The Harmonic Sieve: A Novel Application of Fourier Analysis to Machine Learning Theory and Practice

Jeffrey Charles Jackson

August, 1995

CMU-CS-95-183

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:

Merrick Furst, Chair
Avrim Blum
Steven Rudich
Yishay Mansour, Tel Aviv University

© 1995 Jeffrey Charles Jackson

This research is sponsored by NSF under grant no. CCR-9119319. Views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the United States Government.

Abstract

This thesis presents new positive results—both theoretical and empirical—in machine learning. The primary learning-theoretic contribution is the Harmonic Sieve, the first efficient algorithm for learning the well-studied class of Disjunctive Normal Form (DNF) expressions (learning is accomplished within the Probably Approximately Correct model with respect to the uniform distribution using membership queries). Of particular interest is the novel use of Fourier methods within the algorithm. Specifically, all prior Fourier-based learning algorithms focused on finding large Fourier coefficients of the function to be learned (the target). The Harmonic Sieve departs from this paradigm; it instead learns by finding large coefficients of certain functions other than the target. The robustness of this new Fourier technique is illustrated by applying it to prove learnability of noisy DNF expressions, of a circuit class that is even more expressive than DNF, and of an interesting class of geometric concepts.

Empirically, the thesis demonstrates the significant practical potential of a classification-learning algorithm closely related to the Harmonic Sieve. The Boosting-based Perceptron (BBP) learning algorithm produces classifiers that are nonlinear perceptrons (weighted thresholds over higher-order features). On several previously-studied machine learning benchmarks, the BBP algorithm produces classifiers that achieve accuracies essentially equivalent to or even better than the best previously-reported classifiers. Additionally, the perceptrons produced by the BBP algorithm tend to be relatively intelligible, an important feature in many machine learning applications. In a related vein, BBP and the Harmonic Sieve are applied successfully to the problem of rule extraction, that is, the problem of approximating an unintelligible classifier by a more intelligible function.

Contents

List of Figures

List of Tables

Acknowledgments

While my advisor would be the first to tell me to make clear that the theoretical results reported in this thesis are my own, I believe it is also true that I would never have obtained these results without Merrick Furst's guidance over the last—ahem—seven years. Merrick recognized the potential of the main tool used to obtain my thesis results (Fourier analysis of Boolean functions) long before I did, and carefully guided my early research in this area. He has also been *consistently* encouraging and supportive, as well as a superb mentor to me in a number of non-technical areas. For example, you probably don't want to know what I was planning to call the main algorithm of this thesis before Merrick suggested the name "Harmonic Sieve"...

My research has also been strongly influenced by Avrim Blum, who has for several years been much like a second advisor to me. A great deal of what I know about learning theory has come through my interactions with Avrim. He has also been a wonderful source of stimulating problems and research directions; in fact, I believe Avrim was largely responsible for my interest in the central theoretical problem addressed by this thesis. Both Avrim and Merrick gave particularly helpful comments on early drafts of the results reported on in this thesis, and Avrim also suggested a simplification to my original proof of Fact 8.1.

Steven Rudich also deserves special mention. On the technical side, an insight of Steven's inspired the initial proof of Fact 6.1. Perhaps more importantly, a frank discussion with Steven may have done as much as anything to motivate me to solve the core learnability problem of this thesis! His courses in complexity and cryptography were excellent character-building experiences: his homeworks are humbling to say the least. And finally, his comments on an earlier draft of the thesis played an important role in the framing of the final document.

Yishay Mansour has long been supportive of my work on applying Fourier techniques in machine learning, which I appreciate probably more than he realizes. Yishay also suggested considering the extension to the main thesis result discussed in Chapter 7. Finally, he advised me not to spend too much time writing acknowledgments, so...

The empirical results reported in this thesis represent joint work with Mark Craven. I appreciate very much his help with filling in my knowledge of techniques and results in empirical machine learning and his willingness to speculate on the practical viability of the Harmonic Sieve.

A number of others have contributed in various ways to this thesis. Mike Kearns, along with the first four people mentioned above, was a co-author with me of a "weak" learning result on which the main theoretical thesis result is built; his enthusiasm about my results has also been a wonderful encouragement. Dick Lipton suggested considering the extension of the Harmonic Sieve to geometric concepts (Chapter 8). Yoav Freund graciously provided unpublished details about his boosting algorithm. Rob Schapire suggested that the Harmonic Sieve should be capable of learning a certain class of function representation (the class of DNF expressions defined over parity functions; see Chapter 7). Doug Tygar helped with the proof of Lemma 8.2. Andrew Tomkins shared very helpful comments during the initial development of the Harmonic Sieve. Charlie Rackoff supplied useful pointers to some related literature.

Carnegie Mellon's School of Computer Science has been a tremendously stimulating and enjoyable place to engage in graduate studies. In addition to those mentioned above (and with apologies to anyone I am inadvertently omitting), I especially thank Randy Bryant, Prasad Chalasani, Lonnie Chrisman, Ed Clarke, Steve Guattery, Somesh Jha, Sergei Nirenburg, Jiří Sgall, Sean Smith, Shang-Hua Teng, and Xudong Zhao for interesting discussions and/or collaborations. Through their coursework and other interactions, Alan Frieze, Ravi Kanaan, Gary Miller, and Danny Sleator have broadened my knowledge of theoretical computer science.

I also appreciate very much the opportunities I have had to present my work and engage in related research in other environments. Thanks especially to Nader Bshouty and Hans Simon for hosting extended visits at the Universities of Calgary and Dortmund, respectively, and also thanks for stimulating discussions to Andreas Birkendorf, Richard Cleve, Eli Dichterman, Paul Fischer, Norbert Klasner, and Tino Tamon.

Finally, I am very grateful to my wife Cindy and children Rebecca, Peter, Emily, and Benjamin for embarking on this great adventure with me—or perhaps I should say, all too often without me!

Chapter 1

Introduction

The starting point for the research described in this thesis is a question that many studying the theory of machine learning have pondered: in what sense, if any, is the class of Disjunctive Normal Form (DNF) expressions learnable? (A Boolean function is expressed in Disjunctive Normal Form if it is written as an OR of AND's of (possibly negated) Boolean inputs; see Figure 1.1.) A number of factors have led to the strong interest in DNF learnability. An early motivation was that DNF expressions can succinctly represent many types of human knowledge, and these representations seem to be readily understood. For example, consider the following instructions adopted from the 1994 Form 1040 tax publication, a publication that is intended to be read by a very wide audience:

```
If as of December 31, 1994, you:
   had never been married
      or
   had been married
   and were then legally separated
      or
   had been married
   and were widowed before January 1, 1994
   and were not remarried in 1994
then you may file as a single taxpayer.
```

The fact that DNF expressions are easily understood motivated a conjecture that machine-based learning systems should be able to learn DNF expressions [Val84]. Since that conjecture, there has been a great deal of research related to DNF learnability [Val85, KLPV87,

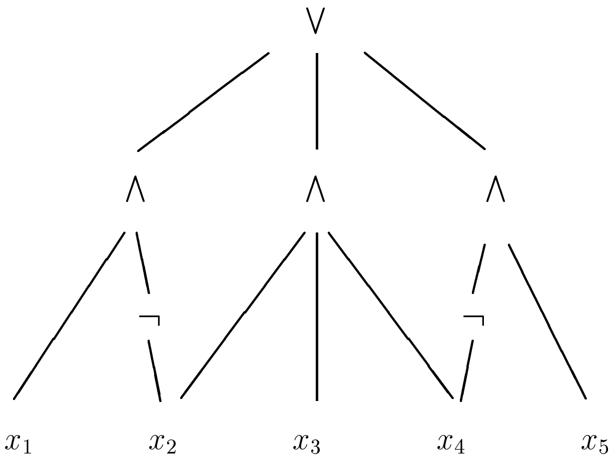

Figure 1.1: Graphical representation of the DNF expression $(x_1 \wedge \overline{x_2}) \vee (x_2 \wedge x_3 \wedge x_4) \vee (\overline{x_4} \wedge x_5)$. The function represented by each of the AND gates is a *term* of the DNF.

Ang90, AFP90, Ver90, AP91, AK91, Han91, KM93, AHP92, AP92, BR92, Man92, Bsh93, Han93, KR93, BKK$^+$94, SM94, Bsh95]. However, most of the positive results of this research only applied to various *restricted* forms of DNF; for example, it was shown that monotone DNF, in which no negated variables are allowed in the DNF expression, is learnable. There were also a few algorithms that could learn DNF in slightly superpolynomial time. But prior to the results of this thesis there was no provably efficient (polynomial-time) algorithm for learning arbitrary DNF expressions.

1.1 Learning DNF Efficiently: The Harmonic Sieve

This thesis presents the Harmonic Sieve (`HS`), an algorithm that efficiently learns DNF. Specifically, we show that the `HS` algorithm learns DNF with respect to the uniform distribution using membership queries in the Probably Approximately Correct (PAC) model of learning.[1] This model of learning is defined formally in Chapter 3; for now, we give an informal but reasonably precise definition. In particular, consider the following 20-questions-like game between two players, Bob and Alice. Bob secretly chooses an arbitrary DNF expression f. Alice is then allowed to ask about the value of f on a number of inputs that is bounded

[1]It should be noted that most of the earlier restricted DNF algorithms learn in a stronger sense than `HS`. Therefore, while a few of the earlier algorithms are corollaries of the thesis results, most of the prior DNF-related algorithms are not implied by our results.

by a polynomial in the size of f (the number of characters required to write f down), and the total processing time she is allowed is similarly bounded. Alice wins the game if she produces a function h (not necessarily a DNF) such that, over the domain of f, h agrees with f 99% of the time. This thesis presents a winning strategy for Alice. That is, regardless of the expression f chosen by Bob, Alice can win the game by running `HS` (or more precisely, win with very high probability over the random choices made by the algorithm).

While we find it convenient to describe the result in terms of a game, we also believe that the Harmonic Sieve and the ideas underlying it have the potential for significant real-world application. Specifically, "Bob" above should be thought of as representing some real-world process that associates inputs with Boolean outcomes. An example real-world process we will consider in detail later in the thesis comes from computational biology. The promoter-recognition task is to identify those regions of DNA that immediately precede genes. It is known that the sequence of nucleotides in such a promoter region distinguishes it from non-promoter DNA regions; that is, there is a function that maps input data consisting of (a Boolean representation of) sufficiently long DNA sequences to the Boolean outcome "gene begins immediately after this sequence." If there is a small DNF expression describing this real-world input/output relationship and if there is some entity willing to answer hypothetical questions about the relationship (in this example, such data could conceivably be collected by laboratory experiments), then the Harmonic Sieve can be used to learn to classify a very high percentage of randomly-generated DNA sequences with the same classification assigned by the real-world process.

1.2 Algorithmic Techniques

The primary tool used to obtain our DNF learnability result is Fourier analysis. Fourier techniques were first applied to problems in machine learning theory by Linial, Mansour, and Nisan [LMN93] and have since been used extensively [Bel91, AM91, FJS91, KM93, Bel92, Man92, BFJ$^+$94, BJ95, BT95]. However, not even Fourier methods had produced an efficient learning algorithm for DNF. In fact, Mansour had shown that to prove DNF learnability by

Fourier analysis would require something beyond then-known techniques [Man92].

This thesis presents a novel Fourier analysis that overcomes the limitations of earlier Fourier techniques. At a high level, the algorithm resulting from this new analysis can be thought of as implementing a search strategy that differs markedly from early Fourier search strategies. More specifically, all previous Fourier-based learning algorithms implemented a strategy for finding the so-called "large Fourier coefficients" of the function f to be learned (the meaning of the quoted term is not important in the current discussion; it is defined in Chapter 4). The Harmonic Sieve searches for not only the large Fourier coefficients of f, but also the large coefficients of certain *other* functions. While it is perhaps counterintuitive that this approach should facilitate the learning of f, our Fourier analysis proves that it does. The discovery of this new paradigm for Fourier-based learning is a major contribution of this thesis. The proof that this approach is appropriate for learning DNF relies on a newly-discovered relationship—that may be of independent interest—between DNF expressions and parity functions.

Another algorithmic aspect of the Harmonic Sieve that is worth noting is that it is based on a *hypothesis boosting* mechanism, in particular one due to Freund [Fre90]. Such mechanisms are designed to boost a weak learner (one that produces a classifier that is correct, say, 51% of the time) into a learner of the type described earlier (one that produces a classifier that is correct, say, 99% of the time, a so-called *strong* learner). Hypothesis boosting is a powerful idea, and versions of boosting have been applied successfully in practice [DSS93]. Boosting has also been used to derive provably efficient and noise-tolerant learning algorithms [AD93]. However, DNF appears to be the first class of expressions that has been shown to be efficiently learnable using hypothesis boosting and for which no alternative learnability proof was previously known.

It is also interesting to note that we use boosting to obtain a *distribution-dependent* result. That is, our learning algorithm produces a classifier c that agrees well with the function f in the sense that if an input x is drawn uniformly at random then with high probability $c(x) = f(x)$. However, if the requirement was to find a c that agreed well with f with respect to some other probability distribution over the inputs, the Harmonic Sieve would *not* be guaranteed to meet this requirement. It is in this sense that `HS` is a distribution-dependent algorithm. Hypothesis boosting, on the other hand, was originally designed to boost distribution-*independent* weak learning algorithms into distribution-independent strong learners (distribution-independent learning is explained more fully below). Nevertheless, we show how to use an existing boosting mechanism (with minor modifications) to boost a distribution-dependent weak learning algorithm into a strong distribution-dependent learner for DNF.

The minor modification mentioned above also merits some attention. As originally envisioned, boosting would be applied to a weak algorithm that learned from examples. Such a weak learner is given a large set of examples $\{\langle x^i, f(x^i)\rangle\}$ of the function f to be learned, where each x^i is drawn at random according to some fixed distribution D over the inputs to f, and produces a classifier c such that the probability of agreement of f and c over inputs drawn at random according to D is at least, say, 0.51. Any weak algorithm that can learn in this way is a *distribution-independent* weak learner. Note that the learner is not explicitly told the distribution D against which its output classifier c will be tested, but is only given information about D implicitly through the set of examples provided. It is at this point that we modify the boosting mechanism: our booster supplies the distribution D to the weak learner explicitly. In particular, we will supply a function (the *distribution oracle*) to the weak learner that, given an input x, returns the probability weight assigned to x by D.

This change to the boosting mechanism is important because the weak learning algorithm for DNF described in this thesis requires access to distribution oracles in order to learn efficiently with respect to the distributions of interest.[2] As far as we know, the notion of providing a distribution oracle to a learning algorithm and the demonstration that such oracles can facilitate efficient learning are further original contributions of this thesis.

[2]While the strong DNF learner produces a classifier the performs well with respect to uniform, the boosting mechanism requires that the weak learner perform well with respect to a variety of nonuniform distributions.

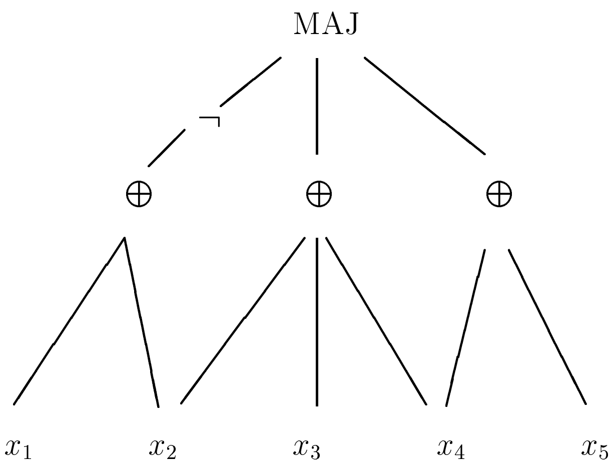

Figure 1.2: Graphical representation of TOP expression MAJ($\overline{x_1 \oplus x_2}, x_2 \oplus x_3 \oplus x_4, x_4 \oplus x_5$). In general, a Threshold-of-Parity (TOP) representation of a Boolean function f is a majority vote over a collection of (possibly negated) parity functions, where each parity is applied to a subset of f's input variables (parity applied to a cardinality-1 subset is the function represented by the single variable in that subset, and parity of the empty subset is a constant). The above function is `true` on an assignment x if at least two of the three (a majority) of the parity functions are made `true` by the assignment; the all-1's vector is such an assignment.

1.3 Extensions to the Harmonic Sieve

Actually, the Harmonic Sieve is more than a DNF learning algorithm. This same algorithm can learn another class of function representations, the Threshold-of-Parity, or TOP, class (Figure 1.2); in fact, the output of the Harmonic Sieve is always a TOP. TOP is a more succinct class of function representations than DNF in the sense that any DNF expression can be rewritten as a TOP with only a polynomial increase in size [KP94], while even a single n-bit parity requires 2^{n-1} terms in any DNF representation. Recalling that Alice's processing time is bounded by a polynomial in, among other parameters, the size of Bob's representation of the function f to be learned, we see that TOP is more difficult to learn than DNF.

To further demonstrate the robustness of the ideas underlying the Harmonic Sieve, we extend the algorithm in a variety of ways. For example, the Harmonic Sieve is designed to learn Boolean functions defined over the domain $\{0,1\}^n$ of n-bit Boolean strings. However, the algorithm can also be extended to learn certain classes of Boolean functions defined

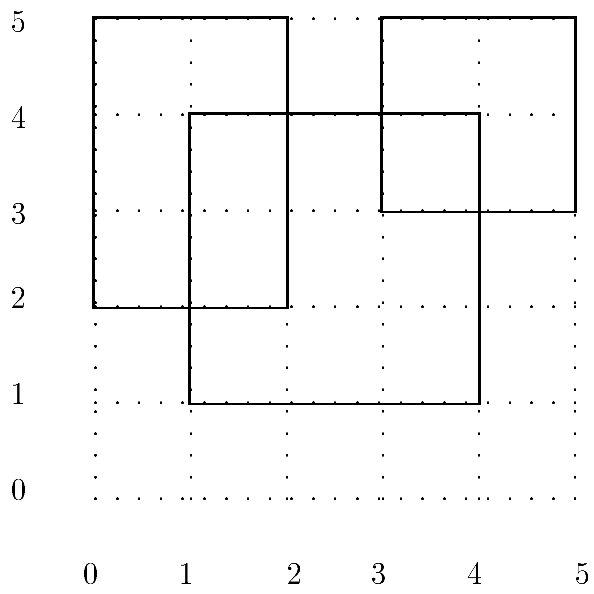

Figure 1.3: Graphical representation of a UBOX function over $\{0,\ldots,5\}^2$ consisting of three rectangles. The function value is `true` on those input vectors that fall on or within any of the rectangles.

over the domain $\{0,\ldots,b-1\}^n$, where b is any positive integer. In particular, we show that for b constant, the class UBOX of unions of axis-parallel rectangles (Figure 1.3) can be learned efficiently in the same uniform-plus-membership-query model of learning for which the Harmonic Sieve is designed. Previous algorithms for learning UBOX (in stronger learning models) are polynomial-time only when the dimensionality n or the number of rectangles is held constant [BEHW89, LW90, GGM94, BGGM94]; our algorithm is polynomial-time in both of these parameters but has superpolynomial dependence on b.

Extending in another direction, we show that the distribution dependence of `HS` can be relaxed somewhat. Our main result is that DNF can be learned with respect to certain *product distributions*. Product distributions are a natural generalization of the uniform distribution. To draw an n-bit string x uniformly at random, we can flip n fair coins and assign each bit of x accordingly. A product distribution is defined similarly, except now each coin is assigned a fixed bias representing the probability of producing a head. We show that DNF is learnable with respect to any *constant-bounded* product distribution D, *i.e.*, any product distribution having the property that all of the biases defining D differ from $\frac{1}{2}$ by no more than a fixed constant $c < \frac{1}{2}$.

Finally, we also show that HS is able to tolerate well certain types of errors in the information it receives about the Boolean function f to be learned. In terms of the game representation of the learning model, we show that DNF is learnable even if Bob frequently lies about the value of f. More specifically, each time Bob is asked about the value of f on an input x, he first checks to see if he has been asked about x before. If so, he gives the same answer given previously. If not, Bob flips a coin with fixed bias strictly less than $\frac{1}{2}$. If this coin comes up heads, Bob lies about the value of $f(x)$, otherwise he tells the correct value. Alice is now allowed more running time as the bias of Bob's coin nears $\frac{1}{2}$ (it is information-theoretically impossible to learn f if the a noise rate is $\frac{1}{2}$). Alice's goal remains to produce an approximation to the *noiseless* f. A formal definition of this model of learning in the presence of *persistent random classification noise* is given in Chapter 10.

Notice that if Alice is given rather noisy data describing a function f then if she produces a classifier c that closely approximates f, c will not agree especially well with the noisy data. That is, in some sense what Alice hopes to produce is a classifier that captures the underlying structure of the data she is given and not the noisy aspects of the data. This is very similar to the goal in much of applied machine learning, where in many domains it is assumed that there exists a relatively simple function approximating a given data set representing some real-world phenomenon, but that there is no simple function exactly fitting the data. The goal in this setting is also to find the simple approximating function rather than a perfect classifier for the given data.

Thus we have some reason to hope that an algorithm that tolerates noise well will also perform well on many real-world learning problems. Of course, only careful experiments can determine how well a learning algorithm actually performs on real-world data, which leads us to experiment with the Harmonic Sieve and a related learning algorithm.

1.4 Rule Extraction

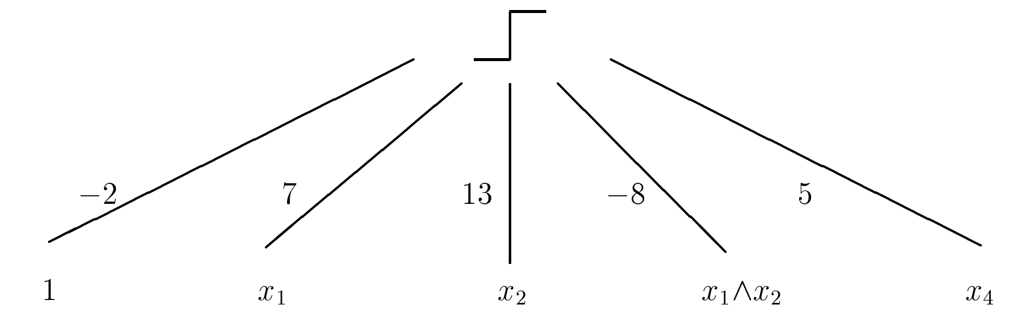

Figure 1.4: Graphical representation of perceptron $\text{sign}(-2+7x_1+13x_2-8(x_1 \wedge x_2)+5x_4)$. A perceptron is a threshold over the original input variables as well as over functions of limited numbers of these variables.

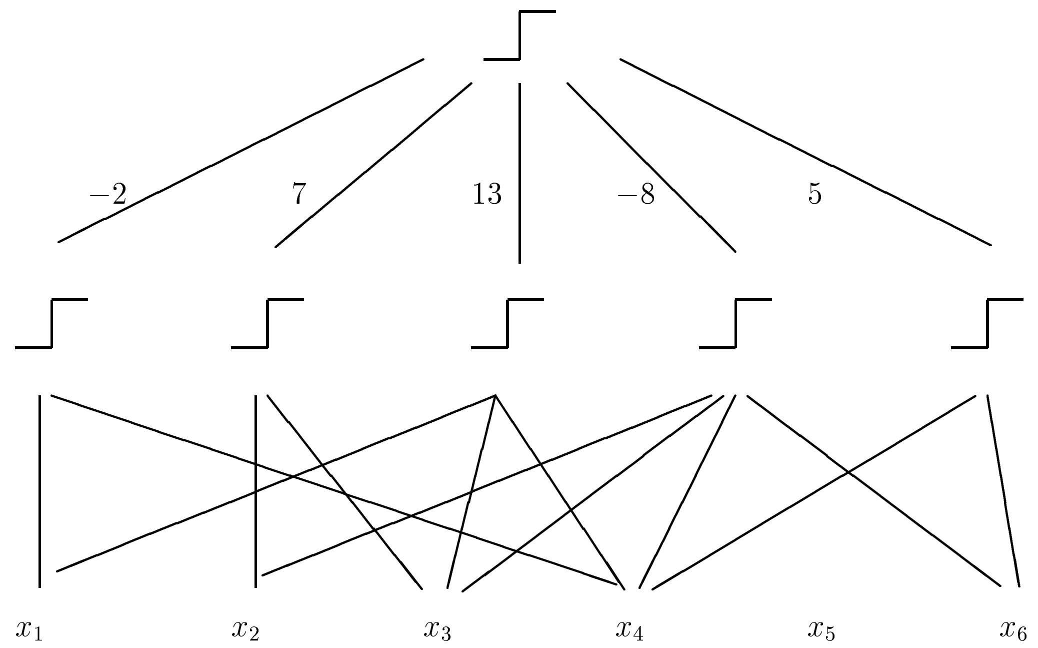

Figure 1.5: Graphical representation of a multilayer perceptron (lower weights suppressed). Typically the lowest level perceptrons are defined over the original inputs only.

First, we consider using `HS` as a means of *rule extraction*. The rule extraction problem is roughly the following. In a number of application domains, the best-performing concept-learning algorithms produce rules in a form that is very difficult for humans to understand (*e.g.*, multilayer perceptrons, or "neural networks"; see Figure 1.4 and Figure 1.5). Of course, there are many reasons to desire a rule which is easy to understand: deeper understanding of the process being modeled by the rule, increased confidence in the predictive power of the rule, less time and memory required to compute the prediction of the rule, etc. The rule extraction problem, then, is to take a rule which is relatively opaque and approximate it by a second function which is more intelligible.

The Harmonic Sieve can be used to address the rule extraction problem in the following way. Since we are given a function f (the opaque rule) as input, we can run the `HS` algorithm, using the opaque representation of f to answer the questions asked by `HS` about f. If `HS` produces a function that is a good approximation to the original function and that is easier to understand, then we will have succeeded in extracting an intelligible rule from an opaque one. We describe a simple modification to the Harmonic Sieve that biases it toward producing understandable approximators, and we examine the performance of this modified algorithm on a well-known benchmark learning problem (predicting party affiliation based on Congressional voting record). We find, as hoped, that our approximator is generally much simpler than the neural net from which it was derived. An interesting side benefit is that this simple function actually performs (very slightly) better than the original network!

Although in general `HS` can output arbitrary TOP representations, on this benchmark the TOP's produced by the biased `HS` have parity gates with fanin at most two. This leads us to experiment with a second algorithm that is restricted to produce only such limited TOP's. This algorithm is more efficient than `HS`, and performs rule extraction reasonably well on a larger benchmark problem from computational biology (the promoter recognition task mentioned earlier). In fact, unlike the more general Harmonic Sieve, this restricted algorithm does not need to ask questions about the function to be learned, but instead can learn from examples alone. This leads us to pursue applications beyond rule extraction for this algorithm.

1.5 Learning Sparse Perceptrons

The *supervised classification* learning task is: given a set of classified data—such as a set of DNA sequences, each sequence classified as either a promoter region or nonpromoter—learn a classification function c that can accurately classify future data. The classification learning task is an important and well-studied problem in applied machine learning, and a number of algorithms have been shown to perform this task well in practice. This thesis develops a new classification learning algorithm that appears to have certain advantages over previous algorithms.

In particular, we develop a new practical perceptron learning algorithm which is based on the hypothesis boosting mechanism underlying the Harmonic Sieve as well as on other concepts developed in the thesis. A distinguishing feature of this Boosting-based Perceptron (BBP) learning algorithm with respect to other algorithms for learning perceptrons—such as the classical perceptron learning algorithm [Ros62], the backpropagation learning algorithm [RHW86], and Winnow [Lit88]—is that it is *constructive* in the following sense. Typical perceptron learning algorithms assign nonzero weights to all of the *features* (input variables or small functions of these variables) and vary these weights during a training phase. At the end of training, all features will in general have nonzero weight. In contrast, the boosting-based algorithm adds features one at a time to the rule constructed during training. An outgrowth of this methodology is that, loosely speaking, if the function to be learned is representable as a sparse perceptron (one with few nonzero weights) then the BBP algorithm will produce a sparse perceptron as its output.

Thus it is to be expected that the BBP algorithm may have intelligibility advantages over many existing classification learning algorithms, as it may produce smaller and therefore more understandable classifiers. What is perhaps more surprising is how well the perceptrons learned by this algorithm perform. On three different benchmark learning problems, the boosting-based algorithm did indeed produce relatively small and intelligible perceptrons; but in addition, these perceptrons performed essentially as well as or better than the classifiers produced by the previous best-performing alternative general-purpose learning methods. Specifically, the performance of our algorithm compared very favorably with neural network learning, and in general was noticeably better than the backpropagation algorithm for learning perceptrons as well as another well-known classification learning algorithm, C4.5. Furthermore, the results on one of the tasks suggest that the algorithm is successfully selecting a small set of the most relevant features from a large set of possible features.

1.6 Summary and Thesis Organization

Summarizing, this thesis contributes a number of new results. We consider the following three results the primary contributions of the thesis:

- Introduction of a new Fourier-based learning strategy: find large Fourier coefficients of certain functions other than the function to be learned.

- Application of this strategy to the well-studied open problem of DNF learnability, resulting in the Harmonic Sieve algorithm for learning DNF and, more generally, TOP expressions.

