-07-01, University of Washington.

R. Beigel, N. Reingold, and D. Spielman. 1991. The perceptron strikes back. In *Proceedings of the Sixth Conference on Structure in Complexity Theory*, pages 286–291.

R. Beigel. 1993. The polynomial method in circuit complexity. In *Proceedings of the Eigth Conference on Structure in Complexity Theory*, pages 82–95.

R. Beigel. 2000. Personal communication.

A. Beimel, F. Bergadano, N. Bshouty, E. Kushilevitz, and S. Varricchio. 1996. On the applications of multiplicity automata in learning. In *Proceedings of the Thirty-Seventh Annual Symposium on Foundations of Computer Science*, pages 349–358.

M. Bellare. 1992. A technique for upper bounding the spectral norm with applications to learning. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 62–70.

H. Block. 1962. The Perceptron: a model for brain functioning. *Reviews of Modern Physics*, 34:123–135.

M. Blum and S. Micali. 1984. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864.

A. Blum and S. Rudich. 1995. Fast learning of k-term DNF formulas with queries. *Journal of Computer and System Sciences*, 51(3):367–373.

A. Blum, M. Furst, J. Jackson, M. Kearns, Y. Mansour, and S. Rudich. 1994. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proceedings of the Twenty-Sixth Annual Symposium on Theory of Computing*, pages 253–262.

A. Blum, L. Hellerstein, and N. Littlestone. 1995. Learning in the presence of finitely or infinitely many irrelevant attributes. *Journal of Computer and System Sciences*, 50:32–40.

A. Blum, A. Frieze, R. Kannan, , and S. Vempala. 1997. A polynomial time algorithm for learning noisy linear threshold functions. *Algorithmica*, 22(1/2):35–52.

A. Blum. 1992. Rank-r decision trees are a subclass of r-decision lists. *Information Processing Letters*, 42(4):183–185.

A. Blum. 1996. On-line algorithms in machine learning. available at `http://www.cs.cmu.edu/~avrim/Papers/pubs.html`.

A. Blum. 1997. Empirical support for Winnow and weighted-majority algorithms: results on a calendar scheduling domain. *Machine Learning*, 26:5–23.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. 1987. Occam's razor. *Information Processing Letters*, 24:377–380.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. 1989. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965.

D. Boneh and R. Lipton. 1993. Amplification of weak learning over the uniform distribution. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 347–351.

R. Boppana and M. Sipser, 1990. *The complexity of finite functions*, In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume A*, pages 757–804. MIT Press.

N. Bshouty and C. Tamon. 1996. On the Fourier spectrum of monotone functions. *Journal of the ACM*, 43(4):747–770.

N. Bshouty, J. Jackson, and C. Tamon. 1999. More efficient PAC learning of DNF with membership queries under the uniform distribution. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 286–295.

N. Bshouty. 1995. Exact learning via the monotone theory. *Information and Computation*, 123(1):146–153.

N. Bshouty. 1996. A subexponential exact learning algorithm for DNF using equivalence queries. *Information Processing Letters*, 59:37–39.

N. Bshouty. 1997. Simple learning algorithms using divide and conquer. *Computational Complexity*, 6:174–194.

T. Bylander. 1998a. Learning noisy linear threshold functions. submitted for publication, available at `http://ringer.cs.utsa.edu/research/AI/bylander/pubs/pubs.html`.

T. Bylander. 1998b. Worst-case analysis of the Perceptron and exponentiated update algorithms. *Artificial Intelligence*, 106.

N. Cesa-Bianchi, E. Dichterman, P. Fischer, E. Shamir, and H.U. Simon. 1999. Sample-efficient strategies for learning in the presence of noise. *Journal of the ACM*, 46(5):684–719.

E. Cheney. 1966. *Introduction to approximation theory.* McGraw-Hill, New York, New York.

H. Chernoff. 1952. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statist.*, 23:493–507.

E. Cohen. 1997. Learning noisy perceptrons by a perceptron in polynomial time. In *Proceedings of the Thirty-Eighth Symposium on Foundations of Computer Science*, pages 514–521.

D. Coppersmith and S. Winograd. 1987. Matrix multiplication via arithmetic progressions. In *Proceedings of the Nineteenth Symposium on Theory of Computing*, pages 1–6.

I. Dagan, Y. Karov, and D. Roth. 1997. Mistake-driven learning in text categorization. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 55–63.