- Development of the related Boosting-based Perceptron learning algorithm and empirical assessment of `BBP` in several previously-studied application domains, showing that `BBP` can produce classifiers that are relatively accurate, capture relevant features well, and are reasonably intelligible.

Secondarily, the thesis also contains the following results:

- Discovery of a relationship between DNF expressions and parity functions.

- Demonstration that boosting can be used to prove positive learnability results for classes of function representations for which no alternate proof of learnability is known.

- Demonstration that a (slightly modified) off-the-shelf boosting algorithm can be used to boost a distribution-dependent weak learner.

- Introduction of the notion of learning from a distribution oracle.

- Extension of the Harmonic Sieve in various ways, showing its robustness.

- Demonstration empirically that both the Harmonic Sieve and `BBP` algorithms can successfully perform rule extraction.

The remainder of the thesis presents details of these results and is organized as follows. After some mathematical background in Chapter 2, the thesis is divided into two parts, the first part dealing with our learning-theoretic results and the second part with applications of these results. Within Part I, Chapter 3 presents some of the history of the DNF learning problem. It also contains formal definitions for a number of models of learning and discusses the relative advantages of various models. In Chapter 4 we define the multidimensional discrete Fourier transform that is a key tool in obtaining our results. We explain how this transform relates to the more familiar one-dimensional Fourier transform and describe several useful properties of the multidimensional version. We then discuss a number of learning algorithms which have employed this transform. We will focus on one such algorithm—discovered by Goldreich and Levin [GL89] and introduced to learning theory by Kushilevitz and Mansour [KM93]—that plays an integral role in the Harmonic Sieve. Chapter 5 discusses at length the hypothesis boosting mechanism used by `HS`.

In Chapter 6 we prove the existence of a certain relationship between DNF expressions and parity functions, use this relationship to develop a Fourier-based algorithm for weakly learning DNF, and show how to apply boosting to the weak algorithm to produce a strong DNF learning algorithm, the Harmonic Sieve. We conclude the chapter by comparing the Harmonic Sieve with other Fourier-based learning results. In Chapter 7 we show that the Harmonic Sieve can learn any TOP function in time polynomial in the number of parities in the function. As mentioned earlier, this is actually a stronger result than the fact that DNF is learnable, since TOP is a more succinct class of representations than DNF. Results further relating TOP to other classes of representations are also presented. We close this chapter by showing that some of these classes, even though conceptually very similar to DNF and TOP, are *not* efficiently learnable by the Harmonic Sieve. In the next several chapters we consider the previously-discussed extensions of the Harmonic Sieve to learning certain geometric concepts (Chapter 8), learning DNF with respect to certain nonuniform distributions (Chapter 9), and learning the DNF and TOP classes when the algorithm receives incorrect answers to some of the questions it asks (Chapter 10).

We then turn to the second part of the thesis, which explores applications of the theoretical ideas developed in Part I. We begin in Chapter 11 by describing several learning applications: prediction of party affiliation from Congressional voting records and two problems from computational biology. Chapter 12 demonstrates the applicability of modified versions of the Harmonic Sieve to the rule extraction problem in two of these domains, and Chapter 13 presents the boosting-based perceptron learning algorithm and analyzes its performance on all three benchmark learning problems. The thesis closes with a brief discussion of further research directions.

Chapter 2

Mathematical Preliminaries

The main purpose of this chapter is to present a collection of lemmas that are often referred to generically as *Chernoff bounds*. These lemmas will help us to obtain bounds on the running times of our learning algorithms. We will also present some miscellaneous mathematical notation.

2.1 Estimating Expected Values

We will frequently want to estimate the expected value of random variables. The following three lemmas allow us to compute the sample size needed to ensure that with high probability the sample mean closely approximates the true mean of a random variable. The lemmas differ in the restrictions placed on the random variable and in how the closeness of approximation is measured (additively or multiplicatively).

Lemma 2.1 (Chernoff) *Let $X_1, X_2, \ldots, X_m$ be independent $\{0,1\}$ random variables all with mean μ. Then for any $0 \leq \lambda \leq 1$,*

$$\mathbf{Pr}\left[\frac{1}{m}\sum_{i=1}^{m} X_i \geq (1+\lambda)\mu\right] \leq e^{-\lambda^2 m\mu/3}$$

and

$$\mathbf{Pr}\left[\frac{1}{m}\sum_{i=1}^{m} X_i \leq (1-\lambda)\mu\right] \leq e^{-\lambda^2 m\mu/2}.$$

Lemma 2.2 (Hoeffding) *Let $X_1, X_2, \ldots, X_m$ be independent random variables all with mean μ such that for all i, $a \leq X_i \leq b$. Then for any $\lambda > 0$,*

$$\mathbf{Pr}\left[\left|\frac{1}{m}\sum_{i=1}^{m} X_i - \mu\right| \geq \lambda\right] \leq 2e^{-2\lambda^2 m/(b-a)^2}.$$

Lemma 2.3 (Bienaymé-Chebyschev) *Let $X_1, X_2, \ldots, X_m$ be pairwise independent random variables all with mean μ and variance σ^2. Then for any $\lambda > 0$,*

$$\mathbf{Pr}\left[\left|\frac{1}{m}\sum_{i=1}^{m} X_i - \mu\right| \geq \lambda\right] \leq \frac{\sigma^2}{m\lambda^2}.$$

Hoeffding's inequality will be particularly useful to us. For later reference, we state the following immediate corollary of Lemma 2.2:

Corollary 2.1 *Let X represent a random variable that produces values in the range $[a, b]$. Then there is an algorithm* `AMEAN`$(X, b-a, \lambda, \delta)$ *that for any such X produces a value μ' such that $|\mathbf{E}[X] - \mu'| \leq \lambda$ with probability at least $1 - \delta$. Furthermore,* `AMEAN` *runs in time polynomial in $b - a$, λ^{-1}, and $\log \delta^{-1}$.*

Proof: Take $m = (b-a)^2 \ln(2/\delta)/(2\lambda^2)$ and let S represent the sum of m draws from the random variable X. Then $\mu' = S/m$ satisfies the requirements of the corollary. $\Box$

The Chernoff bound can be used to obtain a similar algorithm for estimating the mean of a $\{0, 1\}$-valued random variable within a multiplicative factor. At times we will want to similarly estimate the mean of an $[a, b]$-valued random variable within a multiplicative factor, rather than within an additive factor as above. The following corollary shows that under certain conditions this estimate can be performed efficiently using a standard guess-and-double technique, a technique that is also used frequently elsewhere in the thesis.

Corollary 2.2 *Let X represent a random variable that produces values in the range $[a, b]$, and let $\mathbf{E}[X] = \mu$. Then there is an algorithm* `MMEAN`$(X, b-a, \delta)$ *that for any such X produces a value μ' such that $|\mu - \mu'| \leq \mu/2$ with probability at least $1 - \delta$. Furthermore, when* `MMEAN` *is successful (i.e., with probability at least $1 - \delta$) it runs in time polynomial in $|\mu|^{-1}$, $b - a$, and $\log \delta^{-1}$.*

Proof: The idea behind MMEAN is the following. By Hoeffding, drawing $m \geq 8(b-a)^2 \ln(2/\delta)/(\mu^2)$ random examples of X is sufficient to produce μ' such that $|\mu' - \mu| \leq \mu/4$ with probability at least $1-\delta$ (for readability, we will not use absolute value bars on μ when it is clear from context that we are referring to its magnitude). Of course, we do not know μ and therefore we cannot directly compute the required sample size m. However, if we compute m using a value $\mu_g < \mu$ then we will still get, with high probability, an estimate μ' within $\mu/4$ of the true mean. On the other hand, if we happen to use a value μ_g for μ that is much larger than μ—say $\mu_g = 4\mu$—and attempt the same estimation using μ_g to select the number of samples, then with high probability we will produce an estimate μ' that is much smaller than $3\mu_g/4$. Such an event is evidence that our "guess" μ_g is much larger than μ; therefore, we can refine our guess and try again. In this way we can start with a very poor guess μ_g for μ and yet quickly converge on an adequate guess, which will lead to a good estimate μ' for μ.

Specifically, we begin by taking $\mu_g = \frac{1}{2}$ and computing $\mu' = \texttt{AMEAN}(X, b-a, \mu_g/4, \delta/2)$. If $|\mu'| \geq 3\mu_g/4$ then $\mu \geq \mu_g/2$ and therefore $|\mu' - \mu| \leq \mu/2$ with probability at least $1-\delta/2$. However, if $|\mu'| < 3\mu_g/4$ then we halve both μ_g and δ and again estimate μ'. By the reasoning above, this procedure will, with probability at least $1-\delta$, terminate after at most $\lceil \log(\mu^{-1}) \rceil$ steps and will produce μ' such that $|\mu' - \mu| \leq \mu/2$. $\square$

2.2 Some Notation

We will use the following notation at various points in the thesis. For any complex-valued function g we define the norms $L_\infty(g) = \max_x\{|g(x)|\}$, $L_1(g) = \sum_x |g(x)|$, and $L_2(f) = \sqrt{\sum_x |g|^2(x)}$, where x ranges over the entire (assumed finite) domain of g. At times we will refer to the norm of a set instead of a function, with the obvious intended meaning. We will typically be interested in learning functions with domain $\{0,1\}^n$ for some fixed value of n. For $x \in \{0,1\}^n$ we use x_i to denote the ith bit of x. For a complex variable $g = a + b\sqrt{-1}$, we denote the complex conjugate of g by $g^* = a - b\sqrt{-1}$. Logarithms denoted by log are base 2 and by ln are base e. A function $f : \{0,1\}^n \to \{0,1\}$ is *monotone* if for all $x \in \{0,1\}^n$,

$f(x) = 1 \Rightarrow f(y) = 1$ for any $y \in \{0,1\}^n$ such that $y_i = 1$ for all i such that $x_i = 1$. Real-valued functions f and g over the positive reals satisfy $f = O(g)$ if there exist finite constants c_1 and c_2 such that for all $\rho > 0$, $f(\rho) \leq c_1 g(\rho) + c_2$. Alternatively, under the same circumstances we can write $g = \Omega(f)$. The function $f(\rho)$ over the positive reals is polynomial in its parameter ρ if there exists a constant c such that $f(\rho) = O(\rho^c)$.

Part I

Learning-Theoretic Results

Chapter 3

The DNF Learning Problem

> "[DNF] expressions appear particularly easy for humans to comprehend. Hence we expect that any practical learning system would have to allow for them." [Val84]

The publication of Valiant's "A Theory of the Learnable" in 1984 sparked significant interest in what has come to be known as computational learning theory. Arguably the major contribution of Valiant's paper was a framework within which formal questions about learnability can be studied. Within this framework, Valiant posed and answered a number of learnability questions.

However, as the quote above indicates, one learnability question Valiant posed seemed particularly critical: how can Disjunctive Normal Form (DNF; see Figure 1.1) expressions be efficiently learned? An initial motivation for studying this question was that DNF rules seem to be relatively easy for humans to understand and communicate, and therefore it seems natural to require that a good machine learning system be able to learn DNF rules. As Valiant also noted, "The question of whether [a DNF learning algorithm] exists is tantalizing because of its apparent simplicity." Another reason for studying DNF learnability is that DNF expressions are widely used in computer science, particularly in circuit design. Thus any knowledge we gain about DNF in the process of searching for a DNF learning algorithm may have ramifications for other areas of computer science as well.

Subsequent to Valiant's initial work there has been sustained research on questions about DNF learnability [Val85, KLPV87, Ang90, AFP90, Ver90, AP91, AK91, Han91, KM93, AHP92, AP92, BR92, Man92, Bsh93, Han93, KR93, BKK+94, SM94, Bsh95]. This research has produced results in a variety of models of learning, but for the most part these results have only given partial answers to the question of DNF learnability in the various models studied. Furthermore, for the two models in which the question of DNF learnability had been resolved prior to the results of this thesis, the answers were negative. That is, there was no known efficient algorithm for learning DNF in *any* of the standard learning models.

The theoretical focus of this thesis is on producing such an algorithm. Actually, we present two such algorithms, although the first of these algorithms is interesting primarily for historical reasons now that the second, stronger algorithm is known. In this chapter we present formal definitions of a number of models of learning, including the specific models within which our algorithms learn DNF expressions. We then review some of the prior work on DNF learnability and discuss the relationship of this work with the thesis results.

3.1 Models of Learning

3.1.1 Preliminary Definitions

In this thesis we will primarily be interested in the efficient learnability of Boolean functions, where efficiency is measured relative to the *size* of the function when represented as a Disjunctive Normal Form expression. A DNF expression is a disjunction of terms, where each term is a conjunction of literals and a literal is either a variable or its negation. The *size of a DNF expression* r is the number of terms in r.[1] The *DNF-size of a function* f is the size of the smallest DNF expression representing f. Every Boolean function over $\{0,1\}^n$ can be represented as a DNF; we call function representations that have this property *universal* representations. Thus what we mean by "DNF is learnable" is that arbitrary Boolean functions with domain the Boolean hypercube $\{0,1\}^n$ are learnable in time polynomial in their

[1] This is the standard definition of the size of a DNF expression. We gave a a different, polynomially-related definition in the Introduction for simplicity of exposition.

DNF-size. We call the domain of the function f to be learned its *instance space*. An element x in the instance space is called an *instance*, and the pair $\langle x, f(x)\rangle$ is called an *example* of f.

We also consider the learnability of arbitrary Boolean functions when the size of the function is measured in terms of something other than the function's size as a DNF. One measure of particular interest is when the size is measured in terms of a Threshold-of-Parities, or TOP, representation (Figure 1.2). Any Boolean function f on $\{0,1\}^n$ can be computed by taking a majority vote over a fixed collection of parity functions, *i.e.*, TOP is, like DNF, a universal representation. By a *parity function* we mean a function that computes parity over a fixed subset of the unnegated inputs to f or the negation of such a function; for reasons explained later, we will assume that these functions output values in $\{-1,+1\}$. Parity over the empty set is by definition the constant function 1. Also, a parity function may occur multiple times in a TOP. The *TOP-size* of a function f is the number of parity functions in the smallest TOP computing f. Equivalently, we can think of a TOP as an integer-weighted vote over parity functions; in this case, the TOP-size of a representation is the sum of the magnitudes of the weights.

Given a Boolean function f and probability distribution D on the instance space of f, we say that Boolean function h is an *ϵ-approximator for f with respect to D* if $\mathbf{Pr}_{x\in_R D}[h(x) = f(x)] \geq 1-\epsilon$. The notation here represents the probability that $h = f$ over instances drawn at random according to D; we will typically instead write $\mathbf{Pr}_D[h = f]$. An *example oracle for f with respect to D* ($EX(f,D)$) is an oracle that on request draws an instance x at random according to probability distribution D and returns the example $\langle x, f(x)\rangle$. A *membership oracle for f* ($MEM(f)$) is an oracle that given any instance x returns the value $f(x)$. Queries to these oracles are called example and membership queries, respectively.

3.1.2 PAC Learning

Of the models of learning Valiant introduced [Val84], the model of learning that is perhaps the best known and most studied is what is now called the Probably Approximately Correct (PAC) model. This model is defined as follows. Let ϵ and δ be positive values (called the *accuracy* and *confidence* of the learning procedure, respectively). Then we say that a class $\mathcal{F}$ of representations of functions (*e.g.*, the class DNF) is *PAC learnable* if there is an algorithm $\mathcal{A}$ such that for any ϵ and δ, any $f \in \mathcal{F}$ (the *target function*), and any distribution D on the instance space of f (the *target distribution*), with probability at least $1 - \delta$ algorithm $\mathcal{A}(EX(f, D), \epsilon, \delta)$ produces an ϵ-approximation for f with respect to D. Furthermore, we require that when $\mathcal{A}$ succeeds at producing such an approximator, $\mathcal{A}$'s running time must be bounded by a polynomial in n, the $\mathcal{F}$-size of f, $1/\epsilon$, and $1/\delta$. We generally drop the "PAC" from "PAC learnable" when the model of learning is clear from context.

Note that we used above the notation $f \in \mathcal{F}$ to denote a function f represented by a member of the representation class $\mathcal{F}$. Of course, if $\mathcal{F}$ is a universal representation class, such as DNF, then the requirement that the target be representable by a member of $\mathcal{F}$ does not restrict the choice of target in any way. For other, non-universal representation classes $\mathcal{F}$, several of which will be defined below, the requirement $f \in \mathcal{F}$ *does* restrict the choice of target function. We sometimes refer to the functions represented by such a class as "$\mathcal{F}$ functions." Further note that at times we will also use the notation $f \in \mathcal{F}$ to mean a *representation* f in $\mathcal{F}$ rather than the function represented by f. Which meaning we intend will be clear from context.

A number of variations on this basic PAC model have been studied. For example, let $\mathcal{M}$ represent a learning model, such as PAC, and let $\mathcal{F}$ and $\mathcal{H}$ be representation classes. We say that $\mathcal{F}$ *is $\mathcal{M}$-learnable by $\mathcal{H}$* if $\mathcal{F}$ is $\mathcal{M}$-learnable by an algorithm $\mathcal{A}$ that always outputs a function representation $h \in \mathcal{H}$. If $\mathcal{F}$ is $\mathcal{M}$-learnable by $\mathcal{F}$ then we say that $\mathcal{F}$ is *properly* $\mathcal{M}$-learnable.

A particularly important variation on learning models is allowing the learning algorithm access to a membership oracle. Specifically, if $\mathcal{F}$ is $\mathcal{M}$-learnable by an algorithm that also uses a polynomially-bounded number of membership queries (bounded in the same parameters as $\mathcal{M}$-algorithms that do not use membership queries), we say that $\mathcal{F}$ is *$\mathcal{M}$-learnable with membership queries*. Another variation on learning models is to restrict the choice of distributions D against which examples will be drawn and the hypothesis tested. Let $\mathcal{D}_n$

denote a nonempty set of probability distributions on $\{0,1\}^n$. By a *distribution class* we mean any set $\mathcal{D} = \cup_n \mathcal{D}_n$. Then we say that $\mathcal{F}$ *is* $\mathcal{M}$*-learnable with respect to distribution class* $\mathcal{D}$ if $\mathcal{F}$ is $\mathcal{M}$-learnable for all target distributions D in $\mathcal{D}$ but is not necessarily learnable for arbitrary D. For some distributions that have obvious generalizations to distribution classes (such as the uniform distribution on $\{0,1\}^n$), we blur the distinction between distribution and distribution class and say simply that $\mathcal{F}$ is learnable with respect to the distribution rather than with respect to the distribution class. For purposes of differentiation, we sometimes refer to models such as the basic PAC model that place no restrictions on the distribution D as *distribution-independent* models of learning, and to models that do impose such restrictions as *distribution-dependent*.

A further useful learning variation is so-called *weak learning* [KV89] (in contrast, the PAC model above is sometimes called *strong learning*). If $\mathcal{F}$ is $\mathcal{M}$-learnable for $\epsilon = 1/2 - 1/p(n,s)$, where p is a fixed polynomial and s is the $\mathcal{F}$-size of f, then $\mathcal{F}$ is *weakly* $\mathcal{M}$*-learnable*. Since a fair coin is a $\frac{1}{2}$-approximator for any representation class, weak learning is essentially asking only that we produce a hypothesis that is slightly (but detectably) more accurate than random guessing. In fact, as the learning problem becomes more difficult (*i.e.*, as the size of the target function and number of input bits increase), even less accuracy is required of the hypothesis.

The original motivation for considering weak learning was the following. First, the definition is strong enough that it is possible to show that some representation classes are not weakly learnable given standard complexity theoretic/cryptographic assumptions (arbitrary Boolean circuits are one example [Val84]). On the other hand, the definition is enough weaker than the definition of strong learning that it seemed likely some representation classes could be shown to be weakly learnable which are not strongly learnable. Somewhat surprisingly, it has been shown that for the PAC and PAC with membership queries models, weak learnability implies strong learnability [Sch90]. However, these results do not in general extend to distribution-dependent PAC learning. For example, let $\mathcal{F}$ be a class of representations that is hard (given certain assumptions) to weakly learn, even with respect to uniform (some examples are given in a later section). Now consider the class $\mathcal{F}^+$ obtained by conjoining a new variable with each of the representations in $\mathcal{F}$, *e.g.*, $\mathcal{F}^+ = \{x_0 \wedge f \mid f \in \mathcal{F}\}$. This class is easy to learn weakly with respect to uniform but, given the hardness of weakly learning $\mathcal{F}$, is not learnable in a strong sense with respect to uniform.

Thus distribution-dependent boosting is not possible in general. However, in this thesis we will use an existing distribution-independent boosting technique to convert a distribution-dependent weak learner for DNF into a distribution-dependent strong learner for DNF. We are not aware of any other proof of learnability of a class based on applying a standard distribution-independent boosting algorithm to the task of boosting a distribution-dependent weak learner.

3.1.3 Exact Identification

Another standard model of learning is due to Angluin [Ang88] and is sometimes referred to as the Exact Identification model. Here the goal is not to approximate the target function f but to produce a hypothesis h that is functionally equivalent to f, *i.e.*, an h such that for all x, $f(x) = h(x)$. In general, to accomplish efficient learning in this model requires a more powerful oracle than the ones we have already considered. The specific oracle we will use is an *equivalence oracle for* f $(EQ(f))$, an oracle that, given a hypothesis function h, returns an instance x such that $f(x) \neq h(x)$ if such an x exists; otherwise, the oracle returns "equivalent." A representation class $\mathcal{F}$ is *exactly identifiable* if there is an algorithm $\mathcal{A}$ such that for all $f \in \mathcal{F}$, $\mathcal{A}(EQ(f))$ runs in time polynomial in n and the $\mathcal{F}$-size of f and returns a hypothesis h equivalent to f. Representation class $\mathcal{F}$ is *exactly identifiable with proper equivalence queries* if $\mathcal{F}$ is exactly identifiable by some algorithm $\mathcal{A}$ such that for all of the hypotheses h posed by $\mathcal{A}$ to the equivalence oracle, $h \in \mathcal{F}$.

A number of learnability results have been produced for the exact identification model; several of them are mentioned in the next section. One of the reasons for this interest is the close relationship between this model and PAC learning. In particular, if a class $\mathcal{F}$ is exactly identifiable (resp., using membership queries) then $\mathcal{F}$ is PAC learnable (resp., using membership queries) [Ang88]. Thus positive results for the exact identification model immediately give positive results for the PAC model as well. Also, essentially all classes that are PAC learnable are also known to be exactly identifiable, although it is possible to construct cryptographically-inspired classes that separate the models [Blu90].

3.2 Prior DNF Learning Results

With the previous definitions in hand, we now briefly review some of the history of the DNF learning problem.

3.2.1 Learning from Examples

As mentioned at the beginning of this chapter, Valiant first considered the question of whether or not DNF is PAC learnable [Val84]. He noted that given a different measure of the complexity of a DNF—one that will in general give more time to a learning algorithm than our size measure—it is possible to learn DNF using a certain nonstandard and particularly powerful oracle. He also gave algorithms for learning two subclasses of DNF. By a *subclass* of DNF we mean a representation class consisting of DNF expressions which are syntactically restricted in some way. Valiant proved the learnability of the subclass k-DNF of DNF expressions with at most a constant k many literals per term, and of the subclass monotone DNF that consists of all DNF expressions having no negated literals. His algorithm for the latter class used an extended form of membership queries; Angluin [Ang88] subsequently showed that standard membership queries suffice. Furthermore, Valiant considered the dual question of learning CNF (AND of OR's) expressions; note that since the negation of a CNF expression is a DNF of the same size, a learning algorithm for either class can be used to learn the other. This immediately gives that the CNF subclass k-CNF is also learnable. However, he left open the "tantalizing" question of whether the unrestricted class DNF (with standard size measure) was learnable in any reasonable model of learning. Further motivation to focus on DNF learning was provided in a follow-on paper [Val85] in which Valiant presented some evidence that learning Boolean classes more expressive than DNF might be intractable.

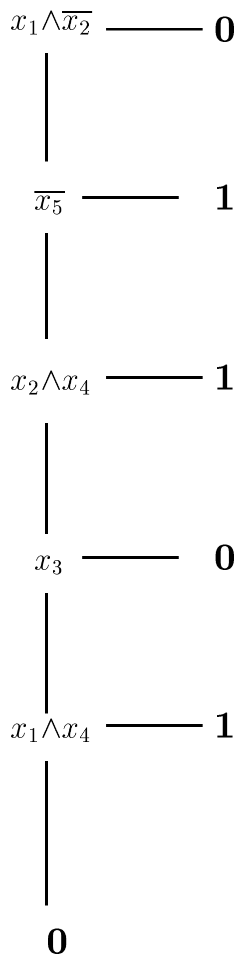

Figure 3.1: A 2-decision list. The value of the function on an input x is computed by beginning at the top of the list and outputting the value associated with the first term satisfied by x. If no term is satisfied then the value at the bottom of the list is output. For example, the input $x = 01111$ first satisfies the term $x_2 \wedge x_4$, and therefore the output of the list on this input is 1. In general, a k-decision list is similar, but each term is allowed up to k literals.

Several subsequent results, both positive and negative, continued the work begun by Valiant on PAC learning of syntactically-restricted subclasses of DNF. Kearns *et al.* [KLPV87] showed that for any constant k, the representation classes $\{f \vee g \mid f \in k\text{-CNF}, g \in k\text{-DNF}\}$ and $\{f \wedge g \mid f \in k\text{-CNF}, g \in k\text{-DNF}\}$ are PAC learnable. These authors also first proved that monotone-DNF is "group-learnable" with respect to the uniform distribution (group learning was later shown equivalent to weak learning, and monotone functions were shown to be weakly learnable with respect to uniform *regardless* of the size measure used [KV89]). Kearns *et al.* further pointed out that the set of functions with representations as k-term-DNF is a subset of the set of functions expressible in k-CNF. Therefore, for constant k, k-term-DNF is learnable by k-CNF. However, in a negative vein, they showed that k-term-DNF is not properly learnable unless $RP = NP$. Rivest [Riv87] proved that the class of functions expressible as k-decision lists (Figure 3.1), a class that contains both k-DNF and k-CNF, is PAC learnable for constant k.

Some other early work produced *quasipolynomial time* (*e.g.*, $n^{\text{polylog}\, n}$ time) algorithms for classes related to DNF.[2] One important example of a class learnable in quasipolynomial time is the well-studied class of functions expressible as Boolean decision trees (Figure 3.2). Decision trees are also a subclass of DNF, as every decision tree can be immediately converted to a DNF expression of the same size or smaller and thus can be viewed as a restricted way of expressing DNF. Due to this subclass relationship, any algorithm that efficiently learns DNF also efficiently (non-properly) learns Boolean decision trees. On the other hand, there are Boolean functions f such that the decision-tree-size of f is exponentially larger than its DNF-size. To see the latter claim, consider a DNF of the form $(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \ldots$.

Ehrenfeucht and Haussler [EH89] showed that the class of Boolean decision trees is PAC learnable in time $n^{O(\log n)}$. (A nice proof of this fact also follows by combining results of Blum [Blu92] and Rivest [Riv87]). Ehrenfeucht and Haussler also showed that the DNF subclass of polynomial-size DNF expressions that also have polynomial CNF-size is PAC

[2]Here and elsewhere in this chapter we will generally for simplicity drop the ϵ, δ, and function-size factors from our bounds. Specifically, unless otherwise noted, our bounds assume that the other factors are polynomial in n.

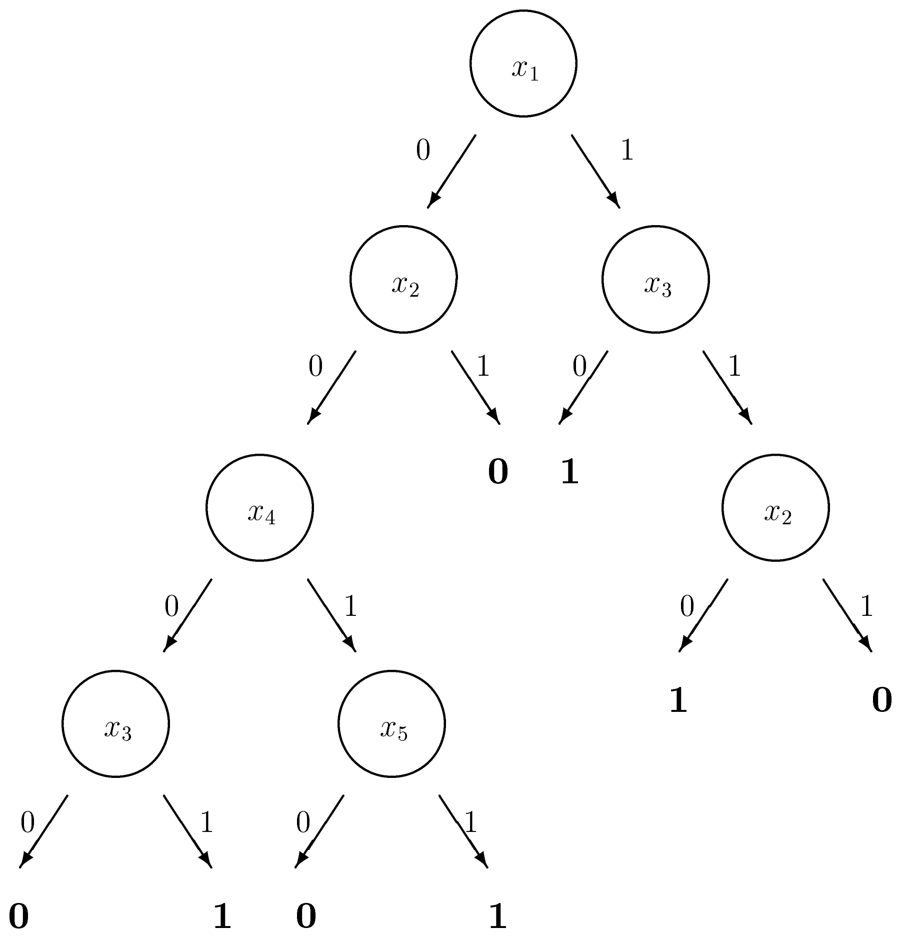

Figure 3.2: A Boolean decision tree. The value of the function on an input x is computed by beginning at the root of the tree and tracing a path to a leaf in the obvious way. The value of this leaf is the value of the function. For example, the input $x = 00001$ defines a path to the leftmost leaf, and therefore the output of the tree on this input is 0. A DNF representation can be immediately obtained for any decision tree by creating a term for each path to a 1-leaf; for this tree, the corresponding DNF representation is $(\overline{x_1} \wedge \overline{x_2} \wedge x_3 \wedge \overline{x_4}) \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_4 \wedge x_5) \vee (x_1 \wedge \overline{x_3}) \vee (x_1 \wedge \overline{x_2} \wedge x_3)$. By definition, the decision-tree-size of a function is the number of leaves in its smallest decision tree representation. Thus by the construction above, every function has decision-tree-size at least as large as its DNF-size.

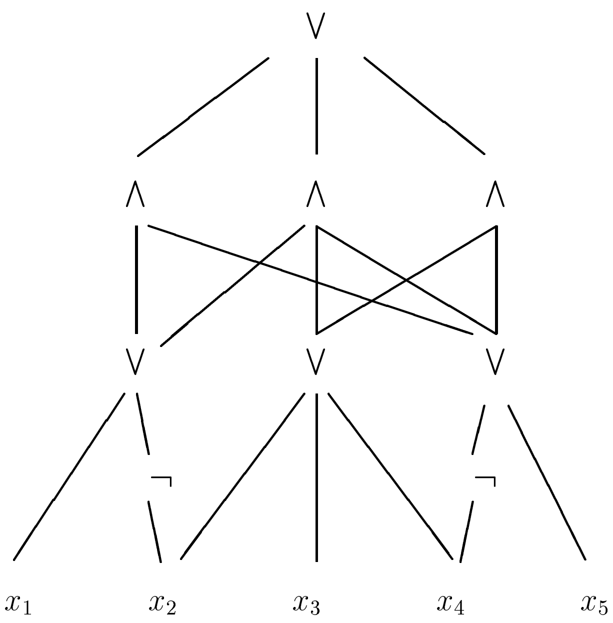

Figure 3.3: A depth-3 Boolean circuit. The size of such a circuit is the number of gates in it. In general, an AC^0 circuit is a generalization of DNF and CNF (depth-2 circuits) that includes a constant number of additional alternating levels of AND's and OR's and is of size polynomial in the number of inputs n.

learnable in time $n^{O(\log^2 n)}$; put another way, every Boolean function is PAC learnable in time quasipolynomial in the larger of the CNF- and DNF-sizes of the function.

In a paper that introduced the use of Fourier techniques for learning problems, Linial *et al.* [LMN93] showed a learnability result for a class which generalizes DNF and CNF by allowing additional levels of OR's and AND's (Figure 3.3). Specifically, they showed that the class of depth-d circuits is PAC learnable with respect to uniform in time roughly $n^{O(\log^d n)}$. The restriction of this class to polynomial-size circuits (which is assumed in this bound, as noted earlier) is known as the class AC^0; we will have more to say about AC^0 below. Subsequent to the Linial *et al.* result, Verbeurgt [Ver90] showed that DNF is learnable with respect to uniform in time $n^{O(\log s/\epsilon)}$, where s is the DNF-size of the target. This result does not require Fourier analysis; the essential observation is that dropping from a DNF expression f all its terms of size at least $\log(s/\epsilon)$ produces an expression f' that is an ϵ-approximator to f. These were the first positive results for the unrestricted class of DNF expressions.

However, there was (and is) also some reason to believe that DNF might not be efficiently PAC learnable, even with respect to the uniform distribution. Perhaps the strongest such evidence to date is a result of Blum *et al.* [BFJ$^+$94] showing that in a particular model of learning from "noisy" data (the Statistical Query model of Kearns [Kea93]) DNF is not learnable with respect to uniform. Because of the close connection between this noise model and the PAC learning model—almost every representation class known to be PAC learnable is learnable in the Statistical Query model—this result strongly suggests that DNF is not PAC learnable.

3.2.2 Learning with Membership Queries

Most of the above results apply to learning from an example oracle alone, which was a dominant concern in much of the early learning-theoretic work. Angluin was largely responsible for demonstrating the utility of learning from alternate oracles. For example, she showed that many PAC-learnable classes, such as k-DNF and k-CNF, are also learnable from the equivalence oracle presented earlier, thus generalizing somewhat the positive PAC results for these classes.

However, perhaps more important than the introduction of the equivalence oracle was Angluin's work demonstrating the power to be gained by using membership queries. For example, as noted above, she showed that monotone DNF can be exactly learned from proper equivalence and membership queries [Ang88], in contrast with her subsequent proof that monotone DNF (and therefore unrestricted DNF as well) is *not* learnable from proper equivalence queries alone [Ang90]. Furthermore, k-term-DNF (and k-term-CNF) were shown to be learnable from proper equivalence queries and membership queries [Ang88], in stark contrast to the proper PAC-learning hardness results for these classes without membership queries cited above. Also in the model of learning from proper equivalence plus membership queries, joint work with Frazier and Pitt [AFP90] showed that conjunctions of Horn clauses are learnable. This implied that the dual class—the subclass of DNF's in which each term contains at most one negated literal—is also learnable using membership and proper equivalence queries.

Subsequent work by other researchers has shown that membership queries do indeed seem to provide a great deal of help in learning many representation classes. Kushilevitz and Mansour [KM93] showed that the class of parity decision trees (a generalization of Boolean decision trees that allows arbitrary parity functions at the internal nodes of the tree) is PAC learnable with respect to the uniform distribution using membership queries. Unlike the class of Boolean decision trees, the parity decision tree class is not a subclass of DNF as it includes many functions that have small parity-decision-tree-size (such as parity) but which are not expressible as polynomial-size DNF's. While it is also true that DNF is not a restriction of the parity decision tree class, and thus the Kushilevitz/Mansour result did not imply learnability of DNF, it seemed that similar techniques might lead to a polynomial-time learning algorithm for DNF in the same uniform-plus-membership model. In fact, Mansour [Man92] subsequently presented a similar algorithm for learning DNF in time $n^{O(\log\log n \log \epsilon^{-1})}$. However, as we will discuss in more detail in the next chapter, in the same paper he presented some evidence that this approach might not give a polynomial-time learning algorithm for DNF.

In the model of distribution-independent learning with membership queries there have been several nice polynomial-time learning results for strict subclasses of DNF. Blum and Rudich [BR92] gave an algorithm for exactly learning k-DNF for $k = O(\log n)$ using membership and equivalence queries (Bshouty has given simpler alternate algorithms for this problem [Bsh93, Bsh95]). And in a very surprising yet elegant result, Bshouty [Bsh93] proved that Boolean decision trees are likewise learnable with membership and equivalence queries. In fact, he showed more generally that every Boolean function f is learnable with membership queries in time polynomial in the larger of the CNF- and DNF-sizes of f.

3.2.3 Hardness Results

There are, however, known limitations on the power of learning with membership queries. Aizenstein *et al.* showed that if read-thrice DNF (DNF in which each variable occurs at most three times) is learnable with membership and proper equivalence queries then $NP =$ co-NP. More generally, Angluin and Kharitonov have shown that if one-way functions exist and if DNF is PAC learnable with membership queries then, with respect to the class of distributions computable by polynomial-time circuits, DNF is PAC learnable *without* membership queries. In essence, this result says (assuming a fundamental cryptographic assumption, *i.e.*, that one-way functions exist) that membership queries will not help with PAC learning DNF.

This Angluin and Kharitonov result, while theoretically interesting, did not close the door on the possibility that membership queries *might* help for learning DNF with respect to restricted classes of distributions, such as with respect to uniform. To address this possibility, Kharitonov pursued a line of research [Kha92, Kha93] that culminated in showing that, given a specific assumption about the hardness of factoring, AC^0 is not learnable in polynomial time, even weakly, with respect to uniform, and using membership queries. This means that for the representation class of polynomial-size circuits composed of AND's and OR's of some fixed constant depth, it is apparently impossible to learn in even a very weak sense. Furthermore, Kharitonov hoped to show that the "fixed constant depth" was in fact depth two, that is, that polynomial-size DNF was similarly hard to learn.

3.2.4 Summary and New Results

In summary, while there has been sustained interest in the learnability of DNF in various models, most of the positive results have been for restricted versions of DNF learnability or have not been polynomial time. In fact, there were no known algorithms for efficiently learning unrestricted DNF in any generally-accepted model. Furthermore, there were negative results for learning without membership queries in certain models [Ang90, BFJ$^+$94], and progress was apparently being made toward proving that DNF was not learnable with membership queries even in a weak sense of learning and with respect to uniform.

It is at this point that the primary theoretical results of this thesis enter the picture. In joint work [BFJ$^+$94], we showed that DNF is indeed learnable weakly with respect to uniform using membership queries. This was then strengthened [Jac94] to show that DNF is strongly learnable in the same model by the Harmonic Sieve. The results depend heavily on Fourier techniques, and the strong learning algorithm also uses a hypothesis boosting mechanism. We discuss our results in detail in later chapters, but first we review the Fourier transform and hypothesis boosting tools that will be used to obtain these results.

Chapter 4

The Fourier Transform

In this chapter we present what has proved to be a particularly useful tool in learning theory, a multidimensional discrete Fourier transform known as the Walsh transform. We begin by defining the transform and some of its basic properties. We then outline several important learning results obtained using analysis based on this transform. Finally, we focus on a particular result that will play a key role in the Harmonic Sieve.

4.1 Definition and Properties of the Fourier Transform

The Discrete Fourier Transform (DFT) is perhaps best known for its applications in signal processing. However, a fast algorithm for the DFT (the Fast Fourier Transform, or FFT) also plays a central role in efficient algorithms for tasks such as multiplication of integers and of polynomials, and is therefore well-known to computer scientists. The DFT can be defined as follows[1]: given a vector $f = [f_0, f_1, \ldots, f_{N-1}]$ of N complex numbers, the DFT of f is a vector $\hat{f} = [\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_{N-1}]$ of N complex numbers defined by

$$\hat{f}_i = \frac{1}{N} \sum_{k=0}^{N-1} f_k \cdot \omega_N^{ik}$$

[1] There are slight differences between our definition and one commonly used in computer science [AHU74].

for $0 \leq i < N$, where $\omega_N = e^{2\pi\sqrt{-1}/N}$. A useful property of the DFT is that it is essentially its own inverse; that is, again for $0 \leq i < N$,

$$f_i = \sum_{k=0}^{N-1} \hat{f}_k \cdot \omega_N^{-ik}.$$

We will be interested in applying the DFT to functions, and in particular to functions defined over hypercubes. For any integer b, let $[b]$ represent the set $\{0, \ldots, b-1\}$ and consider a function $f : [b]^n \to \mathbf{C}$. We can view such a function as a vector of b^n complex numbers, one value in the vector for each value in the truth table of f. Thus we could define the DFT for such a function by simply taking the DFT described above over the vector representation of the function.

However, as we will see, there is some advantage to defining the DFT in a way that takes into account the underlying structure of the domain of the function to be transformed. In particular, we will define the *multidimensional discrete Fourier transform* (also known as the *Walsh transform*) as follows: given a function $f : [b]^n \to \mathbf{C}$, the multidimensional discrete Fourier transform (or just Fourier transform) of f is defined for each $a \in [b]^n$ by

$$\hat{f}(a) = \frac{1}{b^n} \sum_{x \in [b]^n} f(x) \cdot \omega_b^{\sum_{i=1}^n a_i x_i}.$$

Using the notation

$$\chi_a(x) = \omega_b^{\sum_i a_i x_i}$$

note that we can express $\hat{f}(a)$ as

$$\hat{f}(a) = \mathbf{E}_{x \in_U [b]^n}[f(x) \cdot \chi_a(x)],$$

where $\mathbf{E}_{x \in_U S}[\cdot]$ represents the expectation over x drawn uniformly at random from the set S (we will often write this simply as $\mathbf{E}[\cdot]$ when the set S is understood from context). We call $\hat{f}(a)$ a *Fourier coefficient* of f. The inverse (multidimensional discrete) Fourier transform is given by

$$f(x) = \sum_{a \in [b]^n} \hat{f}(a) \cdot \chi_a^*(x)$$

where χ_a^* is the complex conjugate of χ_a, that is,

$$\chi_a^*(x) = \omega_b^{-\sum_i a_i x_i}.$$

We will later use several properties of the Fourier transform and especially of the functions χ_a. It can be shown that the χ functions have the property that

$$\mathbf{E}[\chi_a \cdot \chi_b^*] = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise.} \end{cases}$$

Thus with respect to the inner product $\langle f, g \rangle = \mathbf{E}[f \cdot g^*]$ these functions form an orthonormal basis for the space of all complex-valued functions on $[b]^n$. Also note that $\chi_{\bar{0}}$ is the constant $+1$ function; therefore, $\hat{f}(\bar{0}) = \mathbf{E}[f \cdot \chi_{\bar{0}}] = \mathbf{E}[f]$. By Parseval's theorem, for every function $f : [b]^n \to \mathbf{R}$,

$$\mathbf{E}[f^2] = \sum_a |\hat{f}(a)|^2.$$

For f mapping to $\{-1, +1\}$ it follows that $\sum_a |\hat{f}(a)|^2 = 1$. Because of this, when we refer to a *Boolean* function we will have in mind one that maps to $\{-1, +1\}$ rather than the usual mapping to $\{0, 1\}$ (we take $+1$ to represent `true` in either case). More generally, it can be shown that for any real-valued functions f and g, $\mathbf{E}[fg] = \sum_a \hat{f}(a)\hat{g}^*(a)$.

For the special case $b = 2$ we prefer to use an equivalent set-based notation in defining the Fourier basis functions and transform. In this case, for each set $A \subseteq \{1, \ldots, n\}$ we can define the function $\chi_A : \{0, 1\}^n \to \{-1, +1\}$ as

$$\chi_A(x) = (-1)^{\sum_{i \in A} x_i} = 1 - 2\left(\sum_{i \in A} x_i \bmod 2\right)$$

(this is consistent with the earlier definition, as $\omega_2 = -1$). That is, $\chi_A(x)$ is the Boolean function that is 1 when the parity of the bits in x indexed by A is even and is -1 otherwise. Also note that for Boolean f, $\hat{f}(A) = \mathbf{E}[f \cdot \chi_A]$ represents the correlation of f and χ_A with respect to the uniform distribution. Specifically, it can be shown that for every $A \subseteq \{1, \ldots, n\}$,

$$\mathbf{Pr}_{x \in_R U}[f(x) = \chi_A(x)] = \frac{1}{2} + \frac{\mathbf{E}[f \cdot \chi_A]}{2}.$$

4.2 Applications to Learning

In this section we review several learning algorithms that have been derived using Fourier techniques, some of which have already been mentioned in the previous chapter. Our emphasis in this section will be on the intuition behind these results rather than on giving mathematically rigorous descriptions of the algorithms. In the next section we will give details of a particular Fourier-based algorithm that will play a key role in our approach to learning DNF.

4.2.1 Learning AC^0 in Quasipolynomial Time

Linial, Mansour, and Nisan introduced the idea of applying Fourier analysis in the study of learnability of representation classes [LMN93]. In particular, they gave an algorithm for learning the representation class AC^0 in quasipolynomial time. AC^0 is a subclass of the class of Boolean functions over the set $\{0,1\}^*$ of finite Boolean strings. For any $f : \{0,1\}^* \to \{-1,+1\}$, define $f_n : \{0,1\}^n \to \{-1,+1\}$ to be the function over $\{0,1\}^n$ consistent with f over this domain. Then f is in AC^0 if and only if there is a polynomial p and a constant d such that for all n, f_n is computable by an $\{\wedge,\vee,\neg\}$-circuit of depth at most d and of size bounded by $p(n)$ (Figure 3.3). For $g : \{0,1\}^n \to \{-1,+1\}$, we say that g is an AC^0 function if there exists $f \in AC^0$ such that $g = f_n$.

AC^0 is an important class from a learning point of view in part because the class of polynomial-size DNF expressions is a subclass of AC^0. It is also a well-studied class in complexity theory. For our purposes, a key complexity result concerning AC^0 is that n-bit parity is not in AC^0 [FSS81, Has86]. In fact, Hastad has shown that even, say, parity over a subset S of the inputs such that $|S| = \sqrt{n}$ cannot be weakly approximated by any function $f \in AC^0$ [Has86]. That is, for all sets $A \subseteq \{1,\ldots,n\}$ such that $|A| \geq \sqrt{n}$ and for every $f \in AC^0$, the Fourier coefficient $\hat{f}(A)$ has superpolynomially small magnitude.

While Hastad's results are actually somewhat stronger than this, Linial *et al.* went even further, showing an overall bound on the sum of squares of a very large set of Fourier coefficients. Although this bound does not follow immediately from Hastad's results cited above, the proof of the bound does build (in a very nontrivial way) on Hastad's well-known Switching Lemma. Before we state the bound, recall that every function $f : \{0,1\}^n \rightarrow \{-1,+1\}$ can be written in terms of its Fourier coefficients $\hat{f}(A)$ as

$$f(x) = \sum_A \hat{f}(A) \cdot \chi_A(x).$$

What Linial *et al.* showed is that for any $f \in AC^0$, for all sets A of size exceeding roughly $k \approx \log^d(n/\epsilon)$, where d is the depth of the circuit computing f, the coefficients $\hat{f}(A)$ of f are *so* small that the function g defined by

$$g(x) = \sum_{|A| \leq k} \hat{f}(A) \cdot \chi_A(x)$$

is very close to f. Specifically, for an appropriate choice of k, $\mathbf{E}_{x \in_R U}[(f(x) - g(x))^2] \leq \epsilon$. They then proved that this implies that $\text{sign}(g)$ is an ϵ-approximator for f with respect to uniform.

While the analysis underlying this approximation result is quite sophisticated, the learning algorithm that results is very simple. Assume that we are able to draw examples of an AC^0 function f uniformly at random over $\{0,1\}^n$ (*i.e.*, we are given an example oracle $EX(f, U)$). Ignoring computational considerations, we can use this oracle to estimate $\hat{f}(A) = \mathbf{E}[f(x)\chi_A(x)]$ for all sets A of size at most polylogarithmic in n/ϵ. By Chernoff bound arguments we need only quasipolynomially many samples to obtain adequate estimates for all of these values. Thus we can estimate all of these coefficients in quasipolynomial time. We then use these estimated coefficients to create an approximation to the function g above and output $\text{sign}(g)$ as our final hypothesis. By the earlier claims this hypothesis will well-approximate f with respect to the uniform distribution. Therefore, AC^0 is PAC learnable in quasipolynomial time with respect to uniform.

This LMN algorithm was an exciting discovery: AC^0 is a much richer class than those for which efficient learning algorithms were known at the time. Furthermore, the Fourier techniques were obviously quite powerful and seemed likely to lead to further algorithms. The major question was, could they lead to *efficient* learning algorithms rather than ones running in quasipolynomial time?

4.2.2 Efficient Fourier Learning

Two different results at about the same time showed that Fourier techniques could indeed give rise to efficient learning algorithms for certain representation classes.

Probabilistic Concepts

Aiello and Mihail [AM91] showed that certain simple probabilistic concepts could be efficiently learned with respect to the uniform distribution using an example oracle. A *probabilistic concept* p can be thought of as a randomized function that operates in two stages. Given an input x, the first stage invokes a deterministic function $p_1(x)$ that produces a value in $[0, 1]$. The second stage treats the value produced by the first stage as a probability and outputs 1 with probability $p_1(x)$ and 0 with probability $1 - p_1(x)$. The learning problem for probabilistic concepts is to take a sample of the $\{0, 1\}$-outputs of a probabilistic concept p and produce a good approximation to the underlying deterministic function p_1. The underlying functions considered by Aiello and Mihail were 1-decision lists (Figure 3.1) and read-once decision trees (functions expressible as a Boolean decision tree (Figure 3.2) in which each variable x_i appears in at most one decision node of the tree).

A key to the Aiello and Mihail results was showing that for any concept p in the two probabilistic classes mentioned above there exists a good approximator to p with respect to uniform that depends on only a *polynomial* (in n and ϵ^{-1}) number of Fourier coefficients. By "good approximator to p" we mean that the expected absolute difference between the approximator h and the underlying deterministic function p_1 is bounded by ϵ. In fact, they showed that a good approximator to the probabilistic concepts they considered is a hypothesis of the form

$$h = \sum_{A \in S} \hat{f}(A)\chi_A,$$

where the set $S \subset 2^{\{1,\ldots,n\}}$ is polynomial size. Furthermore, they showed how to efficiently find the set S, as well as how to find sufficiently accurate approximations to the coefficients indexed by S, given only an example oracle for p.

Decision Trees

Kushilevitz and Mansour took a somewhat different approach to produce an efficient Fourier-based algorithm. They were the first to consider the model of learning that our DNF algorithm is designed for: learning with respect to the uniform distribution using a membership oracle (rather than an example oracle, as in the previous results). While having a membership oracle available makes learning somewhat easier, it is not an unreasonable model of learning. In fact, we will discuss potential applications for algorithms developed in this model in the second part of this thesis.

Given the additional power of membership queries, Kushilevitz and Mansour were able to show that decision trees are learnable with respect to uniform in time polynomial in the size of the tree (*i.e.*, the number of leaves). This was an interesting result for several reasons. First, because of their understandability, decision trees have been the subject of much study in both the theoretical and applied machine learning communities. In fact, one of the more popular current machine learning algorithms, C4.5 [Qui93], is a heuristic algorithm for learning decision trees.

Perhaps even more importantly, the Kushilevitz and Mansour proof technique was novel and powerful. Kushilevitz and Mansour proved a Fourier property about Boolean decision trees, much as Linial *et al.* had done for AC^0, and then showed that representation classes possessing this property were efficiently learnable. In fact, both the AC^0 and decision tree learning algorithms output a hypothesis approximating

$$\operatorname{sign}\left(\sum_{A \in S} \hat{f}(A)\chi_A\right)$$

for an appropriate set $S \subseteq \{1, \ldots, n\}$. What sets the Kushilevitz and Mansour approach apart is that they gave an algorithm for efficiently finding a *polynomial-size* set S such that the above hypothesis is a good approximator. In particular, the algorithm uses membership queries to find the Fourier coefficients $\hat{f}(A)$ with the largest magnitude; the indices of these coefficients define the set S. The algorithm for finding these coefficients plays a key role in the Harmonic Sieve and is discussed in detail in Section 4.3.

Finally, it should be noted that Kushilevitz and Mansour showed that the Fourier property of Boolean decision trees used to show learnability of this class also holds for decision trees with arbitrary parity functions at the internal nodes (parity decision trees); thus the class of parity decision trees is also learnable. Subsequently, Bellare [Bel92] proved that the property holds for several other classes, including k-term-DNF for $k = O(\log n)$.

4.2.3 Learning DNF in Slightly Superpolynomial Time

While the Kushilevitz and Mansour decision tree learning algorithm was a significant step forward, the knotty question of whether a similar algorithm existed for DNF remained open. In particular, there are polynomial-size DNF expressions that are not expressible by any polynomial-size decision tree, even a tree with parity functions at the decision nodes. Could such functions be learned in time polynomial in their size as a DNF?

For a while, the best answer was "almost." As mentioned in the previous chapter, it is relatively straightforward to show that DNF can be learned with respect to uniform in time roughly $n^{O(\log(s/\epsilon))}$, where s is the DNF-size of the target [Ver90]. This result does not require Fourier techniques and does not require membership queries. Subsequently, Mansour [Man92] combined Fourier ideas from Linial *et al.* and Kushilevitz-Mansour to show that polynomial-size DNF is learnable with respect to uniform and using membership queries in time roughly $n^{O(\log \log n \log \epsilon^{-1})}$. Once again, the hypothesis produced was an approximation to a function h of the form

$$h = \operatorname{sign}\left(\sum_{A \in S} \hat{f}(A)\chi_A\right),$$

but now S was (slightly) superpolynomial in size in the worst case.

4.2.4 Weakly Learning DNF

One of the contributions of this thesis provides the next step in our progression of Fourier results in learning theory. This step might appear to be quite small, but as we will see it plays a significant role in leading up to our strong DNF learning algorithm. The result was this: DNF is weakly learnable with respect to the uniform distribution using membership queries

[BFJ+94]. As previously noted, this was the first known algorithm for learning unrestricted DNF in a standard model of learning. The result also provided a new bound on how far learnability hardness results for AC^0 in this same model of learning [Kha93] could be taken. To some extent, however, the main reason for interest in the result was that it renewed hope that DNF might be strongly learnable in the same model, perhaps by building on the weak learner. This hope was well-founded, as we show in Chapter 6.

The proof that DNF is weakly learnable boils down to one simple observation: for every DNF expression f of size s, there is some Fourier coefficient $\hat{f}(A)$ such that $|\hat{f}(A)| \geq 1/3s$. While this is quite simple to prove, we will not bother to do so now, as we prove a more general result in Chapter 6. Given this observation, the technique used by Kushilevitz and Mansour for finding large Fourier coefficients can be employed, given a membership oracle for any DNF expression f, to find a coefficient of f having magnitude at least $1/3s$. Since a coefficient $\hat{f}(A)$ represents the correlation between the parity function χ_A and the target function f, having found a suitably large $\hat{f}(A)$ means that the corresponding parity χ_A is a weak approximator to f. The form of the hypothesis h output by this weak DNF learner is then particularly simple:

$$h = \text{sign}(\hat{f}(A))\chi_A.$$

4.2.5 Summary and Comparison with Harmonic Sieve

Reviewing the Fourier-based learning algorithms presented in this section, we see a common theme. First, a Fourier property of the class of representations to be learned had to be proved. Second, based on this property, an algorithm for finding the "right" Fourier coefficients to describe the target function was constructed. And third, the output hypothesis was (the sign of) a linear combination of the parities corresponding to the right coefficients, where the coefficients of this combination were estimates of the corresponding Fourier coefficients.

While this methodology had produced a string of impressive results, Mansour [Man92] gave some evidence that it might not be sufficient for strongly learning DNF in polynomial time. In particular, for *all* Fourier-based learning algorithms prior to the Harmonic Sieve the "right" set S of Fourier coefficients was defined as a set having a property something like

$$\sum_{A \in S} \hat{f}^2(A) \geq 1 - \epsilon,$$

where f represents the target function. Since for Boolean f, $\sum_{A \subseteq \{1,\ldots,n\}} \hat{f}^2(A) = 1$ (by Parseval), a set S as above represents most of the "spectral power" of f. In turn, it can be shown that for the function $g = \sum_{A \in S} \hat{f}(A)\chi_A$, $\mathbf{E}[(f - g)^2] \leq \epsilon$. Thus g is in some sense very close to f, and g (or its sign) can be used to approximate f. However, Mansour showed that there are DNF expressions for which any polynomial number of the Fourier coefficients represent relatively little of the spectral power of the function. Specifically, there are some DNF expressions such that any set S having the property above must be superpolynomially large in ϵ^{-1}. Thus any algorithm for learning DNF by finding such an S requires superpolynomial time. Therefore, to prove that DNF is efficiently learnable using Fourier techniques would require a very different approach than that taken in previous work.

The Harmonic Sieve provides such a novel approach. Specifically, it finds not only Fourier coefficients of the target function, but also Fourier coefficients of certain functions *other* than the target. Furthermore, the output hypothesis, while similar to the hypotheses of most previous Fourier algorithms in that it is the sign of a sum of parity functions, does not have weights that correspond to Fourier coefficients of the target. Instead, the weights are determined by a very different method based on hypothesis boosting. While at this point it is probably not at all clear that such an approach could produce any sort of learning algorithm, let alone an efficient learning algorithm for DNF, we will soon prove that it does.

We continue this comparison between the Harmonic Sieve and previous Fourier learning algorithms in Chapter 6 after presenting the Harmonic Sieve algorithm in detail. For now, we turn to a discussion of the Kushilevitz/Mansour algorithm for finding the large Fourier coefficients of a function.

4.3 Finding Well-correlated Parity Functions

As noted in the previous section, our DNF learning algorithm relies in part on a technique that, given a membership oracle for a function f, efficiently finds the large Fourier coefficients of f. In other words, this technique can be used to find those parity functions that are well-correlated with f.

In this section we describe the first-known technique for performing this task, which was discovered by Goldreich and Levin [GL89]. Subsequently, Kushilevitz and Mansour [KM93] gave a Fourier-based proof of correctness for this technique and showed how to use it to learn decision trees with respect to the uniform distribution. This technique (which we call the `KM` algorithm) and a proof of correctness will be presented in detail. In addition to the basic `KM` algorithm, we will give a slight extension of `KM` that will be used by the Harmonic Sieve.