A. Deaton and R. Servedio. 2001. Gene structure prediction from many attributes. To appear in *Journal of Computational Biology.*

S. Decatur, O. Goldreich, and D. Ron. 1999. Computational sample complexity. *SIAM Journal on Computing*, 29(3):854–879.

S. Decatur. 1993. Statistical queries and faulty PAC oracles. In *Proceedings of the Sixth Workshop on Computational Learning Theory*, pages 262–268.

M. Dertouzos. 1965. *Threshold logic: a synthesis approach.* MIT Press, Cambridge, MA.

T.G. Dietterich. 2000. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 40(2):139–158.

C. Domingo and O. Watanabe. 2000. MadaBoost: a modified version of AdaBoost. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 180–189.

A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. 1989. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82(3):247–251.

Y. Freund and R. Schapire. 1996. Game theory, on-line prediction and boosting. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, pages 325–332.

Y. Freund and R. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Y. Freund and R. Schapire. 1998. Large margin classification using the Perceptron algorithm. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 209–217.

Y. Freund. 1990. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216.

Y. Freund. 1992. An improved boosting algorithm and its implications on learning complexity. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 391–398.

Y. Freund. 1995. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285.

Y. Freund. 1999. An adaptive version of the boost-by-majority algorithm. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 102–113.

B. Fu. 1992. Separating PH from PP by relativization. *Acta Math. Sinica*, 8(3):329–336.

M. Furst, J. Saxe, and M. Sipser. 1984. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27.

M. Furst, J. Jackson, and S. Smith. 1991. Improved learning of AC^0 functions. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 317–325.

E. Gardner and B. Derrida. 1989. Three unfinished works on the optimal storage capacity of networks. *J. Phys. A: Math. Gen.*, 22:1983–1994.

C. Gentile and N. Littlestone. 1999. The robustness of the p-norm algorithms. In *Proceedings of the 12th Annual Conference on Computational Learning Theory*, pages 1–11.

A.R. Golding and D. Roth. 1999. A Winnow-based approach to spelling correction. *Machine Learning*, 34:107–130.

M. Goldmann, J. Hastad, and A. Razborov. 1992. Majority gates vs. general weighted threshold gates. *Computational Complexity*, 2:277–300.

O. Goldreich and L. Levin. 1989. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual Symposium on Theory of Computing*, pages 25–32.

O. Goldreich, S. Goldwasser, and S. Micali. 1986. How to construct random functions. *Journal of the Association for Computing Machinery*, 33(4):792–807.

O. Goldreich, N. Nisan, and A. Wigderson. 1995. On Yao's XOR-Lemma. Electronic Colloquium on Computational Complexity TR95-050.

O. Goldreich. 1995. Foundations of cryptography (Fragments of a Book). Available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.

O. Goldreich. 1998. *Modern cryptography, probabilistic proofs and pseudo-randomness.* Springer-Verlag, New York.

S. Goldwasser and M. Bellare. 1996. Lecture notes on cryptography. Available at `http://www-cse.ucsd.edu/users/mihir/papers/gb.html`.

A. Grove, N. Littlestone, and D. Schuurmans. 1997. General convergence results for linear discriminant updates. In *Proceedings of the Tenth Annual Conference on Computational Learning Theory*, pages 171–183.

T. Hancock and Y. Mansour. 1991. Learning monotone k-μ DNF formulas on product distributions. In *Proceedings of the Fourth Annual Conference on Computational Learning Theory*, pages 179–193.

T. Hancock. 1992. *The complexity of learning formulas and decision trees that have restricted reads.* Ph.D. thesis, Harvard University.

J. Hastad. 1986. *Computational Limitations for Small Depth Circuits.* MIT Press, Cambridge, MA.

J. Hastad. 1994. On the size of weights for threshold gates. *SIAM Journal on Discrete Mathematics*, 7(3):484–492.

D. Haussler. 1988. Space efficient learning algorithms. Technical Report UCSC-CRL-88-2, University of California at Santa Cruz.

D. Helmbold, R. Sloan, and M. Warmuth. 1990. Learning nested differences of intersection-closed concept classes. *Machine Learning*, 5:165–196.

W. Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30.

R. Impagliazzo and A. Widgerson. 1997. P = BPP unless E has subexponential circuits: derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual Symposium on Theory of Computing*, pages 220–229.

R. Impagliazzo. 1995. Hard-core distributions for somewhat hard problems. In *Proceedings of the Thirty-Sixth Annual Symposium on Foundations of Computer Science*, pages 538–545.