We have chosen to detail the `KM` algorithm because it is relatively intuitive and has been used in a variety of learning algorithms [KM93, Bel91, Man92, BFJ$^+$94, Jac94]. It should be noted that an alternative algorithm that may give better performance (at least for many problems) has been discovered more recently by Levin [Lev93]. Finally, for applications in which the number of inputs n is so small that the entire truth table of the function to be learned can be generated in a computationally-feasible amount of time, the Fast Fourier Transform (FFT; see [AHU74] for a definition and discussion) can be used to find the large Fourier coefficients of the target function. That is, we can produce a vector representing the truth table of f, perform a multidimensional version of the FFT on this truth table to get a vector of all of the Fourier coefficients of f, and then exhaustively search this vector for the maximum coefficient. While asymptotically `KM` is polynomial-time in n and this FFT approach is exponential-time, the FFT is more efficient than the sampling-based `KM` on small problems. In fact, we use the FFT in one of the applications reported on in Part II of this thesis.

4.3.1 The KM Algorithm

The basic KM algorithm finds, with probability at least $1-\delta$, close approximations to all of the large Fourier coefficients of a Boolean function f on $\{0,1\}^n$. Since on this domain a Fourier coefficient $\hat{f}(A)$ represents the correlation between a parity χ_A and the target function f, KM can be used to find all parities that correlate well with f. By "large" coefficients we mean coefficients of magnitude exceeding some threshold θ; the algorithm runs in time polynomial in n, $\log(\delta^{-1})$, and θ^{-1}. KM makes membership queries for f, but otherwise f is treated as a black box.

KM is a recursive algorithm that is given as input a membership oracle $MEM(f)$, threshold θ, confidence δ, and the specification of a set of Fourier coefficients. A set of coefficients is specified by two parameters, an integer $k \in [0,n]$ and a set $A \subseteq \{1,\ldots,k\}$; these parameters define the set $C_{A,k} = \{\hat{f}(A \cup B) \mid B \subseteq \{k+1,\ldots,n\}\}$. Initially, KM is run on the set of all Fourier coefficients of f, $C_{\emptyset,0}$.

Each time KM is called, it begins by defining a partition of the input set $C_{A,k}$ into the two equal-sized sets $C_{A,k+1}$ and $C_{A\cup\{k+1\},k+1}$. For notational convenience we let C_i denote either of the sets in the partition. After partitioning the input set, KM next tests to see whether or not a large coefficient can possibly be a member of one or both of the sets. It (conceptually) does this by computing the sum of squares of the Fourier coefficients in each set; we denote the sum for set C_i by $L_2^2(C_i)$. If $L_2^2(C_i) < \theta^2$ for either C_i then certainly none of the coefficients in that C_i exceeds the threshold θ, and the coefficients in that C_i can be ignored in subsequent processing. On the other hand, if $L_2^2(C_i) \geq \theta^2$ and $|C_i| > 1$ then KM recurses on C_i. The above-threshold singleton sets remaining at the end of this process are the desired large coefficients. Because the sum of squares of the Fourier coefficients of a Boolean function is 1 (by Parseval), the algorithm will recurse on at most θ^{-2} subsets at any of the n levels of the recursion. Thus this algorithm runs in time polynomial in the depth n of the recursion, in θ^{-1}, and in the maximum time required to compute $L_2^2(C_i)$ for any C_i.

Of course, the time required to compute $L_2^2(C_i)$ could be exponentially large. The KM algorithm gets around this difficulty by estimating $L_2^2(C_i)$ for each C_i in an ingenious way:

for every function $f : \{0,1\}^n \to \mathbf{R}$, every $k \in [0,n]$, and every $A \subseteq \{1,\ldots,k\}$,

$$L_2^2(C_{A,k}) = \mathbf{E}_{x,y,z}[f(yx)f(zx)\chi_A(y \oplus z)],$$

where the expectation is uniform over $x \in \{0,1\}^{n-k}$, $y \in \{0,1\}^k$, and $z \in \{0,1\}^k$, and yx represents the concatenation of y and x. (For a derivation of a generalization of this equality, see chapter 8.) Thus $L_2^2(C_{A,k})$ can be estimated by using membership queries to sample f appropriately. In particular, by Hoeffding's inequality, `KM` can efficiently estimate a value μ' such that

$$\mathbf{Pr}\left[|\mu' - L_2^2(C_{A,k})| \geq \frac{\theta^2}{4}\right] \leq \frac{\delta\theta^2}{2n}.$$

Therefore, with probability at least $1 - \delta\theta^2/2n$, if $L_2^2(C_{A,k}) \geq \theta^2$ then $\mu' \geq 3\theta^2/4$, and if $L_2^2(C_{A,k}) < \theta^2/2$ then $\mu' < 3\theta^2/4$. With high probability, then, if we recurse only on sets that have $\mu' \geq 3\theta^2/4$ then we will recurse on all the sets that have $L_2^2(C_{A,k}) \geq \theta^2$ and we will not recurse on any set such that $L_2^2(C_{A,k}) < \theta^2/2$. By the earlier argument this will give us a polynomial-time algorithm that with probability at least $1 - \delta$ returns a set $S \subseteq 2^{\{1,\ldots,n\}}$ such that

1. For all A such that $|\hat{f}(A)| \geq \theta$, $A \in S$; and

2. For all $A \in S$, $|\hat{f}(A)| \geq \frac{\theta}{\sqrt{2}}$.

We say that such a set has the *large Fourier coefficient property.*

4.3.2 Extending `KM`

The Harmonic Sieve will need to find the large coefficients of certain functions that are not Boolean valued. This leads us to extend `KM` slightly:

Lemma 4.1 *There is an algorithm* `KM'` *such that, for any function* $g : \{0,1\}^n \to \mathbf{R}$*, threshold* $\theta > 0$*, and confidence* $\delta > 0$*,* `KM'`$(n, MEM(g), \theta, L_\infty(g), \delta)$ *returns, with probability at least* $1 - \delta$*, a set with the large Fourier coefficient property.* `KM'` *uses membership queries and runs in time polynomial in* n*,* θ^{-1}*,* $\log(\delta^{-1})$*, and* $L_\infty(g)$*.*

Proof: For each set $C_{A,k}$, by Hoeffding's inequality (Lemma 2.2) a number of samples polynomial in θ^{-1}, $\log(\delta^{-1})$, and $L_\infty(g)$ (*i.e.*, $\max_x |g(x)|$) is sufficient to estimate μ' such that

$$\mathbf{Pr}\left[|\mu' - L_2^2(C_{A,k})| \geq \frac{\theta^2}{4}\right] \leq \frac{\delta\theta^2}{2nL_\infty^2(g)}.$$

The fact that by Parseval $\sum_A \hat{g}^2(A) = \mathbf{E}[g^2] \leq L_\infty^2(g)$ means that the number of recursive calls at each of the n levels of the recursion is, with probability at least $1-\delta$, bounded above by $2L_\infty^2(g)/\theta^2$. The rest of the argument is analogous to that for `KM`. $\square$

Chapter 5

Hypothesis Boosting

As mentioned in the last chapter, in Blum *et al.* we showed that DNF is weakly learnable with respect to the uniform distribution using membership queries [BFJ+94]. This is an interesting result from a theoretical point of view, since it is also known that AC^0 is *not* learnable in this sense modulo cryptographic assumptions [Kha93]. It will also form the basis of our main theoretical result, that DNF is *strongly* learnable with respect to uniform using membership queries. A key tool we will use to get the strong result from the weak one is hypothesis boosting, which we now discuss.

5.1 The Boosting Concept

Schapire [Sch90] first discovered the surprising fact that every representation class that can be weakly learned (in a distribution-independent model) can be strongly learned. Subsequently, several improved boosting algorithms have been developed [Fre90, Fre92, Fre93, FS95]. In addition to certain efficiency advantages, the subsequent boosters tend to be much simpler than Schapire's original algorithm. As we will see, this simplicity will be exploited in a novel way by our Harmonic Sieve algorithm.

All of the currently-known boosting algorithms are similar in certain respects. In each case, we assume that the boosting algorithm is given a weak learner; in fact, we typically assume that the weak learner is distribution-independent and produces $(\frac{1}{2}-\gamma)$-approximators for the target representation class, where γ is known by the boosting algorithm. All of the boosters run the given weak learner multiple times and combine the resulting weak hypotheses in some manner to produce a strong hypothesis. Boosting occurs as a result of running the weak learner on different (simulated) example oracles $EX(f, D_i)$ each time, thus producing weak hypotheses that perform well on different regions of the instance space.

The boosters differ in how they define the particular simulated distributions D_i against which the weak learner is run and in how they combine the weak hypotheses generated into the final strong hypothesis. The DNF learning algorithm presented in this paper is based on one of Freund's boosting algorithms [Fre90]; we will call this algorithm `F1`. An important feature of `F1` for our purposes is that all of the distributions D_i simulated by `F1` when boosting with respect to uniform are polynomial-time computable functions, *i.e.*, the weight assigned to x by D_i can be efficiently computed for all x. This means that in addition to supplying the weak learner with example oracle $EX(f, D_i)$ at each boosting step, we can also give the learner an oracle for the distribution D_i. This ability to provide the weak learner with a such a distribution oracle is crucial, as our weak algorithm, unlike other learning algorithms that we are familiar with, requires such an oracle.

Another noteworthy aspect of Freund's `F1` algorithm is that the final hypothesis is formed in a simple way: apply the majority function MAJ to the weak hypotheses produced during the boosting process. As the weak hypotheses generated by the Harmonic Sieve are relatively simple (they are parity functions), our final hypothesis is also quite simple. This may have implications for potential practical applications of the Harmonic Sieve and related algorithms, as discussed in Part II of the thesis.

We now discuss the `F1` boosting algorithm in detail.

5.2 Freund's `F1` Boosting Algorithm

As input, `F1` is given positive ϵ, δ, and γ, a weak learner `WL` that produces $(\frac{1}{2}-\gamma)$-approximate hypotheses for functions in $\mathcal{F}$, and an example oracle $EX(f, D)$ for some $f \in \mathcal{F}$. The weak learner `WL` is assumed to take an example oracle and confidence parameter δ' as inputs and produce a $(\frac{1}{2} - \gamma)$-approximate hypothesis with probability at least $1 - \delta'$.

Given these inputs, the F1 algorithm steps sequentially through k stages (an appropriate value for k is given below). At each stage i, $0 \leq i \leq k-1$, F1 performs one run of WL, which produces (with high probability) a $(\frac{1}{2}-\gamma)$-approximate hypothesis w_i. During the first stage, F1 runs WL on the given example oracle $EX(f,D)$. During each of the succeeding stages $i>0$, F1 runs WL on a simulated example oracle $EX(f,D_i)$, where distribution D_i focuses weight on those instances x such that slightly fewer than half of the i weak hypotheses generated during previous stages are correct on x and slightly more than half are incorrect. Recalling that the final hypothesis of F1 is a majority vote over the weak hypotheses it produces, this choice of distribution makes some sense: the distribution focuses the weak learner on those instances that would be misclassified if a majority vote of the current weak hypotheses was taken, but which are also close to receiving a correct vote. It is clearly a good idea to focus the weak learner at the next stage on such instances.

On the other hand, this choice of distribution may appear to have certain problems. Specifically, such a D_i puts very little weight on instances that are very wrong (*i.e.*, instances x such that almost all of the previous hypotheses label x incorrectly), and thus these instances will very likely not be correctly labeled by the final hypothesis. While this is true, the distributions D_i begin (for small i) rather "flat" with respect to variation from the target distribution D and become more focused gradually as i increases. Thus it is not until i is fairly large that the booster begins to effectively ignore the very wrong instances, and Freund shows that for the specific D_i's defined below there are in fact very few of these ignored instances at any stage of boosting. Thus the fact that these instances will likely be wrong in the final hypothesis is not a concern.

Before defining the distributions D_i precisely and explaining how the algorithm simulates $EX(f,D_i)$, we need some notation. Let

$$\tilde{\alpha}_r^i = \begin{cases} B(\lfloor \frac{k}{2} \rfloor - r; k-i-1, \frac{1}{2}+\gamma) & \text{if } i-\frac{k}{2} < r \leq \frac{k}{2} \\ 0 & \text{otherwise} \end{cases}$$

where $B(j;n,p) = \binom{n}{j} p^j (1-p)^{n-j}$ is the binomial formula. Also, let $\alpha_r^i = \tilde{\alpha}_r^i / \max_r\{\tilde{\alpha}_r^i\}$. Finally, let $r_i(x)$ represent the number of weak hypotheses w_j among those hypotheses produced before stage i that are "right" on x, *i.e.*, $r_i(x) = |\{0 \leq j < i \mid w_j(x) = f(x)\}|$.

Now we are ready to describe the simulation of $EX(f, D_i)$. During stage $i > 0$, when the weak learner requests an example from the simulated example oracle $EX(f, D_i)$, F1 queries the example oracle $EX(f, D)$ and receives an example $\langle x, f(x)\rangle$. With probability $\alpha^i_{r_i(x)}$, F1 accepts this example. If F1 does not accept then it queries for another example, repeating this process until it does accept. Finally, F1 passes the accepted example to WL. Thus the distribution D_i simulated by F1 at stage i is

$$D_i(x) = \frac{D(x)\alpha^i_{r_i(x)}}{\sum_y D(y)\alpha^i_{r_i(y)}}. \tag{5.1}$$

Stage i is completed when WL outputs its hypothesis w_i.

It can be shown that if the number of stages k is chosen to be at least $\frac{1}{2}\gamma^{-2}\ln(\epsilon^{-1})$ and the weak learner successfully produces a $(\frac{1}{2} - \gamma)$-approximator at each stage, then F1 will produce an ϵ-approximator as its final hypothesis. However, there is a potential computational difficulty with this approach to boosting. Notice that in general it may require many samples of the example oracle $EX(f, D)$ to simulate the example oracle $EX(f, D_i)$. In fact, it can be seen that the denominator of (5.1) is the probability that F1 accepts an example while simulating the oracle $EX(f, D_i)$. Thus if at some stage i this denominator, $\sum_y D(y)\alpha^i_{r_i(y)}$, becomes quite small then to simulate the example oracle $EX(f, D_i)$ could require a very long (perhaps superpolynomial) time.

To account for this possibility, the algorithm estimates $\sum_y D(y)\alpha^i_{r_i(y)}$ at each stage i. If the estimate is below a threshold then F1 terminates, outputting as its hypothesis h a majority vote over the weak hypotheses discovered before stage i. Intuitively, this is a reasonable stopping condition for the algorithm for the following reasons. As explained earlier, at every stage i very few instances are "very" wrong, *i.e.*, are incorrectly classified by well over half of the weak hypotheses. Also, if the probability of acceptance while simulating $EX(f, D_i)$ is very small then there must be few instances in the region focused on by D_i, that is, few instances on which slightly more than half of the weak hypotheses vote incorrectly. Putting these facts together means that few instances are incorrectly classified by h as defined above.

An implementation of F1 that incorporates these ideas and takes into account several

Invocation: $h \leftarrow \texttt{F1}(EX(f,D), \gamma, \texttt{WL}, \epsilon, \delta)$
Input: Example oracle $EX(f,D)$ for target f and distribution D; $0 < \gamma \leq \frac{1}{2}$; $(\frac{1}{2} - \gamma)$-approximate weak learner $\texttt{WL}(EX, \delta)$ which succeeds with probability at least $1-\delta$; $\epsilon > 0$; $\delta > 0$
Output: h such that, with probability at least $1-\delta$, $\mathbf{Pr}_D[f = h] \geq 1 - \epsilon$

$\bar{\gamma} \equiv \min(\gamma, 0.39)$
$k \leftarrow \frac{1}{2}\bar{\gamma}^{-2}\ln(4/\epsilon)$
$w_0 \leftarrow \texttt{WL}(EX(f,D), \delta/2k)$
for $i \leftarrow 1, \ldots, k-1$ **do**
  $r_i(x) \equiv |\{0 \leq j < i \mid w_j(x) = f(x)\}|$
  $B(j;n,p) \equiv \binom{n}{j} p^j (1-p)^{n-j}$
  $\tilde{\alpha}_r^i \equiv B(\lfloor k/2 \rfloor - r; k-i-1, 1/2+\bar{\gamma})$ if $i - k/2 < r \leq k/2$, $\tilde{\alpha}_r^i \equiv 0$ otherwise
  $\alpha_r^i \equiv \tilde{\alpha}_r^i / \max_{r=0,\ldots,i-1}\{\tilde{\alpha}_r^i\}$.
  $\Theta \equiv \epsilon^3/57$
  $X \equiv$ draw example $\langle x, f(x)\rangle$ from $EX(f,D)$ and compute $\alpha_{r_i(x)}^i$
  $E_\alpha \leftarrow \texttt{AMEAN}(X, b-a=1, \frac{1}{3}\Theta, \delta/2k)$
  if $E_\alpha \leq \frac{2}{3}\Theta$ **then**
    $k \leftarrow i$
    break do
  endif
  $D_i(x) \equiv \frac{D(x)\alpha_{r_i(x)}^i}{\sum_y D(y)\alpha_{r_i(y)}^i}$
  $w_i \leftarrow \texttt{WL}(EX(f,D_i), \delta/2k)$
enddo
$h(x) \equiv MAJ(w_0(x), w_1(x), \ldots, w_{k-1}(x))$
return h

Figure 5.1: The F1 hypothesis boosting algorithm. AMEAN is described in Corollary 2.1. X is a random variable used to estimate $\mathbf{E}_D[\alpha_{r_i(x)}^i]$.

smaller issues is given in Figure 5.1. The following lemma concerning the functionality and running time of `F1` is due to Freund; the previously unpublished proof of one part of Freund's argument is presented in the chapter appendix.

Lemma 5.1 (Freund) *Algorithm* `F1`*, given positive* ϵ*,* δ*, and* γ*, a* $(\frac{1}{2}-\gamma)$*-approximate PAC learner for representation class* $\mathcal{F}$*, and example oracle* $EX(f, D)$ *for some* $f \in \mathcal{F}$ *and any distribution* D*, runs in time polynomial in* n*,* s*,* γ^{-1}*,* ϵ^{-1}*, and* $\log(\delta^{-1})$ *and produces, with probability at least* $1-\delta$*, an* ϵ*-approximation for* f *with respect to* D*.*

`F1` can also be used to boost a weak membership-query learner into a strong membership-query learner. The only change to `F1` is that it accepts the membership oracle $MEM(f)$ as an argument and includes this oracle as an argument in the calls to `WL`. The Harmonic Sieve will be based on the membership-query version of `F1`.

5.3 Applying Boosting to DNF Learning

Hypothesis boosting as described above is designed to take a distribution-independent weak learner and use it to create a similarly distribution-independent strong learner. So if we had a distribution-independent weak PAC learning algorithm for DNF, we would immediately have a strong PAC learning algorithm for DNF. However, it is generally believed (and this author shares this view) that DNF is *not* distribution-independent PAC learnable. If this belief is correct then the existence of hypothesis boosting gives that DNF is also not weakly distribution-independent PAC learnable.

On the other hand, we would like to use the `F1` boosting algorithm to produce a strong distribution-dependent learning algorithm for DNF. Also, `F1` requires a weak learning algorithm that learns with respect to many different distributions. Thus we are led to search for a weak learning algorithm that, while not completely distribution-independent, learns with respect to a wide variety of distributions. In the next chapter we present precisely such a weak learner. To the best of our knowledge, this is the first distribution-dependent weak learner that can be boosted using an off-the-shelf boosting technique.

5.4 Alternative Boosting Algorithms

It should be noted that `F1` is not the only boosting algorithm that could be applied to the DNF learning problem. In particular, Freund has also developed a similar and more efficient (in the distribution-independent setting) boosting algorithm [Fre92, Fre93] that can also be used to learn DNF. We have chosen to detail `F1` for two reasons. First, the later algorithm (call it `F2`) is slightly more involved in that it may choose not to call the weak learner at all at some stages of the boosting process; therefore, clarity of exposition favors `F1`. Second, `F2` produces distributions D_i which may deviate from uniform more than the distributions produced by `F1`. In turn, this deviation from uniform plays a role in the time bound on the Harmonic Sieve, as we will see. Therefore, the `F2` algorithm does not offer a clear performance advantage over `F1` in the context of the Harmonic Sieve. However, if efficient implementation of the Harmonic Sieve for a particular application is an issue, both boosting algorithms should be considered. As mentioned earlier, Schapire's original boosting algorithm [Sch90] is probably not an appropriate basis for `HS`, as it is not clear how to efficiently simulate oracles for the distributions defined by this algorithm.

Recently, Freund and Schapire have developed yet another boosting algorithm, `AdaBoost` [FS95]. `AdaBoost` is designed specifically for "boosting-by-sampling" as opposed to the "boosting-by-filtering" technique employed by `F1`. A boosting-by-filtering algorithm defines each probability distribution D_i by filtering examples received from the oracle $EX(f, D)$. On the other hand, in boosting-by-sampling a large set S of examples is drawn once at the beginning of learning, and regardless of the true target distribution D, a new target distribution D' is defined. This distribution is uniform over the set S and has zero weight on all instances not represented in S. Furthermore, the distributions D_i defined in the course of boosting-by-sampling are also nonzero only on the instances in S. The idea of boosting-by-sampling is to use boosting to find a hypothesis (nearly) consistent with the set S. For large enough S, uniform-convergence arguments can then be used to show that the resulting hypothesis is with high probability an ϵ-approximator to the target function [VC71, Hau88].

In practice, the notion of finding a hypothesis consistent with a given data set plays an important role in many learning algorithms. Thus from an applied point of view, it could be useful to base the Harmonic Sieve on a boosting-by-sampling algorithm. In fact, any boosting-by-filtering algorithm can be converted to a boosting-by-sampling algorithm (the basic idea is to simulate an example oracle $EX(f, D')$). However, `AdaBoost`, which is designed with boosting-by-sampling in mind, has potential practical advantages over earlier boosting algorithms. Primarily, `AdaBoost` is *adaptive* (this is what the "Ada" in `AdaBoost` stands for). Specifically, the distributions D_i created by `AdaBoost` depend on (adapt to) how well the weak hypotheses produced before stage i approximate the function f. Furthermore, `AdaBoost` uses a measure of how well each weak hypothesis approximates the target to assign weight to that weak hypothesis in the final strong hypothesis. As a result, we expect `AdaBoost` to require fewer boosting stages in general than would be required by any of the earlier boosting algorithms.

Therefore, in Part II of the thesis, where we consider applying versions of the Harmonic Sieve to several benchmark problems, we will base our empirical algorithms on `AdaBoost`. However, in the current part of the thesis we will utilize `F1`, as we do not know how to obtain positive DNF-learning theoretical results using a boosting-by-sampling algorithm. The key problem is that the distribution D' described above is typically extremely nonuniform; as we will see, the running time of `HS` depends directly on the nonuniformity of the target distribution.

5.5 Appendix

Here we prove the validity of the `F1` boosting algorithm. In fact, it follows immediately from Freund's work [Fre90] that if the algorithm given in Figure 5.1 runs for all k stages, then the hypothesis produced will, with probability at least $1 - \delta$, be an ϵ-approximator to the target f with respect to distribution D. This fact together with the following lemma gives the proof of Lemma 5.1. The proof of the following lemma is based on unpublished work by Freund [Fre].

Lemma 5.2 *If at any stage i of* `F1`

$$\sum_y D(y)\alpha^i_{r_i(y)} \leq \frac{\epsilon^3}{57}$$

then the hypothesis h formed by taking a majority vote over the weak hypotheses $w_0, w_1, \ldots, w_{i-1}$ is an ϵ-approximator for f with respect to D.

Before proving this lemma, we note that the 0.39 bound on $\bar{\gamma}$ in `F1` was chosen to keep the statement of the above stopping criterion relatively simple. As will become apparent in the proof, other choices of the bound can be used to derive alternative stopping criteria for `F1` having exponents of ϵ arbitrarily close to 2 and smaller constants.

Proof of Lemma 5.2: For $0 \leq i < k$ and $0 \leq r < i$, define

$$D(r) = \sum_{x.r_i(x)=r} D(x)$$

(recall that $r_i(x)$ counts the number of weak hypotheses found before stage i that are "right" on x). Our goal will be to show that at every stage i of `F1`,

$$\sum_y D(y)\alpha^i_{r_i(y)} \leq \frac{\epsilon^3}{57} \Rightarrow \sum_{r \leq i/2} D(r) \leq \epsilon. \tag{5.2}$$

Given this, a majority vote over the weak hypotheses found prior to stage i is an ϵ-approximator to the target function f.

The argument for (5.2) consists of two parts. First, we show that it is always the case that at any stage i,

$$\sum_{r=0}^{\lfloor i/2-(k-i)\bar{\gamma}\rfloor} D(r) \leq \epsilon/2. \tag{5.3}$$

Then we will show that given the antecedent of (5.2) it is also true that

$$\sum_{r=\lfloor i/2-(k-i)\bar{\gamma}\rfloor+1}^{\lfloor i/2\rfloor} D(r) \leq \epsilon/2. \tag{5.4}$$

To show (5.3) we need a fundamental result from Freund. Define for each stage i of the `F1` boosting process the function β^i_r, $0 \leq r \leq i$, as

$$\beta^i_r = \begin{cases} 0 & \text{if } r > \frac{k}{2} \\ \sum_{j=0}^{\lfloor \frac{k}{2}\rfloor - r} B(j; k-i, \frac{1}{2}+\bar{\gamma}) & \text{if } i - \lceil \frac{k}{2} \rceil \leq r \leq \frac{k}{2} \\ 1 & \text{otherwise} \end{cases}$$

(recall that $B(\cdot;\cdot,\cdot)$ represents the binomial formula). Then Freund [Fre90, Fre93] shows that if `F1` is run for $k \geq \frac{1}{2}\gamma^{-2}\ln(4/\epsilon)$ stages on a $(\frac{1}{2}-\gamma)$-approximate weak learner, at each stage i we will have

$$\sum_{r=0}^{i} \beta_r^i D_i(r) \leq \frac{\epsilon}{4}.$$

Note also that if $\bar{\gamma} \leq \gamma$ and we take $k = \frac{1}{2}\bar{\gamma}^{-2}\ln(4/\epsilon)$ then $k \geq \frac{1}{2}\gamma^{-2}\ln(4/\epsilon)$.

Now the median of a binomial distribution $B(n,p)$ is either $\lfloor np \rfloor$ or $\lceil np \rceil$ [JS68]. Because β_r^i represents a cumulative binomial distribution, where the underlying binomial distribution has mean $\mu = (k-i)(\frac{1}{2}+\bar{\gamma})$, it follows that for r such that $\lfloor k/2 \rfloor - r \geq \mu$, $\beta_r^i \geq \frac{1}{2}$. In particular, for $r \leq \lfloor \frac{i}{2} - (k-i)\bar{\gamma} \rfloor$, β_r^i is thus lower bounded, which gives us

$$\frac{\epsilon}{4} \geq \sum_{r=0}^{\lfloor i/2-(k-i)\bar{\gamma}\rfloor} \beta_r^i D_i(r) \geq \frac{1}{2} \sum_{r=0}^{\lfloor i/2-(k-i)\bar{\gamma}\rfloor} D_i(r).$$

The argument for (5.4) is somewhat more involved. First, since the binomial distribution $B(n,p)$ reaches a maximum at $\lfloor p(n+1) \rfloor$ (see, *e.g.*, [Bol85] p. 6), we have that α_r^i is maximized for some $r \leq \lfloor i/2 - (k-i)\bar{\gamma} \rfloor + 1$. For larger values of r, α_r^i is monotonically decreasing. Therefore, in the region $\lfloor i/2 - (k-i)\bar{\gamma} \rfloor < r \leq i/2$, α_r^i reaches a minimum α_{min} at $r = \lfloor \frac{i}{2} \rfloor$. We will show that $\alpha_{min} \geq \epsilon^2/28.5$, and therefore that

$$\begin{aligned}
\sum_{r=\lfloor i/2-(k-i)\bar{\gamma}\rfloor+1}^{\lfloor i/2 \rfloor} D(r) &\leq \sum_{y.i/2-(k-i)\bar{\gamma}<r(y)\leq i/2} D(y)\frac{\alpha_{r_i(y)}^i}{\alpha_{min}} \\
&\leq \frac{\epsilon^3}{57\alpha_{min}} \\
&\leq \frac{\epsilon}{2}
\end{aligned}$$

as desired.

The lower bound argument for $\alpha_{min} = \alpha_{\lfloor i/2 \rfloor}^i$ also consists of two parts. In particular, recall that $\alpha_r^i = \tilde{\alpha}_r^i / \max_r\{\tilde{\alpha}_r^i\}$. Freund has shown [Fre93, Lemma 2.3.10] that the denominator of this expression is upper bounded by $\sqrt{8/(3\pi(k-i-1))}e^{1/12}$. Thus we only need an appropriate lower bound on $\tilde{\alpha}_{min} = \tilde{\alpha}_{\lfloor i/2 \rfloor}^i$ to complete the proof.

To show the required bound on $\tilde{\alpha}_{min}$ we first consider several cases. If k is odd and i is even then

$$\tilde{\alpha}_{min} = \binom{k-i-1}{\frac{k-i-1}{2}} \left(\frac{1}{2}+\bar{\gamma}\right)^{\frac{k-i-1}{2}} \left(\frac{1}{2}-\bar{\gamma}\right)^{\frac{k-i-1}{2}}. \tag{5.5}$$

We use $\tilde{\alpha}_{1/2}$ to denote the right-hand side of (5.5). If k is even and i is odd we have

$$\tilde{\alpha}_{min} = \tilde{\alpha}^i_{\lfloor i/2 \rfloor} \geq \tilde{\alpha}^i_{\lceil i/2 \rceil} = \tilde{\alpha}_{1/2}.$$

For the other two cases (k and i both even or both odd), we have

$$\tilde{\alpha}_{min} = \tilde{\alpha}^i_{\lfloor i/2 \rfloor} \geq \tilde{\alpha}^i_{\lfloor i/2 \rfloor + 1/2} = \tilde{\alpha}_{1/2}, \tag{5.6}$$

where we have generalized the definition of the binomial coefficient to the reals using the $\Gamma(\cdot)$ function.

To lower bound $\tilde{\alpha}_{1/2}$, we first note that using Stirling's approximation it can be shown (see, *e.g.*, [Fre93, Lemma 2.3.10]) that for any real $x \geq 1$,

$$\binom{x}{\frac{x}{2}} > \sqrt{\frac{2}{\pi x}} 2^x e^{-1/3}.$$

Also,

$$\left(\frac{1}{2}+\bar{\gamma}\right)^{x/2} \left(\frac{1}{2}-\bar{\gamma}\right)^{x/2} = \left(\frac{1}{4}-\bar{\gamma}^2\right)^{x/2} = 2^{-x}\left(1-4\bar{\gamma}^2\right)^{x/2}.$$

It can also be shown that for $0 < y < 1$, $(1-y)^{\frac{1}{y}-1} \geq e^{-1}$. Furthermore, for $\bar{\gamma} \leq 0.39$, $(1-4\bar{\gamma}^2) \geq e^{-1}$. Therefore, for $x = k-i-1 \leq k = \frac{1}{2}\bar{\gamma}^{-2}\ln(4/\epsilon)$, we have

$$\left(\frac{1}{2}+\bar{\gamma}\right)^{\frac{k-i-1}{2}} \left(\frac{1}{2}-\bar{\gamma}\right)^{\frac{k-i-1}{2}} \geq 2^{-(k-i+3)}\epsilon^2$$

which means that

$$\tilde{\alpha}_{min} \geq \sqrt{\frac{2}{\pi(k-i-1)}} 2^{k-i-1} e^{-1/3} 2^{-(k-i+3)} \epsilon^2,$$

and combining this with the earlier upper bound on $\max_r\{\tilde{\alpha}^i_r\}$ gives

$$\alpha_{min} \geq \frac{\sqrt{3}}{32} e^{-5/12}\epsilon^2 \geq \frac{\epsilon^2}{28.5}.$$

$\square$

Chapter 6

Learning DNF

In this chapter we prove our main result, that DNF is learnable with respect to the uniform distribution using membership queries. We begin by extending our weak DNF learning result described in Section 4.2.4. Recall that the key to the weak result is that for every DNF f there is a parity function that weakly approximates f with respect to the uniform distribution [BFJ+94]. Below we show that for every DNF f *and* for every distribution D there is a parity function that weakly approximates f with respect to D. Next we show how to exploit this fact to produce an algorithm for weakly learning DNF with respect to certain nonuniform distributions. It is then shown that this weak learner can be boosted into a strong learner for DNF with respect to the uniform distribution. We close the chapter by expanding on the comparison between the DNF algorithm and earlier Fourier-based algorithms begun in Chapter 4.

6.1 A Key Fact about DNF

We now prove that for every DNF f and every probability distribution D over the instance space of f there is a parity that weakly approximates f with respect to D. Before presenting the proof we give some of the intuition behind this result.

First, note that if $\mathbf{E}_D[f(x)]$ deviates noticeably (*i.e.*, inverse polynomially in the size of f) from 0 then the constant parity $\chi_\emptyset$ is a weak approximator to f with respect to D (recall that we are taking Boolean functions as mapping to $\{-1, +1\}$). The harder case, then, is when D is such that f is unbiased, or nearly so. In this case one of the s terms of f must correlate noticeably well with f. This is because all terms agree with f on x's such that $f(x) = -1$, since all terms must be unsatisfied when the function is unsatisfied.[1] And at least one term (call it T) must be satisfied by a $1/s$ fraction of the inputs that satisfy the function.

Now a term is a Boolean function and therefore can be expressed as a linear combination of parity functions by applying the Fourier transform. In fact, a term has a very simple Fourier representation. Call χ_A a *parity in term* T (denoted $\chi_A \in T$) if A is a (possibly empty) subset of the variables in T, and let $|T|$ represent the number of variables in term T. Then the Fourier representation of the $\{-1,+1\}$-valued function represented by a term T is

$$-1 + 2 \cdot \left(\frac{1}{2^{|T|}} \sum_{\chi_A \in T} \pm \chi_A \right),$$

where the sense of a parity (negated or not) in the sum depends on the senses of the variables in the term. This implies that

$$|\mathbf{E}_{x \in_R D}[f(x) \cdot T(x)]| \leq |\mathbf{E}_D[f(x)]| + 2\mathbf{E}_{\chi_A \in T}\left[|\mathbf{E}_D[f(x)\chi_A(x)]|\right],$$

where $\mathbf{E}_{\chi_A \in T}[\cdot]$ represents the expected value over uniform random choice of $\chi_A \in T$. Because we are considering the case in which $\mathbf{E}_D[f]$ is very small but $\mathbf{E}_D[f \cdot T]$ is noticeably large, the above relation implies that the "average" of the parities in T is noticeably well-correlated with f, and therefore at least one of these parities must be well-correlated.

Our original proof formalized these ideas. Subsequently, Bshouty [Bsh] refined our approach and developed the proof that we present here, which gives somewhat better constants.

Fact 6.1 *For every DNF f with s terms and for every distribution D on the instance space of f there exist a term T in f and a $\chi_A \in T$ such that $|\mathbf{E}_D[f\chi_A]| \geq 1/(2s+1)$.*

Proof: There is at least one term T in f such that $\mathbf{Pr}_D[x \text{ satisfies } T] \geq \mathbf{Pr}_D[x \text{ satisfies } f]/s$. Let $T(x)$ be the Boolean function represented by T, *i.e.*, $T(x) = 1$ when x satisfies T and

[1] We take -1 to represent `false` and $+1$ to represent `true`, although the opposite convention is often used in Fourier work.

$T(x) = -1$ otherwise. Also, assume without loss of generality that none of the literals in T are negated and let V represent the set of variables appearing in T. Then for all x,

$$\frac{T(x)+1}{2} = \prod_{v \in V} \frac{1-\chi_{\{v\}}(x)}{2} = \mathbf{E}_{A \subseteq V}[(-1)^{|A|}\chi_A(x)]$$

where the expectation is uniform over the subsets $A \subseteq V$. Let $T'(x) = (T(x)+1)/2$. Then $\mathbf{E}_D[f \cdot T'] = \mathbf{E}_{A \subseteq V}[(-1)^{|A|}\mathbf{E}_D[f\chi_A]] \leq \mathbf{E}_{A \subseteq V}[|\mathbf{E}_D[f\chi_A]|]$. Also, since T is a term of f, for any x such that $f(x) = -1$, $T'(x) = 0$. Thus $\mathbf{E}_D[fT'] = \mathbf{E}_D[T'] = \mathbf{Pr}_D[T = 1]$.

Therefore, there is some $\chi_A \in T$ such that $|\mathbf{E}_D[f\chi_A]| \geq (\mathbf{E}_D[f]+1)/2s$. Since $\mathbf{E}_D[f] = \mathbf{E}_D[f\chi_\emptyset]$, the proof is completed by noting that the inequality above implies that either $|\mathbf{E}_D[f\chi_A]| \geq 1/(2s+1)$ or $\mathbf{E}_D[f] \leq -1/(2s+1)$. □

6.2 Nonuniform Weak DNF Learning

Fact 6.1 says that for every DNF f and every distribution D there exists a parity weakly approximating f with respect to D. This suggests that we may be able to strongly learn DNF by boosting a weak DNF learner that produces parity functions as the weak hypotheses. The question, of course, is whether or not there is an efficient algorithm for finding appropriate parity functions.

We already know the answer to this question when the problem is restricted to finding a weakly-approximating parity with respect to uniform: the `KM` algorithm can efficiently solve this problem. Thus the `KM` algorithm is a natural basis for the more general weak learner we desire. In fact, it is known that the Kushilevitz/Mansour uniform-distribution learnability results can be generalized to learn parity decision trees with respect to a certain subclass of the class of product distributions [Bel91] (product distributions are defined in Chapter 9). However, it is not clear that an algorithm for weakly learning with respect to this distribution class can be boosted into a strong learner; certainly a richer class of distributions is required if we plan to use an existing boosting algorithm such as `F1`.

Thus we seek an efficient mechanism for finding weakly-approximating parity functions with respect to a fairly broad class of distributions. However, as discussed in Chapter 5, we do not expect to be able to efficiently find a weakly-approximating parity with respect to completely arbitrary distributions. What we are seeking, then, is an algorithm that is apparently the first of its kind: an algorithm that learns efficiently with respect to a rather general distribution class (in particular, one with no independence assumptions), but also an algorithm that does not necessarily efficiently learn over arbitrary distributions. This presents quite a puzzle!

The solution to this puzzle begins with the following simple but critical observations. What we would like is an algorithm that generalizes KM, which recall is given a threshold θ and uses membership queries on the target f to find the index A of a Fourier coefficient $\hat{f}(A)$ such that $|\hat{f}(A)| \geq \theta$. Because a Fourier coefficient $\hat{f}(A)$ is by definition $\mathbf{E}[f \cdot \chi_A]$, finding the index A of a large Fourier coefficient $\hat{f}(A)$ leads us to a parity χ_A that weakly approximates f with respect to uniform. We would like a more general algorithm that, given both θ and (in a form to be determined) a distribution D, finds a χ_A such that $|\mathbf{E}_D[f \cdot \chi_A]| \geq \theta$. While the expected value in this relation cannot be viewed as a Fourier coefficient of f as was the case when D was uniform, notice that

$$\mathbf{E}_D[f \cdot \chi_A] = \sum_x f(x)\chi_A(x)D(x) = \frac{1}{2^n}\sum_x 2^n f(x)D(x)\chi_A(x).$$

Thus if we take $g(x) = 2^n f(x)D(x)$ then we have that for all A, $\mathbf{E}_D[f \cdot \chi_A] = \hat{g}(A)$. That is, finding a large Fourier coefficient $\hat{g}(A)$ of g will lead us to a parity χ_A that weakly approximates f with respect to D.

Therefore, we have reduced the problem of efficiently finding a well-correlated parity with respect to arbitrary distributions D to efficiently finding a large Fourier coefficient of a function g which is essentially the product of the target f and the distribution D. If we had a membership oracle for g and if g was Boolean then we could apply KM directly to the problem of finding a large coefficient of g. In fact, if we are given an oracle for distribution D (a function that given input x returns the weight assigned to x by D) along with a membership oracle for f then it is a simple matter to simulate an oracle for $g = 2^n fD$. Also, as shown in Chapter 4, a modified version of KM can find the large Fourier coefficients of non-Boolean g in time polynomial in $L_\infty(g)$ as well as in the normal parameters of KM. For g of the form we are interested in, this means that the algorithm runs in time polynomial in $L_\infty(2^n D)$. These observations form the basis of an algorithm `WDNF` for efficiently learning DNF (in a weak sense) with respect to a broad class of distributions:

Lemma 6.1 *Let D represent both a probability distribution over $\{0,1\}^n$ and the corresponding distribution oracle. There is an algorithm* `WDNF` *such that for any function $f : \{0,1\}^n \to \{-1,+1\}$, for any probability distribution D on the instance space of f, and for any positive δ,* `WDNF`$(EX(f,D), MEM(f), D, \delta)$ *finds, with probability at least $1-\delta$, a Boolean function h such that $\mathbf{E}_D[fh] = \Omega(s^{-1})$, where s is the DNF-size of f. The probability of success is taken over the random choices made by the* `WDNF` *algorithm and by the oracle $EX(f,D)$. The algorithm, when it succeeds, runs in time polynomial in n, s, $\log(\delta^{-1})$, and $L_\infty(2^n D)$. The weak hypothesis h is a parity function (possibly negated).*

Proof: Let $g(x) = 2^n f(x) D(x)$. Then by Fact 6.1 and the above argument there is some χ_A such that $|\mathbf{E}_D[f\chi_A]| = |\hat{g}(A)| \geq 1/(2s+1)$. Thus by Lemma 4.1, `KM'`$(n, MEM(g), \theta = 1/(2s+1), L_\infty(g) = L_\infty(2^n D), \delta)$ will, with probability at least $1-\delta$, find a χ_A such that $|\mathbf{E}_D[f\chi_A]| = \Omega(s^{-1})$. Furthermore, the `KM'` algorithm when given these parameters runs in time polynomial in n, s, $\log(\delta^{-1})$, and $L_\infty(2^n D)$.

Note, however, that `WDNF` is not given the values of s or of $L_\infty(2^n D)$. We circumvent this difficulty as illustrated in Figure 6.1. The primary fact to note is that if `KM'` is called using values s' and $L'_\infty(2^n D)$ that are larger than their respective true values, `KM'` will still succeed (with high probability) at finding an appropriate set of large Fourier coefficients of g. However, the algorithm may run longer than it would have run had the smaller values been used. Therefore, we use a simple guess-and-double technique that quickly converges on parameter values that are larger than necessary and yet small enough to maintain the desired performance guarantees. To assure that the overall algorithm succeeds with the desired confidence, we require successively smaller probabilities of failure δ' of `KM'` as each new set of parameter guesses is used.

There is still another difficulty: because we are guessing the value of $L_\infty(2^n D)$, we do not have any guarantee that a nonempty set S returned by `KM'` has the large Fourier

Invocation: $h \leftarrow \mathtt{WDNF}(EX(f,D), MEM(f), D, \delta)$
Input: Example oracle $EX(f,D)$ for target f and distribution D; membership oracle $MEM(f)$ for f; distribution oracle D; $\delta > 0$
Output: h such that, with probability at least $1-\delta$, $\mathbf{Pr}_D[f = h] \geq \frac{1}{2} + \frac{1}{8s+4}$, where s is the DNF-size of f.

$s \leftarrow 1$; $L \leftarrow 1$; $\delta' \leftarrow \delta/2$; $M \leftarrow 0$
repeat
  $MEM(g)(x) \equiv 2^n \cdot MEM(f)(x) \cdot D(x)$
  $\theta \equiv 1/(2s+1)$
  $S \leftarrow \mathtt{KM}'(n, MEM(g), \theta, L, \delta'/2)$
  if $|S| \leq 2L^2/\theta^2$ **then**
    for each $B \in S$ **do**
      $m \equiv \lceil 8 \ln(8L^2/\delta'\theta^2)/\theta^2 \rceil$
      $t \leftarrow 0$
      for m times **do**
        $\langle x, \ell \rangle \leftarrow EX(f,D)$
        $t \leftarrow t + \ell \cdot \chi_B(x)$
      enddo
      $\mu' \leftarrow t/m$
      if $|\mu'| \geq 3\theta/4$ and $|\mu'| > |M|$ **then** $M \leftarrow \mu'$; $A \leftarrow B$
    enddo
  endif
  $s \leftarrow 2s$; $L \leftarrow 2L$; $\delta' \leftarrow \delta'/2$
until $M \neq 0$
return $h \equiv \text{sign}(M) \cdot \chi_A$

Figure 6.1: The weak DNF learning algorithm WDNF. The notation "$MEM(g)(x)$" represents the value returned by the (simulated) membership oracle for g on input x.

coefficient property (KM$'$ may be using a number of examples insufficient to accurately estimate the required Fourier coefficients). However, we can use calls to the example oracle $EX(f, D)$ to estimate $\mathbf{E}_D[f\chi_A]$ for each $A \in S$. By Hoeffding (Lemma 2.2), if we use $m = \lceil 8\ln(8L^2/\delta'\theta^2)/\theta^2 \rceil$ examples then, with probability at least $1 - \delta'\theta^2/4L^2$, our estimate μ' is within $\theta/4$ of $\mathbf{E}_D[f\chi_A]$. Therefore, if $|\mu'| \geq 3\theta/4$ then $|\mathbf{E}_D[f\chi_A]| \geq \theta/2$. Also note that if a sufficiently large value of L is used in a call to KM$'$ then by the proof of Lemma 4.1, $|S| \leq 2L^2/\theta^2$ with high probability. Therefore, if the algorithm detects that the size of a set S returned by KM$'$ exceeds this bound, it takes this as evidence that the guessed parameter L was too small and does no further processing on S. Thus by allowing KM$'$ to fail with probability at most $\delta'/2$ and each of the at most $2L^2/\theta^2$ estimates of Fourier coefficients $\hat{g}(A)$ to fail with probability at most $\delta'\theta^2/4L^2$, we have that the overall failure of each pass through the main loop is at most δ'. Furthermore, our choices for δ' assure that the overall failure probability of WDNF is at most δ.

Finally, note that the algorithm terminates (with the required probability) after $O(\log(s) + \log(L_\infty(2^n D)))$ steps, where each step requires time polynomial in n, s, $\log(\delta^{-1})$, and $L_\infty(2^n D)$. That it returns the χ_A that is estimated to be best correlated with f among the parities represented by S guarantees (with the same probability) that the true value of $\mathbf{E}_D[f\chi_A]$ is within a multiplicative factor of 2 of the maximal correlation of any parity function, which is at least $1/(2s+1)$. $\square$

Note that the uniform distribution assigns weight 2^{-n} to all inputs. Thus an immediate corollary of the above lemma is that DNF is weakly learnable (given example, membership, and distribution oracles) with respect to any distribution D such that for all $x \in \{0,1\}^n$, the weight that D assigns to x is at most $p(n, s, \epsilon^{-1}, \delta^{-1})/2^n$ for some fixed polynomial p. In other words, WDNF weakly learns DNF with respect to any distribution D that puts only polynomially more weight on its inputs than does the uniform distribution. We will refer to such distributions as *polynomially-near uniform*.

Thus we now have, as desired, a weak DNF algorithm that efficiently learns with respect to a rather broad class of distributions (given certain oracles, including the nonstandard distribution oracle). The learning algorithm also has the expected property that it does not guarantee efficient learning with respect to arbitrary distributions. This is a promising start; what remains is to show how `WDNF` can be integrated with Freund's `F1` hypothesis boosting algorithm to produce a strong learning algorithm for DNF.

6.3 Strongly Learning DNF

For efficiency, the weak learner `WDNF` requires two properties of the target distribution D: the distribution must be polynomially-near uniform, and an oracle for the distribution must be provided to the learner. Now consider the target distributions D_i generated at each stage of Freund's `F1` boosting algorithm when the booster's goal is to produce an ϵ-approximating hypothesis with respect to uniform. As we will see below, these target distributions D_i are polynomially-near uniform. In fact, this is true of all boost-by-filtering algorithms. However, a second property of `F1`'s distributions D_i is not true of the distributions of some other boosting algorithms: each of `F1`'s distributions is defined—modulo a scale factor—by a polynomial-time computable function. The scale factor can also be estimated efficiently. Therefore, an approximate distribution oracle can be provided to the weak learner for each target distribution D_i generated during boosting. Furthermore, we show that the weak learner does not require an exact distribution oracle; this approximate oracle suffices.

Putting this all together gives an algorithm for strongly learning DNF with respect to uniform. We call our algorithm the Harmonic Sieve (`HS`) because conceptually it repeatedly finds a dominant harmonic (parity function that correlates well with the target) and then damps out its influence (shifts the distribution) so that another harmonic becomes dominant.

Theorem 6.1 *DNF is learnable with respect to the uniform distribution using membership queries.*

Proof: We will actually prove a somewhat more general result: the Harmonic Sieve algorithm, given an oracle for any probability distribution D along with the usual parameters, learns DNF with respect to D in time polynomial in n, the DNF-size s of the target f, ϵ^{-1}, $\log(\delta^{-1})$, and $L_\infty(2^n D)$. As indicated above, the algorithm is essentially an application of `F1`

Invocation: $h \leftarrow \texttt{HS}(EX(f,D), MEM(f), D, s, \epsilon, \delta)$
Input: Example oracle $EX(f,D)$ for target f and distribution D; membership oracle $MEM(f)$; distribution oracle D; DNF-size s of f; $\epsilon > 0$; $\delta > 0$
Output: h such that, with probability at least $1-\delta$, $\mathbf{Pr}_D[f = h] \geq 1 - \epsilon$

$\gamma \leftarrow 1/(8s+4)$
$k \leftarrow \frac{1}{2}\gamma^{-2}\ln(4/\epsilon)$
$w_0 \leftarrow \texttt{WDNF}(EX(f,D), MEM(f), D, \delta/2k)$
for $i \leftarrow 1, \ldots, k-1$ **do**
  $r_i(x) \equiv |\{0 \leq j < i \mid w_j(x) = f(x)\}|$
  $B(j;n,p) \equiv \binom{n}{j} p^j (1-p)^{n-j}$
  $\tilde{\alpha}_r^i \equiv B(\lfloor k/2 \rfloor - r; k-i-1, 1/2+\bar{\gamma})$ if $i - k/2 < r \leq k/2$, $\tilde{\alpha}_r^i \equiv 0$ otherwise
  $\alpha_r^i \equiv \tilde{\alpha}_r^i / \max_{r=0,\ldots,i-1}\{\tilde{\alpha}_r^i\}$.
  $\Theta \equiv \epsilon^3/57$
  $X \equiv$ draw example $\langle x, f(x)\rangle$ from $EX(f,D)$ and compute $\alpha^i_{r_i(x)}$
  $E_\alpha \leftarrow \texttt{AMEAN}(X, b-a=1, \frac{1}{3}\Theta, \delta/2k)$
  if $E_\alpha \leq \frac{2}{3}\Theta$ **then**
    $k \leftarrow i$
    break do
  endif
  $D_i(x) \equiv \frac{D(x)\alpha^i_{r_i(x)}}{\sum_y D(y)\alpha^i_{r_i(y)}}$
  $D_i'(x) \equiv (D(x)\alpha^i_{r_i(x)})/E_\alpha$
  $w_i \leftarrow \texttt{WDNF}(EX(f,D_i), MEM(f), D_i', \delta/2k)$
enddo
$h(x) \equiv MAJ(w_0(x), w_1(x), \ldots, w_{k-1}(x))$
return h

Figure 6.2: The HS algorithm for efficiently learning DNF. AMEAN is described in Corollary 2.1. X is a random variable used to estimate $\mathbf{E}_D[\alpha^i_{r_i(x)}]$.

to boosting the weak learner WDNF developed in the preceding section. Figure 6.2 describes the algorithm. For simplicity, this version of the algorithm assumes that the DNF-size s of the target f is provided; this assumption is easily removed by modifying the algorithm to use a guess-and-double technique similar to that used in WDNF. Note also that in most respects the main procedure of the Harmonic Sieve is identical to F1. The major difference is that HS produces an approximate distribution oracle D'_i at every stage of boosting and provides this oracle to the weak learner.

To see that this algorithm satisfies the requirements of the theorem, first note that if WDNF succeeds at producing a $(\frac{1}{2} - \frac{1}{8s+4})$-approximate weak hypothesis with respect to D_i at each stage i then Lemma 5.1 shows that the hypothesis produced by HS will be an ϵ-approximator to the target f with respect to the target distribution D. Furthermore, by Lemma 6.1, WDNF will produce such a hypothesis (with high probability) given an oracle for D_i. This presents the following difficulty: to simulate an exact oracle for D_i (line 16 of Figure 6.2) requires computing exactly the exponentially large sum $\sum_y D(y)\alpha^i_{r_i(y)}$.

To circumvent this difficulty, HS provides WDNF with the approximate oracle D'_i mentioned above (line 17). Note from line 11 that HS, like F1, uses random sampling of the example oracle $EX(f, D)$ to estimate an approximation E_α to the sum $\sum_y D(y)\alpha^i_{r_i(y)}$. This estimate is, with high probability, within an additive factor $\Theta/3$ of the true value. Because the test at line 12 assures (with high probability) that every D'_i is computed using a value of E_α that exceeds $2\Theta/3$, $\frac{2}{3}\sum_y D(y)\alpha^i_{r_i(y)} \leq E_\alpha \leq 2\sum_y D(y)\alpha^i_{r_i(y)}$. This implies that there is a constant $c \in [1/2, 3/2]$ such that for all x, $D'_i(x) = cD_i(x)$.

Now consider the functional impact of supplying this approximate oracle rather than the true oracle to WDNF (we consider the impact on running time below). WDNF uses its given distribution oracle (call it D_W) for exactly one purpose: to simulate a membership oracle for the function $g = 2^n f D_W$. Thus the only impact of multiplying D_W by a constant c is to multiply the function g by the same constant. Furthermore, the Fourier transform is a linear operator, and thus $\widehat{c \cdot g}(A) = c \cdot \hat{g}(A)$ for all A. In summary, multiplying WDNF's distribution oracle by a constant has the effect of multiplying all of the Fourier coefficients of the induced function g by the same constant. Because the relative sizes of the coefficients are unchanged, multiplying g by a constant will not adversely affect the ability of `KM'` to find the large Fourier coefficients of g.

Thus changing the distribution oracle by a constant factor has no expected impact on the functionality of `WDNF`. Therefore, by the earlier argument, `HS` will return an ϵ-approximator to the target f.

The only remaining concern is with the running time of the algorithm. First, recalling that `HS` is very similar to the polynomial-time algorithm `F1`, note that `HS` also runs in time polynomial in the appropriate parameters if two conditions are met: both the number of boosting stages k of `HS` and the running time of `WDNF` must be bounded by polynomials. Clearly the number of boosting stages is polynomially bounded. Thus all that remains is to bound the running time of `WDNF`.

By Lemma 6.1, if `WDNF` was invoked with a distribution oracle for D_i then the running time of `WDNF` would be appropriately bounded, since $L_\infty(D_i) \leq 3L_\infty(D)/\Theta$ and Θ is polynomial in ϵ. Also notice that if the distribution oracle is multiplied by a constant factor then the only impact on the running time of `WDNF` comes from changes to the running time of the call to `KM'`. There are two potential effects on the running time of `KM'` resulting from multiplication of D_W by c. First, because this multiplication may reduce the magnitude of the Fourier coefficients of g, it may be necessary to use a smaller threshold value θ in the call to `KM'` in order to find a large Fourier coefficient. Second, because the running time of `KM'` depends on $L_\infty(2^n D_W)$, this multiplication may directly increase the time bound. However, in both cases the running time is increased by at most a small constant if the multiplicative factor c is near 1, as we guaranteed earlier. Thus the running time of `WDNF`, and therefore of `HS`, is bounded by a polynomial in the desired parameters. □

The final hypothesis output by `HS` is not a DNF but a threshold of parity functions, *i.e.*, a TOP representation. We will have more to say about the relationships between `HS`, DNF, and TOP in the next chapter. For now, notice that this is the same representation output by the Fourier-based learning algorithms discussed in Chapter 4. However, `HS` arrives at its hypothesis in a very different manner than previous Fourier learning algorithms, as discussed in the next section.

6.4 Comparison with Prior Fourier-based Algorithms

The existence of the Harmonic Sieve proves that DNF is learnable in a strong sense. Beyond that, as mentioned previously, the Harmonic Sieve is unique in several ways. The algorithm demonstrates that boosting can be used to prove a positive learnability result for a class for which no other proof technique is known. It also provides the first known example of a distribution-dependent weak learner that can be boosted by an algorithm designed for distribution-independent boosting. And HS introduces the notion of learning from a distribution oracle.

However, a particularly interesting feature of the Harmonic Sieve, as discussed somewhat in Section 4.2.5, is its use of an algorithmic approach that differs markedly from earlier Fourier-based learning algorithms. Recall that Mansour [Man92] had presented evidence that approaches to Fourier-based learning based strictly on finding the large Fourier coefficients of the target were apparently not adequate for learning DNF. HS is based on a new approach to Fourier-based learning: find large Fourier coefficients of functions other than the target.

Alternatively, as pointed out above, this approach can be viewed as finding parity functions that correlate well with the target with respect to a variety of distributions. Although the distributions considered by HS are polynomially-near uniform in the sense defined earlier, even relatively small deviations from uniform can significantly affect how well a given parity approximates the target. The fact that HS finds parities that correlate well with the target over a range of distributions means that the algorithm can potentially find "important" parity functions that might be missed by earlier Fourier-based learning algorithms. Furthermore, even if all of the parity functions included in the final HS hypothesis have large Fourier coefficients, the weights of the parities in the HS hypothesis are not necessarily close estimates of Fourier coefficients, as was the case with previous algorithms.

However, it is possible that, say, the Kushilevitz-Mansour algorithm run with an appropriate (inverse-polynomial) threshold could learn the same classes learnable by HS. That is, it may be that our DNF learnability result has more to do with the analytical approach taken than the HS algorithm itself. Whether or not this is the case is an interesting question for further research.

In the next chapter we present further learnability results that follow from this new Fourier-based approach and also discuss some limits on this method.

Chapter 7

Learning Other Representation Classes

In this chapter we attempt to generalize our DNF learning result to other representation classes with Boolean inputs; our results are both positive and negative. Our primary positive result is that the representation class TOP of Thresholds of Parities (Figure 1.2) is learnable by the Harmonic Sieve. We then discuss several classes that are restricted versions of TOP and that therefore are also learnable with respect to uniform. Finally, we consider two other representation classes—depth-2 threshold circuits and depth-3 Boolean circuits—that cannot be learned efficiently by the Harmonic Sieve.