J. Jackson and M. Craven. 1996. Learning sparse perceptrons. In *Advances in Neural Information Processing Systems 8*, pages 654–660.

J. Jackson. 1995. *The Harmonic sieve: a novel application of Fourier analysis to machine learning theory and practice.* Ph.D. thesis, Carnegie Mellon University, August.

J. Jackson. 1997. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55:414–440.

C. Ji and S. Ma. 1997. Combinations of weak classifiers. *IEEE Transactions on Neural Networks*, 8(1):32–42.

J. Kahn, G. Kalai, and N. Linial. 1988. The influence of variables on Boolean functions. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 68–80.

M. Kearns and M. Li. 1993. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837.

M. Kearns and L. Valiant. 1994. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95.

M. Kearns and U. Vazirani. 1994. *An introduction to computational learning theory.* MIT Press, Cambridge, MA.

M. Kearns, M. Li, L. Pitt, and L. Valiant. 1987a. On the learnability of Boolean formulae. In *Proceedings of the Nineteenth Annual Symposium on Theory of Computing*, pages 285–295.

M. Kearns, M. Li, L. Pitt, and L. Valiant. 1987b. Recent results on Boolean concept learning. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 337–352.

M. Kearns, M. Li, and L. Valiant. 1994. Learning Boolean formulas. *Journal of the ACM*, 41(6):1298–1328.

M. Kearns. 1998. Efficient noise-tolerant Learning from statistical queries. *Journal of the ACM*, 45(6):983–1006.

R. Khardon. 1994. On using the Fourier transform to learn disjoint DNF. *Information Processing Letters*, 49:219–222.

M. Kharitonov. 1995. Cryptographic lower bounds for learnability of Boolean functions on the uniform distribution. *Journal of Computer and System Sciences*, 50:600–610.

J. Kivinen, M. Warmuth, and P. Auer. 1997. The Perceptron algorithm vs. Winnow: linear vs. logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, 97(1-2):325–343.

A. Klivans and R. Servedio. 1999. Boosting and hard-core sets. In *Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science*, pages 624–633.

A. Klivans and R. Servedio. 2001. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. To appear in *Proceedings of the Thirty-Third Annual Symposium on Theory of Computing.*

L. Kucera, A. Marchetti-Spaccamela, and M. Protassi. 1994. On learning monotone DNF formulae under uniform distributions. *Information and Computation*, 110:84–95.

E. Kushilevitz. 1997. A simple algorithm for learning $O(\log n)$-term DNF. *Information Processing Letters*, 61(6):289–292.

L. Levin. 1986. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286.

N. Linial and N. Nisan. 1990. Approximate inclusion-exclusion. *Combinatorica*, 10(4):349–365.

N. Linial, Y. Mansour, and N. Nisan. 1993. Constant depth circuits, Fourier transform and learnability. *Journal of the ACM*, 40(3):607–620.

N. Littlestone. 1988. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318.

N. Littlestone. 1989a. From online to batch learning. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 269–284.

N. Littlestone. 1989b. *Mistake bounds and logarithmic linear-threshold learning algorithms.* Ph.D. thesis, University of California at Santa Cruz.

N. Littlestone. 1991. Redundant noisy attributes, attribute errors, and linear-threshold learning using Winnow. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 147–156.

S. Lokam. 2001. Personal communication.

P. Long. 1994. Halfspace learning, linear programming, and nonmalicious distributions. *Information Processing Letters*, 51:245–250.

P. Long. 1995. On the sample complexity of PAC learning halfspaces against the uniform distribution. *IEEE Transactions on Neural Networks*, 6(6):1556–1559.

W. Maass and G. Turan, 1994. *How fast can a threshold gate learn?*, In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning Theory and Natural Learning Systems: Volume I: Constraints and Prospects*, pages 381–414. MIT Press.

W. Maass and M. Warmuth. 1998. Efficient learning with virtual threshold gates. *Information and Computation*, 141(1):378–386.

Y. Mansour and M. Parnas. 1998. Learning conjunctions with noise under product distributions. *Information Processing Letters*, 68(4):189–196.

Y. Mansour. 1995. An $O(n^{\log \log n})$ learning algorithm for DNF under the uniform distribution. *Journal of Computer and System Sciences*, 50:543–550.

M. Minsky and S. Papert. 1968. *Perceptrons: an introduction to computational geometry*. MIT Press, Cambridge, MA.

S. Muroga. 1971. *Threshold logic and its applications*. Wiley-Interscience, New York.

N. Nisan and A. Wigderson. 1994. Hardness versus randomness. *Journal of Computer and System Sciences*, 49:149–167.

A. Novikoff. 1962. On convergence proofs on perceptrons. In *Proceedings of the Symposium on Mathematical Theory of Automata*, volume XII, pages 615–622.