7.1 Learning TOP

TOP, like DNF, is a universal class. Thus the learnability result we would like to show is that arbitrary Boolean functions are learnable in time polynomial in their TOP-size, *i.e.*, in the number of parities in the smallest TOP representing the target function. To prove this, we first prove that all TOP's have a certain property: for every distribution D, every TOP f is weakly approximated with respect to D by some parity function. The analogous property for DNF (Fact 6.1) was the only property of DNF used in the proof that DNF is learnable. Thus, given this property for TOP we can immediately apply the proof of Theorem 6.1 to show that TOP is learnable with respect to uniform by `HS`. Moreover, TOP is properly learnable, since `HS` produces a TOP as its hypothesis.

The TOP property we seek is actually a corollary of a theorem of Goldmann, Hastad, and Razborov [GHR92]. We give a relatively simple alternate proof similar to one of Blum *et al.* [BFJ$^+$94] for a more specific TOP property.

Fact 7.1 *For every f the majority of s parity functions and for every distribution D on the instance space of f, one of the parity functions χ_A defining f satisfies $|\mathbf{E}_D[f\chi_A]| \geq 1/(2s+1)$.*

Proof: First, we would like the number s of parity functions in f to be odd so that the "vote" of these functions will always be nonzero. Note that if s is even then it is straightforward to create a new circuit with $2s+1$ parities computing the same function (each of the initial parities is replicated and the constant parity $\chi_\emptyset$ is added). Thus for any f representable as a majority of s parity functions there is some $F = \sum_A \hat{F}(A)\chi_A$ such that $f = \text{sign}(F)$ (where sign(0) is undefined), $L_1(\hat{F}) \leq 2s+1$, and all the coefficients $\hat{F}(A)$ of F are integers. Specifically, this F is simply the sum of the parity functions defining f (or defining the circuit with an odd number of parity functions if s is even). Note that $|F(x)| \geq 1$ for all x, and since $f = \text{sign}(F)$, we have

$$\begin{aligned} 1 &\leq \mathbf{E}_D[|F|] = \mathbf{E}_D[fF] = \mathbf{E}_D[f\sum_A \hat{F}(A)\chi_A] = \sum_A \hat{F}(A)\mathbf{E}_D[f\chi_A] \\ &\leq \max_A\{|\mathbf{E}_D[f\chi_A]|\}\sum_A |\hat{F}(A)|. \end{aligned}$$

□

Theorem 7.1 *TOP is properly learnable with respect to uniform using membership queries.*

7.2 Some Specialized TOP Classes

TOP is a fairly expressive class of representations. For example, it has been shown recently that for every Boolean function f, the TOP-size of f is at most polynomially larger than the DNF-size of f [KP94]. Actually, we get as a corollary of our learning result an alternate proof of this relationship (this method of obtaining complexity results from boosting results has been noted before [Fre93]).

Corollary 7.1 *Every DNF f with s terms can be computed as the majority of $O(ns^2)$ parity functions.*

Proof: Run `HS` with a simulated membership oracle for f, with $\epsilon = 2^{-(n+1)}$, with $\delta < 1$, and with D the uniform distribution. By the proof of Theorem 6.1, `HS` produces a majority of $O(ns^2)$ parity functions as its hypothesis. Then note that, while `HS` is no longer guaranteed to run efficiently in n, it is still guaranteed to produce with high probability a hypothesis h with the following properties:

1. h is the majority of $k = O(ns^2)$ parities.

2. $\mathbf{Pr}[h \neq f] < 2^{-n}$.

For the second property to hold, it must be that $h \equiv f$. Since the algorithm succeeds—with positive probability—at finding an h with these properties, such an h must exist. □

Before going further, we need some notation. In particular, we have previously used the notion of a subclass of a representation class; now we generalize this notion somewhat. We will view a representation class $\mathcal{F}$ as a *restriction* (or a *specialization*) of a class $\mathcal{H}$ if there exists a fixed polynomial p such that for every n and every Boolean f over $\{0,1\}^n$, $\mathcal{H}$-size of f is at most $p(n,s)$, where s is the $\mathcal{F}$-size of f. Given such an $\mathcal{F}$ and $\mathcal{H}$, learnability of $\mathcal{H}$ implies non-proper learnability of $\mathcal{F}$. In other words, the restricted class, while it may be capable of representing arbitrary Boolean functions, is from a learning perspective an easier class. We will also say that $\mathcal{H}$ *generalizes* $\mathcal{F}$ if $\mathcal{F}$ is a restriction of $\mathcal{H}$. If $\mathcal{H}$ generalizes $\mathcal{F}$ but not vice versa, then $\mathcal{H}$ *strictly* generalizes $\mathcal{F}$.

Thus what we have shown above is that DNF is a restricted class of TOP. In fact, the restriction is strict, as n-bit parity has only an exponentially large DNF representation. In addition to DNF, TOP strictly generalizes another universal class, this class with the size of a function f measured not in terms of how f is expressed but instead in terms of a Fourier property of the function f itself. In particular, define the *L_1-size* of a Boolean function f to be $L_1(\hat{f})$, that is, the sum of the magnitudes of the Fourier coefficients of f, and let CL_1 be the class of arbitrary Boolean functions with this size measure. Bruck and Smolensky [BS90] have shown that TOP strictly generalizes CL_1. In fact, CL_1 is the class that Kushilevitz and Mansour showed to be efficiently learnable with respect to uniform using membership queries; the fact that the class of parity decision trees is similarly learnable is a corollary of the fact that this class specializes CL_1 [KM93]. Because CL_1 in turn specializes TOP, HS can efficiently learn every class previously shown learnable by Kushilevitz and Mansour.

As another example of a restricted TOP class (and therefore a class learnable by the Harmonic Sieve), consider the class in which each function representation in the class is a threshold of functions g_i, and where each g_i in turn is an arbitrary function of fanin $O(\log n)$. To see that this class specializes TOP, consider the Fourier transform of a function g defined over a set S of the input bits. For all $A \subseteq \{1, \ldots, n\}$ such that $A - S \neq \emptyset$, $\hat{g}(A) = 0$ (the argument for this is similar to the argument that the χ_A's are orthogonal). Thus g has at most $2^{|S|}$ non-zero Fourier coefficients. Furthermore, we may compute the coefficient $\hat{g}(A)$ for any $A \subseteq S$ by fixing the bits indexed by $\{1, \ldots, n\} - S$ to arbitrary values in $\{0, 1\}$ and computing the expectation $\mathbf{E}[g\chi_A]$ over all possible assignments to the bits indexed by S. This means that every non-zero coefficient may be expressed as a rational with denominator $2^{|S|}$. Therefore $2^{|S|}g$ is expressible as the sum of $O(2^{2|S|})$ parities, at most $O(2^{|S|})$ of which are distinct. So given a threshold of s many functions g_i each on $O(\log n)$ inputs, we create for each g_i a polynomial-size (in n) sum of parities computing $2^c g_i$, where c is the maximum fanin of any g_i. Taking a threshold over these sums gives a threshold over polynomially many (in n and s) parities that computes the original function.

As a final example of a class that specializes TOP, consider the class parity-DNF of representations that are depth-3 circuits with an OR at the top level, AND's in the middle, and parity functions at the bottom (Figure 7.1). Parity-DNF also specializes TOP. The high level idea behind this observation is the following. The Fourier representation of a conjunction of parities is very similar to a conjunction of literals: it is roughly an "average" of a collection of parity functions plus a constant. Thus we can prove, much as we did for DNF, that for every parity-DNF f and every distribution D there is a parity that weakly approximates f with respect to D. Thus the proof that DNF is learnable as TOP can

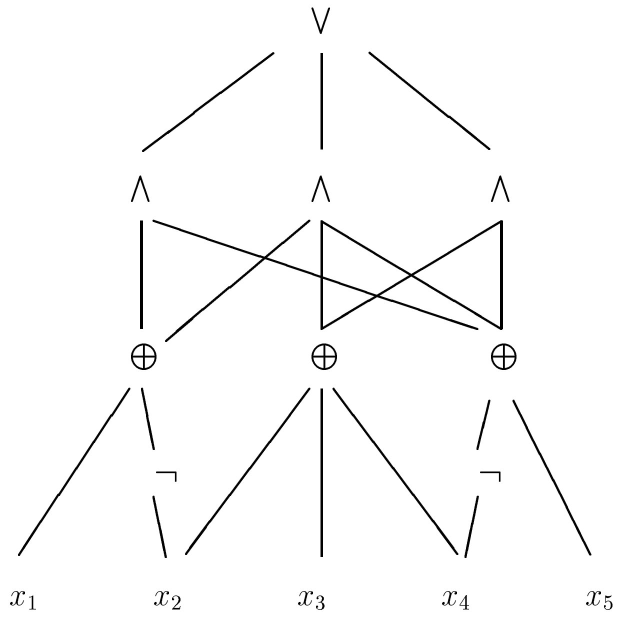

Figure 7.1: A parity-DNF function.

readily be generalized to show that parity-DNF is also learnable as TOP. Then the proof of Corollary 7.1 shows that parity-DNF is a restriction of TOP.

7.3 Two Classes Not Generalized by TOP

The Harmonic Sieve algorithm does not readily extend to learning two representation classes that are seemingly small generalizations of DNF and TOP. In particular, Bruck [Bru90] has shown that there is a polynomial-size (in n) depth-2 threshold circuit (Figure 1.5) f_1 such that $L_\infty(\hat{f}_1) = 2^{-n/2}$, and an argument of Bruck and Smolensky [BS90] can be used to construct a polynomial-size depth-3 $\{\wedge, \vee, \neg\}$-circuit (Figure 3.3) f_2 such that $L_\infty(\hat{f}_2) = n^{-\log n}$. Let s be the size of a target function f according to the size measure of either of these two representation classes. Then the Fourier characterizations above say that for each of these classes there are functions f such that *no* parity is correlated inverse-polynomially in s with f, even with respect to the uniform distribution. Thus the `KM'` procedure invoked by `HS` to find the largest Fourier coefficient of a target function will need an inverse superpolynomially in s small threshold value θ. The running time of `KM'`, in turn, is bounded inverse-polynomially in θ, and therefore is not polynomially bounded for these classes. Furthermore, because the maximum magnitude of any Fourier coefficient $\hat{f}(A)$ of a function f upper bounds the inverse of the TOP-size of f (Fact 7.1), these classes are not restrictions of TOP.

As an example of how these hardness results are obtained, we give the depth-3 circuit construction here. First, it can be shown that the Fourier representation of the Boolean function $\wedge(x) = x_1 \wedge x_2$ is

$$\wedge(x) = -\frac{1}{2}\left(1 + \chi_{\{1\}}(x) + \chi_{\{2\}}(x) - \chi_{\{1,2\}}(x)\right).$$

Now consider a circuit that consists of a parity gate that takes a set $\{C_i\}$ of subcircuits as input. We can obtain an arithmetic representation of the function computed by this circuit by taking the product of the Fourier transforms of the subcircuits C_i. Also, note that the product $\chi_A \cdot \chi_B = \chi_{A \Delta B}$, where Δ denotes symmetric difference. Therefore, the parity of $\log n$ 2-bit $\wedge$ functions, where the $\wedge$'s are applied to pairwise-disjoint sets of variables, has a Fourier transform in which every nonzero coefficient has magnitude n^{-1}. On the other hand, this is a function on $2 \log n$ bits, and therefore it can be represented either as a CNF with at most n^2 clauses or as a DNF with at most n^2 terms.

Let g_i represent the parity of $\log n$ 2-bit $\wedge$ functions over pairwise-disjoint variables, and let S_i represent the set of variables in g_i. Also, let g represent the function formed by taking the parity of a set of $\log n$ functions g_i where each g_i is defined over a set of variables S_i that is disjoint from all the other sets. Then, by reasoning similar to the above, the nonzero Fourier coefficients of the resulting function will all have magnitude $n^{-\log n}$. Furthermore, the function can be written as a polynomial-size DNF over inputs that are polynomial-size CNF's, which is a polynomial-size depth-3 circuit.

In the next chapter we consider learnability of a representation class that generalizes DNF in a different way, moving from functions over $\{0,1\}^n$ to functions over more general hypercubes. We show that `HS` can be extended to learn this more general class.

Chapter 8

Learning Geometric Concepts

In this chapter we describe a generalization of the `HS` algorithm that learns certain classes of representations defined over non-Boolean domains. In particular, the representations in these classes specify rectangular regions of the domain where the represented Boolean function is `true` (Figure 1.3). Learnability of geometric representations has received significant interest recently; [BGGM94] contains a nice summary of research in this area. We briefly mention how our results relate to a few of the previous results below, but first we define the class of primary interest.

In fact, before we formally define the geometric classes of interest, first note that the positive instances of an s-term DNF f are covered exactly by the union of s subcubes of the Boolean hypercube $\{0,1\}^n$. Based on this view of DNF, we define the following generalization of DNF over the domain $[b]^n$ (we use $[b]$ to represent $\{0,\ldots,b-1\}$). For l and u in $[b]^n$ such that $\ell_i \leq u_i$ for all i, the *rectangle* $[\ell, u]$ on $[b]^n$ is the set of instances $\prod_{i=1}^{n}\{\ell_i, \ell_i+1, \ldots, u_i\}$. Thus rectangles are a natural generalization of the terms of a DNF, and unions of rectangles are a natural generalization of DNF expressions. Therefore, we will be interested in the learnability of the representation class UBOX of unions of (axis-parallel) rectangles. The *size of a UBOX representation* r is the number of rectangles in r.[1]

We will show that if b is a constant then UBOX is learnable with respect to the uniform

[1] For consistency with the DNF portion of this paper, the definitions given here deviate from standard notation for geometric concepts (*e.g.* [BGGM94]). Typically, the dimension n of the instance space is denoted by d, the number of attribute values b is denoted by n, and the number of rectangles s is denoted by m.

distribution using membership queries in time polynomial in the usual parameters. This complements results in stronger models of learning showing polynomial-time learnability for UBOX when either the dimension n or the number of boxes is held constant [BEHW89, LW90, GGM94, BGGM94].

We begin by showing how to extend the Harmonic Sieve algorithm to learning a subclass of UBOX that generalizes the class k-DNF of DNF expressions where each term has at most k literals. A *k-rectangle* $[\ell, u]$ on $[b]^n$ is a rectangle such that $|\{i \mid \ell_i \neq 0 \text{ or } u_i \neq b-1\}| \leq k$. k-UBOX is the restriction of UBOX to unions of k-rectangles. After showing that k-UBOX is learnable with respect to the uniform distribution for restricted values of k, we show that the same algorithm can learn unrestricted UBOX in time exponential in $b \log \log b$ and thus in polynomial time for constant b.

Before developing these algorithms, we show how to generalize the `KM` algorithm for learning over the domain $[b]^n$.

8.1 Generalizing `KM`

The UBOX algorithm we develop will, like `HS`, build on an extension of `KM`. In particular, the Kushilevitz/Mansour Fourier analysis that underlies the `KM` algorithm can be generalized to produce an algorithm that finds the large Fourier coefficients of a Boolean function on $[b]^n$. Recall from Chapter 4 that the Fourier basis functions are now of the form

$$\chi_a(x) = \omega_b^{\sum_i a_i x_i},$$

where $a \in [b]^n$ and $\omega_b = e^{2\pi\sqrt{-1}/b}$. These functions form a basis for the space of all complex-valued functions on $[b]^n$, and every function $f : [b]^n \to \mathbf{C}$ can be uniquely expressed as a linear combination of the χ functions: $f = \sum_a \hat{f}(a)\chi_a^*$, where χ^* represents the complex conjugate of χ and $\hat{f}(a) = \mathbf{E}[f\chi_a]$. Also, for such functions f Parseval's identity holds, although now it is expressed in terms of magnitudes:

$$\mathbf{E}_x\left[|f(x)|^2\right] = \sum_a \left|\hat{f}(a)\right|^2.$$

Finally, the basis functions are orthonormal in the sense that

$$\mathbf{E}_x\left[\chi_a(x)\cdot\chi_b^*(x)\right] = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

To find the large Fourier coefficients of a function $f : [b]^n \to \{-1, +1\}$, we first define the set $C_{a,k}$, where k is an integer in $[0, n]$ and a is a vector in $[b]^k$, by generalizing the earlier definition of $C_{A,k}$ (Section 4.3.1) in the obvious way. Then the Kushilevitz-Mansour Fourier analysis can be generalized to show that for every f, k, and a as above,

$$L_2^2(C_{a,k}) = \mathbf{E}_{x,y,z}[\mathrm{Re}(f^*(yx)f(zx)\chi_a(y-z))], \tag{8.1}$$

where $L_2^2(C_{a,k})$ represents the sum of squares of the magnitudes of the elements of $C_{a,k}$; the expectation is uniform over $x \in [b]^{n-k}$, $y \in [b]^k$, and $z \in [b]^k$; and $y - z$ represents the difference between y and z in $\mathbf{Z}_b^k$.

To derive this equation, fix k, let a be any vector in $[b]^k$, and let x be any vector in $[b]^{n-k}$. Then we define $f_a(x)$ in terms of a subset of the Fourier coefficients of f:

$$f_a(x) = \sum_d \hat{f}(ad)\chi_d^*(x)$$

where the sum is over all vectors $d \in [b]^{n-k}$. Then by Parseval's, $\mathbf{E}_x[|f_a(x)|^2] = L_2^2(C_{a,k})$. Next note that

$$\begin{aligned}
\mathbf{E}_{y\in[b]^k}[f(yx)\chi_a^*(y)] &= \mathbf{E}_y\left[\sum_{c\in[b]^k, d\in[b]^{n-k}} \hat{f}(cd)\chi_{cd}^*(yx)\chi_a^*(y)\right] \\
&= \sum_{c,d} \hat{f}(cd)\chi_d^*(x)\mathbf{E}_y\left[\chi_c(y)\chi_a^*(y)\right] = \sum_d \hat{f}(ad)\chi_d^*(x) \\
&= f_a(x).
\end{aligned}$$

Equation 8.1 follows by some algebraic manipulations and simplification.

The generalized algorithm, then, is exactly the same as KM, except it uses the relationship above to estimate $L_2^2(C_{a,k})$ for b sets (one for each possible value of a_k) during each recursive call. This generalized KM runs in time polynomial in b as well as in the original parameters. KM$'$ can be similarly generalized.

8.2 Learning k-UBOX

The main result of this section is that k-UBOX is learnable for certain values of k. To obtain this result, we first generalize Fact 6.1 to the k-UBOX class. Then we will show how to convert the basis functions corresponding to the large Fourier coefficients found by the generalized $\mathtt{KM}'$ outlined above into Boolean weak hypotheses.

8.2.1 A Fact about k-UBOX

Now we show that for every k-UBOX function f (with k appropriately limited) and for every distribution D there is some Fourier basis function χ_a that correlates well with f with respect to D.

Fact 8.1 *Let $f : [b]^n \to \{-1, +1\}$ be any union of s k-rectangles, and let D be any probability distribution on $[b]^n$. Then there is some χ_a such that $|\mathbf{E}_D[f\chi_a]| = \Omega(s^{-1}\log^{-k} b)$.*

Proof: We consider two cases. The simple case is when $\mathbf{E}_D[f] \geq 1/2s$. In this case, the constant function $\chi_{\bar{0}}$ satisfies the lemma.

Otherwise, let $p = \mathbf{Pr}_D[f = +1]$. Note that for this case of the proof we have $\mathbf{E}_D[f] \leq 1/2s$, and thus $p \leq 1/2 + 1/4s$. Also, there is some rectangle R such that $\mathbf{Pr}_D[x \in R] \geq p/s$. Let $R(x)$ be the Boolean function represented by R, *i.e.*, $R(x) = 1$ when x is in rectangle R and $R(x) = -1$ otherwise. Then, since $f = -1$ implies $R = -1$, $\mathbf{Pr}_D[R = f] \geq (1-p) + p/s$. It follows that $\mathbf{E}_D[fR] \geq 1/2s$. Also, by an argument similar to one in the proof of Fact 7.1, $\mathbf{E}_D[fR] \leq \max_a\{|\mathbf{E}_D[f\chi_a]|\} \cdot \sum_a |\widehat{R}(a)|$. Thus the proof is complete once we show that $L_1(\widehat{R}) = O(\log^k b)$.

To see that $L_1(\widehat{R})$ is thus bounded, first observe that the function $(R(x) + 1)/2$ can be written as the product of at most k functions $R_i(x)$, where each $R_i(x)$ is a 1-rectangle with range $\{0, 1\}$. Moreover, it can be shown that $L_1(\widehat{R}) \leq 2\prod_i L_1(\widehat{R}_i)$ [KM93]. Thus $L_1(\widehat{R}) = O((\max L_1(\widehat{R}_i))^k)$, where the maximum is over all possible 1-rectangles R_i producing outputs in $\{0, 1\}$. In fact, this maximum is the same as the maximum over all 1-rectangles $R_i^1 : [b] \to \{0, 1\}$, since the value of a 1-rectangle is determined by a single variable. In the appendix of this chapter we show that $\max L_1(\widehat{R}_i^1) = O(\log b)$. $\square$

8.2.2 Turning χ_a into a Weak Approximator

Given the above fact and the generalization of `KM′` outlined earlier, we can also generalize the other components of the `HS` algorithm to learn k-UBOX for efficiently for certain k. A key difference between the original and generalized `HS` algorithms is in the form of the weak hypothesis produced at each stage of the algorithm. Unlike the original `HS`, which produced a χ_A that could be used directly as the weak hypothesis, the generalized `HS` finds a complex-valued function χ_a at each stage of the boosting process. The proof of the following lemma describes an algorithm for converting such a χ_a into a weak approximator for Boolean f.

Lemma 8.1 *Let D be a probability distribution over $[b]^n$. There is an algorithm* `TABLE` *such that for any function $f : [b]^n \to \{-1, +1\}$, for any distribution D as above, for any $a \in [b]^n$, and for any positive δ,* `TABLE`$(EX(f, D), a, \delta)$ *returns, with probability at least $1-\delta$, a Boolean hypothesis h_a such that $\mathbf{Pr}_D[h_a \neq f] = 1/2 - \Omega(|\mathbf{E}_D[f\chi_a]|)$. The algorithm runs in time polynomial in n, b, $|\mathbf{E}_D[f\chi_a]|^{-1}$, and $\log(\delta^{-1})$.*

Thus the weak learner for k-UBOX consists of running the generalized `KM′` followed by running `TABLE`, where the inputs to `TABLE` are the index a of the function χ_a found by `KM′` and the simulated oracle $EX(f, D_i)$ defined by stage i of `F1`.

Proof of Lemma: `TABLE`, as the name implies, produces a hypothesis h_a that consists of a table of Boolean entries. In particular, for each $j \in [b]$, the table contains a single entry $h_a(j)$ that defines $h_a(x)$ for all x such that $\chi_a(x) = \omega_b^j$. Thus

$$\mathbf{Pr}_{x \in_R D}[h_a(x) \neq f(x)] = \sum_{j=0}^{b-1} \mathbf{Pr}_{x \in_R D}[h_a(x) \neq f(x) \mid \chi_a(x) = \omega_b^j] \cdot \mathbf{Pr}_{x \in_R D}[\chi_a(x) = \omega_b^j].$$

Let D^j be defined by

$$D^j(x) = \begin{cases} 0 & \text{if } \chi_a(x) \neq \omega_b^j \\ \frac{D(x)}{\mathbf{Pr}_D[\chi_a = \omega_b^j]} & \text{otherwise.} \end{cases}$$

Then

$$\mathbf{Pr}_{x \in_R D}[h_a(x) \neq f(x)] \quad = \quad \sum_{j=0}^{b-1} \mathbf{Pr}_{x \in_R D^j}[h_a(j) \neq f(x)] \cdot \mathbf{Pr}_{x \in_R D}[\chi_a(x) = \omega_b^j]$$

$$= \sum_{j=0}^{b-1} \left(\frac{1}{2} - \frac{\mathbf{E}_{x \in_R D^j}[h_a(j) \cdot f(x)]}{2} \right) \cdot \mathbf{Pr}_{x \in_R D}[\chi_a(x) = \omega_b^j]$$
$$= \frac{1}{2} - \frac{1}{2} \sum_{j=0}^{b-1} h_a(j) \cdot \mathbf{E}_{x \in_R D^j}[f(x)] \cdot \mathbf{Pr}_{x \in_R D}[\chi_a(x) = \omega_b^j].$$

Now note that

$$\mathbf{E}_D[f\chi_a] = \sum_{j=0}^{b-1} \mathbf{E}_{D^j}[f] \cdot \omega_b^j \cdot \mathbf{Pr}_{x \in_R D}[\chi_a(x) = \omega_b^j].$$

and therefore

$$|\mathbf{E}_D[f\chi_a]| \leq \sum_j |\mathbf{E}_{D^j}[f]| \cdot \mathbf{Pr}_{x \in D}[\chi_a(x) = \omega_b^j].$$

Define $h'_a(j) = \mathrm{sign}(\mathbf{E}_{D^j}[f])$. Then

$$\mathbf{Pr}_D[h'_a \neq f] \leq \frac{1}{2} - \frac{1}{2}|\mathbf{E}_D[f\chi_a]|$$

and h'_a is a weak approximator to f for all a such that $|\mathbf{E}_D[f\chi_a]|$ is inverse-polynomially large. Furthermore, for h''_a defined by

$$h''_a(j) = \begin{cases} -h'_a(j) & \text{if } |\mathbf{E}_{D^j}[f]| \leq \frac{|\mathbf{E}_D[f\chi_a]|}{8} \text{ or } \mathbf{Pr}_{x \in_R D}[\chi_a(x) = \omega_b^j] \leq \frac{|\mathbf{E}_D[f\chi_a]|}{8b} \\ h'_a(j) & \text{otherwise} \end{cases}$$

it follows that

$$\mathbf{Pr}_D[h''_a \neq f] \leq \frac{1}{2} - \frac{1}{4}|\mathbf{E}_D[f\chi_a]|$$

This analysis leads to the following algorithm for producing a Boolean approximator h_a from function χ_a. Let $\theta = |\mathbf{E}_D[f\chi_a]|$. Estimate $\mu'_j = |\mathbf{E}_{D^j}[f]|$ for all j such that $|\mathbf{E}_{D^j}[f]| \geq \theta/8$ and such that $\mathbf{Pr}_D[\chi_a = \omega_b^j] \geq \theta/8b$ (call these the *large* j). Define $h_a(j) = \mathrm{sign}(\mu'_j)$ for all j as above and define the value of $h_a(j)$ arbitrarily for the remaining j. Then if the estimates for μ'_j are all within $\theta/8$ of the true values, h_a is at least a $(1/2 - \theta/4)$-approximator to f.

We now briefly outline a method of efficiently finding appropriate μ'_j for all the large j. First, we will estimate θ using calls to $EX(f, D)$. Note that by Corollary 2.2, we can estimate θ to within a small multiplicative factor with probability at least $1 - \delta$ in time polynomial in θ^{-1} and $\log \delta^{-1}$. Given such a bound on θ, we can by Hoeffding (Lemma 2.2) compute the number m_1 of examples from a simulated oracle $EX(f, D^j)$ necessary to accurately and with high probability estimate μ'_j for j such that $|\mathbf{E}_{D^j}[f]| \geq \theta/8$. We can also compute a necessary number m_2 of examples from $EX(f,D)$ such that with high probability, for every j such that $\mathbf{Pr}_D[\chi_a = \omega_b^j] \geq \theta/8b$, at least m_1 of the examples $\langle x, f(x)\rangle$ will satisfy $\chi_a(x) = \omega_b^j$. It can be shown that a value of m_2 polynomial in b, θ^{-1}, and $\log\delta^{-1}$ is sufficient to produce sufficiently accurate estimates of μ'_j for all large j. □

8.2.3 Main Result

Putting this all together, we obtain the following:

Theorem 8.1 *For p any fixed polynomial, $t = p(n, s, \epsilon^{-1}, \log(\delta^{-1}))$, and $k = O(\log t/\log\log b)$, k-UBOX is learnable with respect to the uniform distribution using membership queries. The algorithm runs in time polynomial in b, n, s, ϵ^{-1}, and $\log(\delta^{-1})$.*

8.3 Learning UBOX

In fact, unrestricted UBOX is learnable in time polynomial in all parameters except b, and thus is efficiently learnable for b constant.

Corollary 8.1 *For p a particular fixed polynomial and $t = p(n, s, \epsilon^{-1}, \log(\delta^{-1}))$, UBOX is learnable with respect to the uniform distribution using membership queries in time polynomial in the parameters above as well as $t^{O(b\log\log b)}$. Here s represents the UBOX-size of the target function.*

Proof Sketch: We choose $t = 12s/c_1\epsilon^3$, where c_1 is the constant $1/57$ appearing in `HS`. Then for any k-rectangle R such that $k \geq \log_{b/b-1}(t)$, $\mathbf{Pr}[x \in R] \leq c_1\epsilon^3/12s$. Furthermore, for any distribution D simulated by `HS` in the process of boosting with respect to uniform, $\mathbf{Pr}_D[x \in R] \leq 1/4s$. Using this observation, the proof of Fact 8.1 can be generalized to show that, for *any* function f and for any D as above, there is some χ_a such that $|\mathbf{E}_D[f\chi_a]| = \Omega(s^{-1}t^{-O(b\log\log b)})$. □

8.4 Appendix

Here we briefly sketch the proof of a lemma needed for Fact 8.1.

Lemma 8.2 *For every* 1*-rectangle* $R : [b] \to \{0,1\}$, $L_1(\widehat{R}) = O(\log b)$.

Proof Sketch: Let d represent the width of R, that is, $d = |\{x \mid R(x) = 1\}|$. By the definition of the Fourier transform and standard facts about principle roots of unity, we can view each Fourier coefficient $\widehat{R}(a)$ as the vector sum of a sequence of d 2-dimensional vectors of length $1/b$, where there is a fixed angle $2\pi a/b$ between each consecutive pair of vectors. A geometric argument then gives that $|\widehat{R}(0)| \leq 1$ and that for $a \in \{1, \ldots, b-1\}$ the vectors all lie within a circle of diameter $1/b\sin(\pi a/b)$. That is,

$$|\widehat{R}(a)| \leq \frac{1}{b \sin \frac{\pi a}{b}}$$

for $a \in \{1, \ldots, b-1\}$, regardless of the choice of d. We can bound the sum of the magnitudes of the latter coefficients by considering a to be a continuous rather than discrete variable and integrating, which gives after simplification

$$\sum_{a=0}^{b-1} |\widehat{R}(a)| \leq 2 + \frac{2}{\pi}\ln(b).$$

$\square$

Chapter 9

Learning Over Nonuniform Distributions

We have thus far shown that DNF (as well as certain other classes of representations) is learnable with respect to the uniform distribution over the instance space. While the uniform distribution is in some sense quite natural, in many real-world application domains we might expect, for example, significant correlation among the variables in instances. Thus in this chapter we consider the problem of learning DNF with respect to distributions other than uniform.

9.1 Learning with respect to "Nearly" Uniform Distributions

Note that we have already laid the groundwork for a certain amount of generalization to nonuniform-distribution learning. Recall that the proof of Theorem 6.1 actually shows that DNF is learnable with respect to any distribution D such that an oracle for D is available and $L_\infty(2^n D)$ is bounded by a polynomial in n. Actually, we do not even require an oracle for such a D, because if we learn an $(\epsilon/L_\infty(2^n D))$-approximator h with respect to uniform then h is also an ϵ-approximator with respect to D.

Furthermore, we can relax the need for a polynomial bound on $L_\infty(2^n D)$. Recall that KM$'$ runs in time polynomial in, among other parameters, $L_\infty(g)$, where g is the (potentially non-Boolean) function for which a large Fourier coefficient is desired. The bound on the magnitude of g is used in $\mathtt{KM}'$ in two ways:

1. The number of samples needed to with high probability accurately estimate

$$\mathbf{E}_{x,y,z}[g(yx)g(zx)\chi_A(y \oplus z)] \tag{9.1}$$

   can be bounded, using Hoeffding's inequality (Lemma 2.2), by a function that is polynomial in $L_\infty(g)$ along with other parameters.

2. The number of recursive calls made by $\mathtt{KM}'$ is bounded by a polynomial in, among other parameters, $\mathbf{E}[g^2]$. This quantity in turn is bounded by $L^2_\infty(g)$.

A number of samples sufficient to accurately estimate the expectation in (9.1) can be computed using Chebyschev (Lemma 2.3) rather than Hoeffding. To apply Chebyschev, we need to bound the variance σ_p^2 of the function

$$p(x, y, z) = g(yx)g(zx)\chi_A(y \oplus z).$$

We can bound σ_p^2 in terms of the variance σ_g^2 of g, since

$$\sigma_p^2 = \mathbf{E}[p^2] - \mathbf{E}^2[p] \leq \mathbf{E}[p^2] = \mathbf{E}_x[\mathbf{E}_y^2[g^2(yx)]] \leq \mathbf{E}_{x,y}[g^4(yx)]$$

where the final inequality follows from the fact that for any real-valued function f, $\mathbf{E}^2[f] \leq \mathbf{E}[f^2]$. So $\mathbf{E}[g^4]$ upper bounds σ_p^2. Furthermore, $\mathbf{E}[g^4] + 1$ also upper bounds $\mathbf{E}[g^2]$. This means that in both items above a bound on $\mathbf{E}[g^4]$ can be substituted for the bound on $L_\infty(g)$. However, to achieve a confidence of $1-\delta$ that the estimated value of (9.1) is sufficiently close to the true value requires a number of examples polynomial in δ^{-1} rather than the $\log \delta^{-1}$ required when Hoeffding was applied.

Recalling that $\mathtt{HS}$ runs $\mathtt{KM}'$ on g's of the form $2^n f D$ where $f \in \{-1, +1\}$, this means that a modified version of the algorithm can efficiently learn DNF with respect to any distribution D for which it is given an oracle and such that $\mathbf{E}[(2^n D)^4]$ is polynomially-bounded in n.

9.2 Learning with respect to Product Distributions

We can also learn DNF with respect to certain product distributions. A *product distribution* on $\{0,1\}^n$ is defined by a set $\{\mu_i \in \mathbf{R} \mid 1 \leq i \leq n\}$. Each μ_i represents the probability that input bit i is a 1; this probability is independent of the values of all the other bits. Product distributions are a natural generalization of the uniform distribution, which is the product distribution having $\mu_i = \frac{1}{2}$ for all i. They are also distinct from the distribution classes considered above. For example, $\mathbf{E}[(2^n D_{1/4})^4] > 2^n$ for the product distribution $D_{1/4}$ in which $\mu_i = 1/4$ for all i.

Because of the independence between variables inherent in product distributions, standard Fourier analysis readily generalizes to these distributions [FJS91]. The primary difference is that the functions χ_A are replaced by a generalized Fourier basis consisting of functions φ_A, and Fourier coefficients are now defined with respect to this new basis. Specifically, let D be a product distribution with μ_i as above. Then for all $A \subseteq \{1, \ldots, n\}$ we define

$$\varphi_A(x) = \prod_{i \in A} \frac{\mu_i - x_i}{\sqrt{\mu_i(1-\mu_i)}}.$$

This φ basis is orthonormal with respect to the inner product $\langle f, g\rangle = \mathbf{E}_D[fg]$. Therefore, if for $f : \{0,1\}^n \to \mathbf{R}$ we define $\tilde{f}(A) = \mathbf{E}_D[f\varphi_A]$ then

$$f = \sum_A \tilde{f}(A)\varphi_A.$$

It is straightforward to verify that when D is the uniform distribution, $\varphi_A = \chi_A$ and $\tilde{f}(A) = \hat{f}(A)$ for all A.

Building on this generalized Fourier transform, Bellare has described a modified `KM` algorithm that finds all the large generalized Fourier coefficients of a function with respect to certain product distributions [Bel91]. We can use these ideas to extend `HS` and show the following.

Definition 9.1 *The* c-bounded product distribution family $\mathcal{PD}_c$ *is the set of all product distributions D such that for every μ_i defining D, $c \leq \mu_i \leq 1 - c$.*

Theorem 9.1 *For any constant $c \in (0, 1/2]$, DNF is learnable with respect to $\mathcal{PD}_c$ using membership queries.*

Proof Sketch: Let f be the target function and D the target product distribution. We will learn f with respect to D by running `HS` with a modified weak learner. Specifically, we will show that for every D_i defined by `F1` in the process of learning with respect to D we can efficiently and with high probability find a set A such that $|\mathbf{E}_{D_i}[f\varphi_A]|$ is inverse polynomially large. It then will follow by a table technique similar to the one outlined in the proof of Lemma 8.1 that φ_A can be converted to a Boolean weak approximator for f with respect to D_i.

The first step is to show that such a φ_A exists. By the proof of Fact 6.1 we know that for any s-term DNF f either $\mathbf{Pr}_D[f = 1] \leq 1/3$, in which case the constant function $\varphi_\emptyset$ is a weak approximator to f, or there exists a term T of f with the following properties:

- $\mathbf{Pr}_D[T \text{ is satisfied}] \geq \frac{1}{3s}$, and
- At least one subset A of the variables in T is such that $|\mathbf{E}_D[f\chi_A]| \geq \frac{1}{3s}$.

Furthermore, since D is a c-bounded product distribution, T contains at most $\ln(3s)/\ln(c^{-1})$ variables. Thus there is some parity χ_A over $O(\log(s))$ of the inputs which is a weak approximator to f with respect to D. Also, for the distributions D_i generated by `F1` in the process of learning against D, $D_i(x) = O(D(x)/\epsilon^3)$ for all x. So for each of these distributions D_i there is some parity over $O(\log(s/\epsilon))$ inputs which is a weak approximator with respect to D_i.

Now by reasoning similar to the proof of Fact 7.1, for A such that $|\mathbf{E}_{D_i}[f\chi_A]| \geq 1/(3s)$,

$$\frac{1}{3s} \leq \max_B |\mathbf{E}_{D_i}[f\varphi_B]| \cdot \sum_B |\tilde{\chi}_A(B)|.$$

Because χ_A is a Boolean function and Parseval's identity holds for any orthonormal basis, we have that $\sum_B \tilde{\chi}_A^2(B) = 1$. Furthermore, it can be shown that $\tilde{\chi}_A(B) = 0$ unless $B \subseteq A$. By our earlier bound on the size of A and the relationship between L_1 and L_2 norms, this gives

$$\max_B |\mathbf{E}_{D_i}[f\varphi_B]| \geq \left(\frac{\epsilon}{s}\right)^{O(1)}.$$

To find such a φ_B we use a modified `KM` algorithm. Define $C_{A,k} = \{\tilde{f}(A \cup E) \mid E \subseteq \{k+1, \ldots, n\}\}$. Then it can be shown that

$$L_2^2(C_{A,k}) = \mathbf{E}_{x,y,z}[f(yx)f(zx)\varphi_A(y)\varphi_A(z)].$$

For constant-bounded product distribution D and $|A| = O(\log(s/\epsilon))$, $|\varphi_A(y)| = (s/\epsilon)^{O(1)}$ for all y. Since we know (modulo our usual guess of s) that there is some A that obeys this bound and such that φ_A is well-correlated with f, we modify the `KM` algorithm so that it "cuts off" any branch of the recursion that would consider an A exceeding this bound. Thus at every recursive call to the modified `KM` we can efficiently estimate the above expectation for Boolean f, and therefore we can efficiently find a φ_A that correlates well with f with respect to D.

Furthermore, if we define $g(x) = f(x)D_i(x)/D(x)$ then $\mathbf{E}_{D_i}[f\varphi_B] = \mathbf{E}_D[g\varphi_B]$. For D_i as defined by `F1` when learning with respect to D we have $|g(x)| = O(\epsilon^{-3})$ for all x. Therefore, for each of these D_i's we can efficiently find a φ_B that correlates well with f with respect to D_i. □

This result holds whether or not the learner is given an oracle for the target distribution D, since a very good approximation to a product distribution D can be obtained efficiently by using random sampling to estimate the μ_i's defining D.

Chapter 10

Learning Despite Noise

So far, we have assumed that the membership oracle $MEM(f)$ is completely accurate. In this chapter we show that DNF and, more generally, TOP remain learnable (with very high probability) despite substantial noise in the data. As noted in the Introduction, proving that the Harmonic Sieve is noise tolerant is interesting because of the similarity of goals in noise-tolerant and real-world learning.

10.1 Definitions and Intuition

We focus on learning from a membership oracle that exhibits *persistent* classification noise [GKS93]. In particular, for any Boolean f and noise rate η define the noisy oracle $MEM^{\eta}_{CN}(f)$ as follows. If the oracle has been queried about the instance x previously, it answers with the same value returned before. Otherwise, with probability $1 - \eta$ the oracle returns $f(x)$, and with probability η it returns $-f(x)$. Given such a noisy membership oracle, the learning algorithm's goal is to find a hypothesis h that is an ϵ-approximation to the *true* target f, not to the noisy function simulated by the oracle.

Note that for $\eta = \frac{1}{2}$, on any input x, $MEM^{\eta}_{CN}(f)$ produces positive and negative outputs with equal probability, independent of f. Thus it is information-theoretically impossible to learn f in this case. Therefore, we will assume $\eta < \frac{1}{2}$.

The proof that TOP is learnable in this noise model is based on the following ideas. Since the noise is persistent and the decision about whether or not to apply noise to a given example is independent of the decisions for all other examples, we can assume that all noise decisions are made before learning commences. That is, we can treat the noisy oracle as if, given f, it chooses a fixed noisy function f^η according to the probability distribution D_f^η implied by the noise process described above. We will show that, with extremely high probability over the random choice of f^η, every Fourier coefficient $\widehat{f^\eta}(A)$ is very near its expected value $(1-2\eta)\hat{f}(A)$. Thus if we run the KM algorithm with a threshold of $\Omega(s^{-1}(1-2\eta))$ using the noisy oracle $MEM_{CN}^\eta(f)$, we are virtually certain to find a parity χ_A that is a weak approximator for TOP f. Therefore, the first stage of the HS algorithm will give essentially the same result whether we use $MEM(f)$ or $MEM_{CN}^\eta(f)$ (recall that the KM threshold is set automatically by WDNF). We then argue that a similar property holds for succeeding stages of the algorithm.

The only remaining difficulty is deciding when the algorithm should terminate, since it cannot determine that its hypothesis is within ϵ of the true target f by sampling f^η (we assume that the example oracle given to HS also draws from f^η). We will assume that a polynomial bound in n on the TOP-size of the target function is known.[1] This size bound assumption immediately gives us a bound on the number of boosting stages in HS (see Figure 6.2). Also, for sufficiently large n, a polynomial bound on the number of boosting stages will ensure (with high probability) that all of the weak hypotheses found by the algorithm are in fact weak approximators to the true function f. This follows from the strength of the bound given below on the deviation of the noisy Fourier coefficients from their expected values.

10.2 Formal Development

We begin the formal development of these ideas by bounding the probability that any $\widehat{f^\eta}(A)$ differs significantly from $(1-2\eta)\hat{f}(A)$. We use $\mathbf{Pr}_\eta[\cdot]$ and $\mathbf{E}_\eta[\cdot]$ to represent probability and expectation, respectively, taken over f^η chosen randomly according to D_f^η.

[1]In practice, as we will see in the next part of the thesis, there are other techniques for determining when the learning should terminate.

Lemma 10.1

$$\mathbf{Pr}_\eta[\exists A \text{ s.t. } |\widehat{f^\eta}(A) - (1-2\eta)\hat{f}(A)| > \lambda] \le 16e^{n-\lambda^2 2^{n-5}}.$$

Proof: First, consider any fixed A, and let $P_{i,j} = \mathbf{Pr}_x[f(x) = i \;\&\; \chi_A(x) = j]$ for $i,j \in \{-1,+1\}$. Then it follows from its definition that $\hat{f}(A) = \sum_{i,j} ijP_{i,j}$. Similarly, for fixed f^η define $P^\eta_{i,j,\ell} = \mathbf{Pr}_x[f(x) = i \;\&\; \chi_A(x) = j \;\&\; f^\eta(x) = \ell]$, and note that

$$\widehat{f^\eta}(A) = \sum_{i,j,\ell} j\ell P^\eta_{i,j,\ell}.$$

Let $E_{i,j,\ell}$ represent $\mathbf{E}_\eta[P^\eta_{i,j,\ell}]$. Taking expectations over random choice of f^η on both sides of the above equality, it follows that if for each i, j, and ℓ, $|P^\eta_{i,j,\ell} - E_{i,j,\ell}| \le \lambda/8$, then $|\widehat{f^\eta}(A) - (1-2\eta)\hat{f}(A)| \le \lambda$.

There is a simple relationship between each $E_{i,j,\ell}$ and one of the $P_{i,j}$'s; for example, $E_{1,1,1} = (1-\eta)P_{1,1}$. We will use this relationship to bound the probability that $P^\eta_{i,j,\ell}$ differs from its expected value by more than $\lambda/8$. To illustrate the process we consider a specific example. Let $S_{1,1} = |\{x \mid f(x) = 1 \;\&\; \chi_A(x) = 1\}|$ and $S^\eta_{1,1,1} = |\{x \mid f(x) = 1 \;\&\; \chi_A(x) = 1 \;\&\; f^\eta(x) = 1\}|$. That is, $P_{1,1} = S_{1,1}/2^n$ and $P^\eta_{1,1,1} = S^\eta_{1,1,1}/2^n$. Then

$$\mathbf{Pr}_\eta[|P^\eta_{i,j,\ell} - E_{i,j,\ell}| > \lambda/8] = \mathbf{Pr}_\eta\left[\left|\frac{S^\eta_{1,1,1}}{S_{1,1}} - (1-\eta)\right| > \frac{\lambda}{8P_{1,1}}\right].$$

We can bound the latter probability by Hoeffding. That is, we can conceptually associate a random variable with each of the $S_{1,1}$ x's such that $f(x) = \chi_A(x) = 1$. Each random variable takes on value 1 with probability $1-\eta$ and value 0 otherwise. $S^\eta_{1,1,1}$ then represents the sum of these random variables, which are independent and have the same mean. Therefore,

$$\mathbf{Pr}_\eta[|P^\eta_{i,j,\ell} - E_{i,j,\ell}| > \lambda/8] \le 2e^{-\lambda^2 S_{1,1}/32P^2_{1,1}} \le 2e^{-\lambda^2 2^{2n-5}/S_{1,1}} \le 2e^{-\lambda^2 2^{n-5}}.$$

The lemma follows by the fact that the probability of the union of events is bounded above by the sum of the probabilities of the events. □

We now state and prove the main theorem of this chapter. To simplify the statement of the theorem, we place tighter bounds on ϵ, δ, and η than are necessary.

Theorem 10.1 *For all sufficiently large n, TOP is learnable from a persistent classification noise membership oracle $MEM_{CN}^{\eta}(f)$ with respect to the uniform distribution, assuming the TOP-size s of f, ϵ^{-1}, $\log(\delta^{-1})$, and $1/(1-2\eta)$ are each bounded by a polynomial in n. The algorithm is told a polynomial in n upper bound on s. The probability of success is taken over the randomness of $MEM_{CN}^{\eta}(f)$ as well as over the internal randomization of the learning algorithm.*

Proof: Based on the earlier discussion and lemma as well as the given bounds on s, ϵ, δ, and η, for an appropriate choice of λ we can guarantee, with the degree of confidence required by `HS`, that the first stage of the noise-tolerant `HS` will find a weak approximator for the true target f. What remains to be shown is that we can obtain similar guarantees for the subsequent stages of the algorithm. We do this by generalizing Lemma 10.1.

Let D_i by the distribution simulated by the algorithm at stage i, and let $g(x) = 2^n f(x) D_i(x)$. Recalling the definition of D_i (equation (5.1)), it follows that $g(x)$ takes on at most $2k$ distinct values, where k is the number of boosting stages the algorithm will perform. Let G represent this set of values. Then for $i, j \in \{-1, +1\}$ and $\ell \in G$, if we define $P_{i,j,\ell}^{\eta} = \mathbf{Pr}_x[g(x) = i\ell \ \&\ \chi_A(x) = j \ \&\ g^{\eta}(x) = \ell]$ then $\mathbf{E}_{\eta}[\widehat{g^{\eta}}(A)] = \sum_{i,j,\ell} j\ell \mathbf{E}_{\eta}[P_{i,j,\ell}^{\eta}]$. The analysis of Lemma 10.1 applies to the expected difference between $P_{i,j,\ell}^{\eta}$ and $\mathbf{E}_{\eta}[P_{i,j,\ell}^{\eta}]$. Furthermore, $|\ell|$ is bounded by a polynomial in n given our bounds on ϵ, there are at most $8k$ terms in the sum above, and k is polynomially bounded because s and ϵ^{-1} are. Therefore, following the reasoning in Lemma 10.1, we can with very high probability guarantee sufficiently small deviation between $\widehat{g^{\eta}}(A)$ and its expected value, $(1-2\eta)\hat{g}(A)$, for every value of A. $\square$

Part II

Applications

Chapter 11

Introduction to Empirical Results

In the first part of the thesis we have developed a number of positive theoretical results in particular models of learning. The question we have not yet addressed is: are these theoretical results of any practical use? This second part of the thesis (which represents joint work with Mark Craven) addresses this question, and while we currently have results in only a limited number of domains, the results are encouraging.

We begin this chapter with a brief background on empirical machine learning research, an outline of the goals of our empirical research, and an introduction to three specific learning problems that will be addressed in various ways in subsequent chapters. We also present the `AdaBoost` hypothesis boosting algorithm [FS95] that is used in our empirical work.

11.1 Background

Early on in our discussion of theoretical machine learning we defined a number of formal models of what it means to learn. Similarly, in empirical machine learning we can define various frameworks within which learning can be studied. These frameworks, like the learning-theoretic ones, are differentiated largely by aspects such as the form of data that will be available to the learner, the goal of the learning algorithm, and how success will be measured. Some of the ways in which learning frameworks have been categorized include:

- Active vs. passive learning: Is the learning algorithm allowed to affect the selection of data that will be used to train the algorithm?

- Supervised vs. unsupervised learning: Is the algorithm told the class of a data item (such as what digit is represented by a given visual image), or is it simply given vectors of data and expected to find, say, a small hierarchical clustering of the data?

- Classification vs. skill (reinforcement) learning: Is the goal of the algorithm to correctly classify data, or is it to improve at some skill such as navigating through a maze?

We will be particularly interested in the passive, supervised, classification learning framework, which we now discuss in some detail.

Typically, empirical research in the classification-learning framework begins by assuming the existence of a set of examples (labeled instances) called the *training set*. An instance is often called a *feature vector*, the individual elements of the vector being *features* or *attributes*. Features are generally Boolean-valued, numeric (integer or rational), or *nominal-valued*. The latter feature is one that takes on a finite set of possible values; for example, an attribute that can have values `red`, `amber`, or `green` is nominal-valued. The label of an example is called the example's *class*.

The goal of the classification learner is to use the training set to produce a *classifier* that will accurately classify inputs in a *test set* that is unseen during learning. Classifier performance is generally measured as the (unweighted) percentage of the test set that is classified correctly by the output classifier. The performance of a learning algorithm on a task is obtained by averaging the performance of the classifiers it produces over a collection of training/test set pairs.

A common mechanism for producing the training/test pairs is *k-fold cross validation*. Specifically, one large set of examples is partitioned into k (approximately) equally sized sets. Each set is used once as a test set, with the data remaining each time used as the training set. We typically use 10-fold cross validation in the experiments described in this thesis. Cross validation can also be used within a training set. This is useful when there are parameters of the learning algorithm—such as the number of boosting stages performed by the Harmonic Sieve—that must be set carefully to obtain good performance. The idea is to treat the training set as if it was the full data set and use cross validation to compare the performance of different versions of the learning algorithm, each version having a different setting of the parameters. The best-performing version is then trained over the entire training set to produce the final hypothesis.

11.2 Goals

Perhaps the major difference between empirical and theoretical research on machine learning is in how success is measured. In theoretical research, success generally means that a theorem has been proved. Typically, the better known the question answered by the theorem, the more successful the result.

In empirical research on machine learning, success generally means discovering algorithms or techniques that, by some empirical measure, outperform known algorithms and techniques. The more distinct the advantages of the new algorithm, the more successful the result. In some sense an empirical algorithm is ultimately successful if it displays expert human-level performance on some learning task (and, if many algorithms achieve this level of performance, outperforms the competition). Machine learning algorithms have begun to achieve such a level of success in a variety of domains, including automatic vehicle steering [Pom91], handwritten character recognition [DSS93], and game playing [Tes94].

As indicated above, the typical performance measure for empirical machine learning algorithms is: how well did the algorithm learn the task at hand? But there are a variety of other useful measures of performance, including for example: how much time was required for learning; how quickly can the learned function be computed; how much memory is required to learn/store the learned function; how well does the learning algorithm use prior knowledge; how intelligible is the learned function? Of course, a performance measure may be some composite of these simpler measures.

Broadly speaking, the goal of the empirical research reported in this thesis is to develop algorithms designed to perform well against a composite measure of performance: learn as well as the best-known machine-based techniques while producing a learned function that is more intelligible than other techniques with comparable classification performance.

We approach this task using two different techniques. In Chapter 12, we try a two-step method: first, learn using the best-known classification algorithm, producing classifier c; then attempt to produce an approximation to c which is more intelligible than c. This assumes, of course, that the best-known classifier c is relatively unintelligible, which is the case in many application domains. The advantage of this two-step approach is that we can use c as a membership oracle, which (theoretically) allows us to apply the Harmonic Sieve to the task of approximating c by a more intelligible classifier.

There are, however, shortcomings to applying this method in many real-world domains. One difficulty is that `HS` approximates with respect to uniform, which may not give an appropriate approximation with respect to the underlying target distribution. Another difficulty is that the classifier produced by `HS` will be an approximation to an approximation to the target function, and thus may not be a particularly good approximator. Finally, while `HS` is a polynomial-time algorithm, the version of the algorithm presented in Chapter 6 does not scale as well as we would like. To overcome these difficulties we develop two modified versions of `HS`. Each of these versions is used to perform rule extraction in a different benchmark domain, with reasonably successful results.

The other approach we take to achieving our empirical goal is to attempt to learn relatively intelligible classifiers directly from the training set (Chapter 13). In this case we do not have a membership oracle available and therefore cannot use the Harmonic Sieve directly. However, as we will see, one of the modified versions of `HS` developed in Chapter 12 actually does not require membership queries. On three benchmark tasks, including the two on which rule extraction was performed, this Boosting-based Perceptron learning algorithm not only produces quite intelligible classifiers, but also produces among the most accurate classifiers known for these tasks. Furthermore, on one of the tasks the algorithm demonstrates a particularly strong ability for selecting, from a very large set of potential features, those features that are thought to be among the most relevant for that classification task.

We next outline the three benchmark tasks considered.

11.3 Application Domains

11.3.1 Congressional Voting

Schlimmer [SF86] extracted the votes of the 435 members of the US House of Representatives from the Congressional Quarterly Almanac (1985) on 16 issues. The features of each record in the data set are one representative's votes on each of the issues (Y, N, or ?), and the classification of a record is that member's party affiliation (R or D). Some of the votes of members of the opposing parties are almost identical; therefore, we do not expect perfect separation of the data by a general-purpose learning algorithm. In fact, it is just this "messiness" of the data rather than any real-world usefulness that has made this an interesting benchmark data set. The relatively small size of the data set has also led a number of empirical learning researchers to include it in their studies. The version of the data set we use has one of the features (`physician-fee-freeze`) removed; this has been found to make the learning task noticeably more difficult [BN92].

11.3.2 Promoter Recognition

A *promoter* is a relatively short segment of DNA that immediately precedes a gene, which is a blueprint for a functional unit (typically a protein) within an organism. A promoter is in effect a marker which RNA polymerase, the molecular machine used for gene transcription, recognizes. However, the "rules" used by RNA polymerase to recognize promoters are not completely understood, hence for humans to detect the presence or absence of a gene within a strand of DNA is generally a somewhat time-consuming process. Thus a classifier for the promoter recognition problem could be quite valuable to biologists.

The specific promoter set we use is a more complex extension (468 records) of the data set first assembled and analyzed by Towell *et al.* [TSN90]. In these data sets, an instance of the promoter recognition problem is a sequence of 57 nucleic bases (each of the bases is represented by one of the characters "a, c, g, t"). The correct classification of an instance is "promoter" if a gene begins in the 51st position and is "nonpromoter" otherwise. An actual example for the promoter recognition task follows.

```
t t t g t t t t t c a t t g t t g
a c a c a c c t c t g g t c a t g
a t a g t a t c a a t a t t c a
t g c a g t a promoter
```

Exactly half of the records in our data base are promoters.

11.3.3 Protein-coding Region Recognition

The protein-coding region recognition task is similar to promoter recognition. In particular, the input is again a string of characters representing a DNA sequence. Now, however, we are specifically interested in structural genes, those genes that provide a blueprint for protein synthesis. Proteins are complex macro-molecules composed of a sequence of simpler molecules, amino acids. A structural gene contains a sequence of nucleotides that codes for the sequence of amino acids in the corresponding protein.

As noted above, there are only four different nucleotides. On the other hand, there are 20 different amino acids used to construct a typical protein. Therefore, there is not a one-to-one correspondence between the nucleotide sequence in a structural gene and the amino acid sequence in the protein encoded by the gene. Instead, each amino acid is coded for by a sequence of three nucleotides.

If we are given a DNA sequence s that is a portion of the sequence coding for a protein, and if the first character of s is also the first character of a triple coding for an amino acid, then s is an *in-frame protein-coding region.* The protein-coding region recognition task is to tell whether or not a given DNA character string represents an in-frame protein-coding region. We use the protein-region data set studied previously by Craven and Shavlik [CS93b], which consists of 10,000 sequences, each 15 nucleotides (5 triples) long. Roughly half of the sequences are examples of in-frame protein-coding regions.