E. Okol'nishnikova. 1982. On the influence of negations on the complexity of a realization of monotone Boolean functions by formulas of bounded depth (in Russian). *Metody Diskret. Analiz.*, 38:74–80.

M. Opper and D. Haussler. 1991. Calculation of the learning curve of Bayes optimal classification algorithm for learning a perceptron with noise. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 75–87.

I. Parberry. 1994. *Circuit complexity and neural networks*. MIT Press, Cambridge, MA.

L. Pitt and L. Valiant. 1988. Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984.

M. Rabin. 1979. Digitalized signatures as intractable as factorization. Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology.

R. Rivest. 1987. Learning decision lists. *Machine Learning*, 2(3):229–246.

F. Rosenblatt. 1962. *Principles of neurodynamics*. Springer-Verlag, New York.

Y. Sakai and A. Maruoka. 2000. Learning monotone log-term DNF formulas under the uniform distribution. *Theory of Computing Systems*, 33:17–33.

R. Schapire and Y. Singer. 1998. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91.

R. Schapire, Y. Freund, P. Bartlett, and W. Lee. 1998. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686.

R. Schapire. 1990. The strength of weak learnability. *Machine Learning*, 5(2):197–227.

R. Schapire. 1999a. Drifting games. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 114–124.

R. Schapire. 1999b. Theoretical views of boosting. In *Proceedings of the Tenth International Conference on Algorithmic Learning Theory*, pages 12–24.

M. Schmitt. 1998. Identification criteria and lower bounds for Perceptron-like learning rules. *Neural Computation*, 10:235–250.

R. Servedio. 1999a. Computational sample complexity and attribute-efficient learning. In *Proceedings of the Thirty-First Annual Symposium on Theory of Computing*, pages 701–710.

R. Servedio. 1999b. On PAC learning using Winnow, Perceptron, and a Perceptron-like algorithm. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 296–307.

R. Servedio. 2000a. Computational sample complexity and attribute-efficient learning. *Journal of Computer and System Sciences*, 60(1):161–178.

R. Servedio. 2000b. PAC analogues of Perceptron and Winnow via boosting the margin. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 148–157.

R. Servedio. 2001a. On learning monotone DNF under product distributions. To appear in *Proceedings of the Fourteenth Annual Conference on Computational Learning Theory*.

R. Servedio. 2001b. PAC analogues of Perceptron and Winnow via boosting the margin. To appear in *Machine Learning*.

R. Servedio. 2001c. Smooth boosting and learning with malicious noise. To appear in *Proceedings of the Fourteenth Annual Conference on Computational Learning Theory*.

H. Seung, H. Sompolinsky, and N. Tishby. 1992. Statistical mechanics of learning from examples. *Physical Review A*, 25(8):6056–6091.

J. Shawe-Taylor, P. Bartlett, R. Williamson, and M. Anthony. 1998. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940.

M. Sipser and D. Spielman. 1996. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722.

D. Spielman. 1996. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731.

J. Tarui and T. Tsukiji. 1999. Learning DNF by approximating inclusion-exclusion formulae. In *Proceedings of the Fourteenth Conference on Computational Complexity*, pages 215–220.

A. Taylor and W. Mann. 1972. *Advanced Calculus.* Wiley & Sons.

R. Uehara, K. Tsuchida, and I. Wegener. 1997. Optimal attribute-efficient learning of disjunction, parity, and threshold functions. In *Proceedings of the Third European Conference on Computational Learning Theory*, pages 171–184.

P. Vaidya. 1989. A new algorithm for minimizing convex functions over convex sets. In *Proceedings of the Thirtheth Symposium on Foundations of Computer Science*, pages 338–343.

L. Valiant. 1984. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

L. Valiant. 1999. Projection learning. *Machine Learning*, 37(2):115–130.

V. Vapnik. 1998. *Statistical Learning Theory.* Wiley-Interscience, New York.

K. Verbeurgt. 1990. Learning DNF under the uniform distribution in quasi-polynomial time. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 314–326.

K. Verbeurgt. 1998. Learning sub-classes of monotone DNF on the uniform distribution. In *Proceedings of the Ninth Conference on Algorithmic Learning Theory*, pages 385–399.

A. Wigderson. 1999. Personal communication.