11.4 The `AdaBoost` Hypothesis Boosting Algorithm

AdaBoost
Input: training set S of m examples of function f, weak learning algorithm `WL` that is $(\frac{1}{2}-\gamma)$-approximate, γ
Algorithm:
1. $T \leftarrow \frac{1}{2\gamma^2}\ln(m)$
2. for all $x \in S$, $w(x) \leftarrow 1/m$
3. **for** $i = 1$ to T **do**
4. for all $x \in S$, $D_i(x) \leftarrow w(x)/\sum_{j=1}^{m} w(x)$.
5. invoke `WL` on training set S and distribution D_i, producing weak hypothesis h_i
6. $\epsilon_i \leftarrow \sum_{x.h_i(x)\neq f(x)} D_i(x)$
7. $\beta_i \leftarrow \epsilon_i/(1-\epsilon_i)$
8. for all $x \in S$, if $h(x) = f(x)$ then $w(x) \leftarrow w(x)\cdot\beta_i$
9. **enddo**

Output: $h(x) \equiv sign\left(\sum_{i=1}^{T} -\ln(\beta_i)\cdot h_i(x)\right)$

Figure 11.1: The `AdaBoost` boosting algorithm for producing a consistent hypothesis using a Boolean weak learner.

As mentioned earlier in this chapter, in our empirical work we will be particularly interested in learning within the passive classification learning framework. In Chapter 5 we noted that this framework lends itself to a boosting-by-sampling approach rather than the boosting-by-filtering approach employed by `F1`. Due to this and other practical considerations discussed in Chapter 5, we employ a different boosting mechanism in our applied versions of the Harmonic Sieve.

The specific boosting algorithm we use in our empirical studies is a version of the recent `AdaBoost` algorithm [FS95] (Figure 11.1). Given a weak PAC learner that can achieve accuracy $\epsilon = \frac{1}{2}-\gamma$, `AdaBoost` runs for $T = \ln(m)/(2\gamma^2)$ stages. Initially, the algorithm creates a distribution D_1 that is uniform over the given example set S. The probability distribution at each subsequent stage i is defined in a rather simple way: if h_{i-1} correctly classifies an example, then the weight on that example is reduced in the distribution D_i used at stage i. The effect of this is to increase the focus of the weak learner at stage i on those examples which h_{i-1} classified incorrectly. At the end of the T stages a final hypothesis h is output; this is just a weighted threshold over the weak hypotheses $\{h_i \mid 1 \leq i \leq T\}$. Note that, unlike `F1`, the weights are now real-valued. If the weak learner succeeds in producing a

$(\frac{1}{2} - \gamma)$-approximator at each stage then `AdaBoost`'s final hypothesis will be consistent with the training set [FS95]. As noted in Chapter 5, for large enough training sets a consistent hypothesis will generally be a good approximator to the target.

In practice, of course, there will be no guarantee that the weak learner to be boosted will achieve any particular level of approximation. In fact, real-world data may not even describe a function, *i.e.*, the same feature vector may appear more than once in the data set and be assigned different labels at different times. Thus while the theoretical guarantees for `AdaBoost` give us some confidence that the algorithm has potential real-world applications, we will also need to modify the algorithm somewhat for practical use. Specifically, we will choose the stopping criteria in ways other than the method shown in Figure 11.1. We will mention the specific methods used as we describe each of the experiments.

Chapter 12

Rule Extraction

In this chapter we define the *rule extraction* problem and present potential solutions to this problem based on variants of the Harmonic Sieve algorithm developed earlier in the thesis. We begin with some motivation for the rule extraction problem, then consider two benchmark problems and two Harmonic-Sieve-based solutions.

12.1 Motivation

Many problems in science, and particularly in biology, fit into the supervised classification framework of learning discussed in the previous chapter. For example, consider the promoter recognition problem described in the last chapter. We are given a set of known examples of DNA sequences that represent promoter regions as well as regions that do not represent promoters and we are asked to develop a classifier that will correctly classifier future DNA sequences.

Some of the best results for biologically-motivated classification problems such as promoter recognition have been obtained by classification algorithms that produce multilayer perceptrons (MLP's; also sometimes called artificial neural networks) as the classifier. A (single-layer) *perceptron*, as defined by Minsky and Papert [MP88], is simply a weighted threshold over the set of input features and over local functions of these features (Figure 1.4). By "local" we will mean functions that take only a limited number of the features as inputs. A two-layer perceptron (Figure 1.5) is a weighted threshold of perceptrons (which are typically linear, *i.e.*, without local functions as inputs), and multilayer perceptrons generalize this idea in the obvious way. In the neural network flavor of MLP's there are many variations on this simple framework, such as replacing the threshold function used in our definition with a similar but differentiable function, or allowing outputs from one level to feed back to previous levels. In the empirical comparisons to follow we will focus on MLP's that are strictly feedforward, have a hard threshold at the final level (so that a Boolean output is produced), have sigmoidal internal functions, and are trained by a version of the backpropagation learning algorithm [RHW86].

While, as already noted, MLP learning algorithms have proved to be particularly effective at solving biological (and other) classification problems, they are not a perfect solution to these problems. In particular, an MLP solution to a biological classification problem typically sheds little light on what mechanisms underly the physical phenomenon being observed, because MLP's are frequently very difficult to understand.

As one example of the causes of this difficulty, consider that often an MLP is described by a set of hundreds or even thousands of real-valued parameters. Also, while humans tend to find linear functions relatively intelligible, MLP's often represent highly nonlinear functions. Finally, although internal (*hidden*) functional units in multi-layer networks are linear and therefore potentially understandable on an individual basis if not collectively, it is often difficult to ascribe intelligible meaning to individual hidden units. This is because MLP's commonly learn distributed representations, that is, each unit has a meaning only in the context of the values produced by many other units. Thus in an MLP for the Congressional voting problem, there may not be any one unit that represents "conservative," although the concept may be present in the MLP in the sense that inputs representing conservative members of Congress might generate a characteristic set of responses in the various hidden units.

In short, while an MLP may do an excellent job of telling us, say, that a specific DNA sequence represents a promoter region, it will generally not provide us with a general rule saying, for example, "If the 22nd nucleotide in the sequence is 'a' and the 23rd is 'g' then the sequence likely represents a promoter." However, such rules can be extremely useful for several reasons. First, to the extent that the rules match reasonably well with expectations, the rules can add to the confidence we are willing to place in the output of a classifier. Second, unexpected rules may play a role in the scientific discovery process. Finally, a simpler classifier may have useful computational advantages over a more complex rule.

In part because MLP's do not in general represent intelligible rules, several researchers in biology have turned to alternative classification methods that do produce human-intelligible rules, such as decision tree learners [HK93] or inductive logic programming [SKLM94]. In some problem domains these more intelligible classification strategies produce classifiers having essentially the same accuracy as more opaque approaches. However, there are classification problems in biology where MLP learners seem to have a distinct advantage over competing recognition methodologies [CS93a, TS94]. It is specifically this type of problem that motivated our study.

Thus we consider a different approach to rectifying the intelligibility deficiency of MLP's, an approach that relies on a form of *rule-extraction.*[1] Our approach is quite simple: rather than attempting to completely replace the MLP learner with another learning algorithm producing more intelligible rules, we will use the MLP learner as the first step of a two-step learning process. That is, we begin by training an MLP on the given training set. Our second step is to run a second learning algorithm that, rather than attempting to learn from the original training set, attempts to approximate the function represented by the MLP produced in the first stage. The hope is that this second algorithm (the rule extractor) can produce a classifier that is both more intelligible than the MLP and nearly as accurate on the test set.

At this point, a reasonable question is, "If there is an algorithm that can produce a good classifier given an MLP, why not just skip producing the MLP and run this algorithm on the original training set directly?" The answer is that the algorithm we plan to use in the

[1] Actually, the rule-extraction methodology we outline is applicable to any classifier, not merely MLP's. We focus here on MLP's because they are notoriously hard to understand and to the best of our knowledge were the most accurate general-purpose learning methods on the benchmarks we consider.

second stage is a version of the Harmonic Sieve, which is a *membership-query* algorithm. Thus `HS` would not be an appropriate algorithm to run when the only data available about the target function is a relatively small set of training examples. On the other hand, once an MLP has been trained to approximate the target function, this MLP can be used as a surrogate membership oracle for the target. Therefore, if the MLP is a good approximation to the target, and if `HS` is capable of producing an accurate approximation h to the MLP, and if h is more intelligible than the MLP, then our approach will be successful.

Of course, there are several "if"s in statement above, and we might therefore wonder if there is any domain in which this approach can succeed. As we will show presently, in at least one (relatively simple) domain our approach does succeed at producing classifiers that are both as accurate as the MLP's produced and generally much more intelligible. In a second domain having a much larger feature set we again produce relatively intelligible classifiers, but they are slightly less accurate than the MLP's on which they are based. We address this benchmark again in the next chapter using a different approach with better results.

Before turning to our empirical results on the rule extraction problem, it should be mentioned that the Harmonic Sieve is certainly not the only technique that could be used to extract more rule-like classifiers from MLP's. In fact, there has been much interest in the problem of simplifying MLP's, *e.g.*, [ADTss, CS94, HB90, LDS89, Thr93]. A potential advantage of the Harmonic Sieve is that it incorporates powerful algorithmic techniques, such as Fourier methods and hypothesis boosting, that to the best of our knowledge have not been applied to the rule extraction task before. The algorithm is also (at least theoretically) capable of learning a very expressive class of Boolean functions, and is tolerant of certain forms of noise. These attributes led us to apply versions of the Harmonic Sieve to the two benchmark rule extraction tasks discussed in the next section. Before considering the specific tasks, we examine some of the potential difficulties with using the Harmonic Sieve for practical rule extraction.

12.2 Applying HS to Rule Extraction

In practice, it is not at all clear *a priori* that HS is well suited to the rule extraction task. First, many people would not consider an arbitrary TOP representation—which can include numerous parity functions each over a large subset of the inputs—particularly intelligible. Therefore, even if HS succeeds at producing a good approximation to the given MLP representation, it may be that we will have made no progress on extracting a human-intelligible rule. And in fact there is no guarantee that HS will in a reasonable time produce a good approximator to an arbitrary polynomial-size MLP, as discussed in Chapter 7.

Furthermore, HS produces a hypothesis (classifier) that generalizes well *with respect to the uniform distribution*. On the other hand, what we want is a classifier that generalizes well over some unknown, possibly very nonuniform, real-world distribution. Thus, theoretical guarantees aside, it is quite possible that the classifier produced by HS will not generalize well on the test set even if it does approximate the given MLP well with respect to uniform.

Finally, many (if not most) real-world problems do not have exclusively Boolean-valued features. While we showed in Chapter 8 that the Harmonic Sieve can be generalized for certain non-Boolean domains, the general form of the hypothesis becomes even less understandable than a TOP! Of course, it is generally possible to replace an attribute with a set of Boolean attributes; for example, in the voting-data domain, each three-valued voting data feature can be replaced by two Boolean features, say one that is on exactly when the vote is yes and the other on exactly when the vote is no. But such a change expands the number of attributes n, and while HS is a polynomial-time algorithm, in practice its original version is not computationally feasible on typical current workstations for values of n that are, say, much greater than 25 (although replacing KM with the newer Levin algorithm mentioned in Chapter 4 may improve this).

We now consider how we addressed these difficulties in applying HS to two domains: Congressional voting and promoter recognition. A different approach is used in each domain.

12.2.1 Congressional Voting

As noted earlier, the voting benchmark is useful because of its relative difficulty (we are aware of no learning algorithm that achieves even 95% accuracy on the 15-feature version of this task) combined with the small size of the data set (435 records of 15 3-valued features each). Each three-valued feature was expanded into two Boolean-valued features as described above. Using 10-fold cross validation we trained 10 different MLP's, each on 90% of the original data set. MLP's with 0, 5, 10, 20, and 40 hidden units were trained and 20-fold cross validation within each training set was used to select the "best" architecture for each of the 10 training sets. In three of the 10 sets perceptrons (0 hidden units) were selected, and 10 or more units were selected for the remaining sets.

Next, for each MLP a truth table was produced corresponding to the output of the MLP on each possible input of 15 yes/no votes (*i.e.*, "?" votes were not allowed). That is, a Boolean function over 15 Boolean-valued features was extracted from each MLP. We then ran a version of `HS` using the truth tables for these functions as input. This version of `HS` used a Fast Fourier Transform (see Chapter 4) rather than `KM` to find the best-correlated parity function at each stage; because of the small number of inputs, this ran relatively quickly on a Sun SPARC1 workstation. As mentioned previously, `AdaBoost` was used as the boosting algorithm. Initially, the TOP's produced were somewhat difficult to understand because some of the parity functions included in a TOP could involve four or more features. In an attempt to produce simpler but still reasonably accurate TOP classifiers, we introduced a "hack" into the process of selecting a parity as the weak hypothesis at each stage. Rather than selecting the best-correlated parity, each parity was assigned a value equal to its correlation divided by 2^{k-1}, where k was 1 for parities on 0 or 1 variables and the number of variables in the parity otherwise. The parity maximizing this value was then selected as the weak hypothesis at a stage. Thus this version of `HS` considered all possible parity functions at each stage, but would select a parity over a large number of features only if that parity was an overwhelmingly better approximator than all the parity functions over few features. Finally, the number of boosting stages was set at 250 because this value allowed reasonable running times and produced simple classifiers that approximated the MLP's extremely well with respect to uniform. This number was chosen without reference to the training or test sets, but only based on the performance of HS on the truth table it was given as input.

The results of this experiment were quite successful. In terms of intelligibility, eight of the TOP's produced were simple linear perceptrons, one had a single two-feature parity, and the remaining TOP contained two second-order terms. As argued in the next chapter, there is reason to believe that such classifiers are reasonably intelligible. The correlation of the classifiers with the target functions was also excellent: all classifiers were at least 99.4% accurate over the 2^{15} values of their respective targets. When the classifiers were applied to the 10 sets of training and test data (with "?" votes represented as the probability of a "Y" vote on the issue) the performance was even better: 99.8% agreement with the MLP's on the training data (that is, eight disagreements over 10 trials), and 100% agreement on the test data. In fact, the TOP's actually (slightly) outperformed the MLP's on the training data (93.1% vs. 92.9%), even though the MLP's were given the training data and the TOP's were not!

12.2.2 Promoter Recognition

Applying HS to promoter recognition was a more difficult task. In this task there are 57 4-valued nominal features which were expanded to 228 Boolean-valued features in the obvious way. Although the Harmonic Sieve runs in time polynomial in the number of features, it was not feasible to run the current version of the algorithm on such a large problem using available hardware.

As indicated earlier, the KM algorithm is particularly slow in practice, and we could not replace it with an FFT in this task because of the much larger number of features. Therefore, we experimented with replacing KM by various heuristic simplifications. In particular, recall that HS uses KM to find the parity function that correlates best with the target function with respect to a particular distribution. While in the general case there are 2^n possible parity functions to consider (where n is the number of Boolean features), as indicated in the preceding section we are particularly interested in finding "small" parities that correlate well with the target, as this produces a more intelligible classifier. Specifically, if we consider only parities over at most k variables, then there are only $O(n^k)$ such parities. Therefore, for small values of k it becomes computationally feasible to examine the correlation of all possible parity functions of interest. Of course, if the target function is well approximated by a parity over $k+1$ features, this approach will fail to find this parity. The empirical question that arises, then, is how large k needs to be in order to have good performance on a variety of real-world problems.

We applied this approach with $k = 1$ to learning an approximator to an MLP in the promoter domain. That is, our weak hypothesis at each stage was either a single feature or a constant ($+1$ or -1). As the final hypothesis of `HS` is a threshold over the weak hypotheses, this means we were producing a linear threshold as the classifier, but using something very different than traditional algorithms to find this linear threshold. To measure the correlation of each possible weak hypothesis with the MLP, we used a fixed set of 5000 examples of the MLP function drawn uniformly at random when the algorithm commenced execution and measured all correlations over this set. We defined the target distribution for this modified version of `HS` to be uniform over the set and ran the algorithm for 250 boosting stages.

Using 10-fold cross validation and MLP training as described for the voting task, we found that our algorithm gave 90.0% accuracy on the test data while the MLP's gave 91.0%. Our perceptrons generally had approximately 150 nonzero weights (recall that there were 228 features in this data set); four of the MLP's were perceptrons, one had five hidden units, and the remaining five MLP's had 10 or more hidden units. The largest of these (40 units) had over 9000 weights.

It is worth noting again that our perceptrons were being trained on data randomly drawn from the MLP's and not on the actual training data. Thus it is particularly interesting that in spite of this the perceptrons agreed with the MLP's on the training data 98.0% of the time and classified the training data with accuracy 96.6%. This despite the fact that none of the perceptrons achieved even 96% accuracy over the 5000 random examples on which they were actually trained.

Overall, this experiment was reasonably successful, although we were not able to achieve MLP accuracy. In order to boost the accuracy we considered learning with larger values of k. But notice also that the version of `HS` that we used above uses membership queries only to produce a random sample on which to run the algorithm. This raises an obvious question: what happens if rather than drawing a random sample from the MLP, we run a version of this algorithm directly on the "random sample" represented by the training set? This is the subject of the next chapter.

Chapter 13

Learning Sparse Perceptrons

In the previous chapter, we considered a two-stage procedure for producing intelligible yet accurate classifiers: learn a difficult-to-understand classifier such as an MLP and then try to transform the MLP into a more understandable yet still accurate classifier. One obvious difficulty with this approach is that we expect in general that the resulting classifier, while perhaps achieving nearly the performance of the MLP, is likely to be somewhat less accurate. We observed exactly this difficulty at the end of the last chapter. Intuitively, it would seem preferable to produce an intelligible approximator directly from the training data; it seems at least possible that this approach might even lead to the best of both worlds, better intelligibility *and* better generalization performance.

In this chapter we present a one-stage approach to learning understandable, yet accurate, classifiers. This approach builds on the second algorithm of the preceding chapter, which can be viewed as a stripped-down version of the Harmonic Sieve. Specifically, the algorithm gives up any hope of finding large parities (*i.e.*, parity functions over large subsets of the features) that correlate well with the target function. In the applications we consider, we are able to achieve good classification performance despite this change. In return for this change, we obtain an algorithm that does not require membership queries and that therefore can be run directly on the training data.

Specifically, this Boosting-based Perceptron (`BBP`) algorithm is designed to output *sparse perceptrons*, *i.e.*, single-layer (nonlinear) perceptrons that have relatively few non-zero weights. The nonlinear inputs to the threshold are now small conjunctions rather than the arbitrary-size parity functions of a TOP; conjunctions were chosen for intelligibility reasons.

We begin this chapter by formally defining sparse perceptrons and stating some theoretical results that follow readily from our earlier discussions. It should be noted that Freund has previously made similar observations [Fre93]. After this theoretical development we discuss the application of the `BBP` algorithm to three benchmark problems.

13.1 Theory of Sparse Perceptron Learning

For our later theoretical results we will need a more precise definition of sparseness which we develop now. Consider the Boolean function $f : \{0,1\}^n \to \{-1,+1\}$. Let C_k be the set of all conjunctions of at most k of the inputs to f. C_k includes the "conjunction" of 0 inputs, which we take as the identically 1 function, and for $k \geq 1$, includes "conjunctions" of 1 input, *i.e.*, the individual input features. All of the functions in C_k map to $\{-1,+1\}$, and every conjunction in C_k occurs in both a positive sense (+1 represents `true`) and a negated sense (-1 represents `true`). Then the function f is a *k-perceptron* if there is some integer s such that

$$f(x) = \operatorname{sign}\left(\sum_{i=1}^{s} h_i(x)\right) \tag{13.1}$$

where for all i, $h_i \in C_k$, and the sign function is defined as

$$\operatorname{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Note that while we have not explicitly shown any weights in our definition of a k-perceptron f, integer weights are implicitly present in that we allow a particular $h_i \in C_k$ to appear more than once in the sum defining f.

We call a given collection of s conjunctions $h_i \in C_k$ a *k-perceptron representation* and s the *size* of the representation. As usual, we define the *size* of a given k-perceptron function f as the minimal size of any k-perceptron representation of f. By an *s-sparse* k-perceptron we mean a k-perceptron f such that the size of f is at most s.

We now develop the `BBP` algorithm for PAC learning sparse perceptrons. First, we note some well-known facts about perceptrons. Let $\mathcal{H}_r$ represent the class of halfspaces in r

dimensions, that is, the class of functions representable as the sign of a real-weighted sum of at most r inputs. Then for any $f \in \mathcal{H}_r$ and any positive ϵ and δ, given a set S of m examples drawn from $EX(f, D)$, where

$$m = \Omega\left(\frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{r}{\epsilon}\right),$$

then

$$\mathbf{Pr}[\exists h \in \mathcal{H}_r \mid \forall x \in S\ f(x) = h(x)\ \&\ \mathbf{Pr}_D[f(x) \neq h(x)] > \epsilon] < \delta,$$

where the outer probability is over the random choices made by $EX(f, D)$ [BEHW89]. In other words, given enough (polynomial in the parameters describing the difficulty of the learning problem) training data, any hypothesis $h \in \mathcal{H}_r$ that is completely consistent with the target function $f \in \mathcal{H}_r$ on the training data is likely a strong approximator to f with respect to D.

While there are existing PAC algorithms for PAC learning the general class of perceptrons (*e.g.* [BEHW89]), an atypical aspect of our algorithm is that when learning target functions which are in the class of sparse perceptrons, it produces relatively sparse hypotheses as output. In particular, if the target is an s-sparse k-perceptron, then the `BBP` algorithm produces hypotheses in $\mathcal{H}_r$, where (ignoring logarithmic factors) r is quadratic in s and the inputs to the hypothesis are a subset of the elements of C_k. By the above, to show that s-sparse k-perceptrons are PAC learnable using such hypotheses it is sufficient to show that given any set of training data for such a perceptron we can efficiently find a suitable hypothesis consistent with the data. Therefore, in the sequel we will assume that we are given a sufficiently large training set and that our learning task is to find a sparse perceptron consistent with that data set.

In fact, by Freund's boosting results, to show that sparse k-perceptrons are PAC learnable as sparse perceptrons we need only show that k-perceptrons are weakly learnable as conjunctions in C_k. This follows quite easily from the following [GHR92], which generalizes our Fact 7.1:

Lemma 13.1 (Goldmann Hastad Razborov) *For $f : \{0,1\}^n \to \{-1,+1\}$ and H any* *set of functions with the same domain and range, if f can be represented as*

$$f(x) = \text{sign}\left(\sum_{i=1}^{s} h_i(x)\right)$$

where $h_i \in H$, then for any probability distribution D over $\{0,1\}^n$ there is some h_i such that

$$\mathbf{Pr}_D[f(x) \neq h_i(x)] \leq \frac{1}{2} - \frac{1}{2s}.$$

If we specialize this lemma by taking $H = C_k$ then this implies that for any k-perceptron function f of size s and any probability distribution D over the input features of f there is some $h_i \in C_k$ that $(\frac{1}{2} - \frac{1}{2s})$-approximates f with respect to D. Therefore, given such a distribution D the weak learning algorithm is extremely simple: exhaustively test each of the $O(n^k)$ possible conjunctions of at most k features until a weak approximating conjunction is found. The above lemma proves that if f is a k-perceptron then this exhaustive search must succeed at finding a weak approximator that has $\gamma \geq 1/2s$. Therefore, given a sufficiently large (polynomial size) training set with m examples, a booster such as `F1` need only run for $2s^2 \ln(4m)$ stages before it will produce an ϵ-approximator to f. Because each stage adds one weak hypothesis (conjunction) to the output hypothesis, the final hypothesis will be in $\mathcal{H}_{2s^2 \ln(4m)}$, where the inputs to the hypothesis are in C_k.

So far we have shown that sparse k-perceptrons are learnable by (polynomially larger) sparse perceptron hypotheses. In practice, of course, we expect that many real-world classification tasks cannot be performed exactly by sparse perceptrons. However, it follows from the results of Chapter 10 that our algorithm is robust in the sense that even if the target function is only a "noisy" version of a sparse k-perceptron, the `BBP` algorithm is still capable of learning a sparse representation of the underlying perceptron, at least for certain definitions of noise.

Finally, if the `F1` booster is used, the algorithm will produce a sparse perceptron with integer weights, which is perhaps interesting from a theoretical point of view. However, as discussed in Chapter 5, in practice we prefer to use `AdaBoost` [FS95], which produces real-valued weights.

13.2 Implementation Considerations

We have just shown that, for a specific (theoretical) definition of what it means to "learn", sparse perceptrons are learnable. We turn now to a few details of the practical implementation of this theory.

The major implementation concern involves deciding when the learning algorithm should terminate. `AdaBoost` assumes that a lower bound on the accuracy of the weak learner is known when calculating the number of boosting stages. Of course, such information is not available in real-world applications, and in fact, if the target function is not a k-perceptron then conceivably this lower bound is extremely small and therefore would not be useful for computing the number of boosting stages. In practice, we use cross validation to determine an appropriate termination point, as discussed in Chapter 11. We also limit the number of boosting stages to at most n, where n is the number of weights in an ordinary single-layer perceptron for a given task. This is consistent with our desire to have relatively sparse perceptrons and was found to be helpful in practice.

Another implementation detail has to do with attempting to keep the output hypothesis as understandable as possible. Specifically, we would like to favor smaller conjunctions in the perceptrons we produce. To accomplish this, we modify our algorithm in a way similar to that used in the previous chapter. Specifically, for $j > 1$ we multiply each j-order feature's correlation by $\frac{1}{j}$ before selecting the feature that serves as the weak hypothesis at each boosting stage. This policy is related to Rissanen's Minimum Description Length principle [Ris89] in that we are trading off possible performance enhancement (at least in the weak hypothesis) for reduction in the complexity of the final hypothesis.

13.3 Applications of Sparse Perceptron Learning

We now apply the `BBP` algorithm that results from the previous considerations to three benchmark tasks. First, it should be noted that the theory developed above works over *discrete* input domains (Boolean or nominal-valued features). Thus, we consider only tasks with discrete input features. Also, because the algorithm uses exhaustive search over all conjunctions of size at most k, learning time depends exponentially on the choice of k. In this study we chose to use $k = 2$ throughout, since this choice results in reasonable learning times and apparently good performance on the tasks we consider.

In our experiments, we are interested in assessing the generalization ability and the complexity of the hypotheses produced by our algorithm. We compare our algorithm to ordinary perceptrons trained using backpropagation [RHW86], multi-layer perceptrons trained using backpropagation, and decision trees induced using the C4.5 decision-tree system [Qui93]. We use C4.5 in our experiments as a representative of symbolic learning algorithms. The hypotheses produced by symbolic algorithms are generally considered to be more comprehensible than MLP's. Additionally, to test the hypothesis that the performance of our algorithm can be explained solely by its use of second-order features, we train ordinary perceptrons using feature sets that includes all pairwise conjunctions, as well as the ordinary features. To test the hypothesis that the performance of our algorithm can be explained by its use of relatively few weights, we consider ordinary perceptrons which have been pruned using the Optimal Brain Damage (OBD) algorithm [LDS89]. In our version of OBD, we train a perceptron until the stopping criteria are met, prune the weight with the smallest salience, and then iterate the process. Cross validation is used to decide when to stop pruning weights.

We evaluate our algorithm using the three real-world domains described in Chapter 11: Congressional voting [SF86], recognition of promoters in DNA [TSN90], and recognition of protein-coding regions in DNA [CS93b]. For the voting and promoter domains, we randomly partition the data into 10 training and test sets. Because of certain domain-specific characteristics of the data, we use only four training and test sets for the protein-coding task. All of our reported results represent averages over the different training/test sets.

For each training set, cross validation is used to set various parameters for each algorithm: the number of hidden units and training epochs for multi-layer perceptrons; the pruning confidence level for C4.5; the number of weights to delete for OBD; and the number of training stages for our algorithm. We try multi-layer perceptrons with 5, 10, 20, 40 and 80 hidden units. The multi-layer perceptrons and the ordinary single-layer perceptrons are

Table 13.1: Test-set accuracy.

	method					
			perceptrons			
domain	BBP	C4.5	multi-layer	ordinary	2nd-order	pruned
voting	91.5%	89.2%	92.2%	90.8%	89.2%	87.6%
promoters	92.7	85.0	90.6	90.0	88.7	88.2
protein coding	72.9	62.6	71.6	70.7	69.8	70.3

trained using a conjugate-gradient procedure for a maximum of 100 search directions.

Table 13.1 reports test-set accuracy for each method on all three domains. We measure the statistical significance of accuracy differences using a paired, two-tailed t-test with a significance level of 0.05. On the voting domain, the accuracy of the BBP algorithm is comparable to both multi-layer perceptrons and ordinary single-layer perceptrons, *i.e.*, the measured accuracy differences between these three algorithms are not statistically significant. For both the promoter and the protein-coding domains, our algorithm generalizes better than multi-layer perceptrons and ordinary single-layer perceptrons. Both differences are statistically significant in the protein-coding domain, and the difference between our algorithm and single-layer perceptrons is significant in the promoter domain. For all three domains, BBP is superior to C4.5, the perceptrons with second-order features and the OBD-pruned perceptrons, and all of these differences are statistically significant. From these results we conclude that (1) BBP exhibits good generalization performance on a number of interesting real-world problems, and (2) the generalization performance of the algorithm is not explained solely by its use of second-order features, nor is it solely explained by the sparseness of the perceptrons it produces. An interesting open question is whether perceptrons trained with both pruning and second-order features are able to match the accuracy of our algorithm; we plan to investigate this question in future work.

Table 13.2 reports the average number of weights for all of the perceptrons. For all three problems, the BBP algorithm produces perceptrons with fewer weights than the multi-layer perceptrons, the ordinary perceptrons, and the perceptrons with second-order features. The sizes of the OBD-pruned perceptrons and those produced by our algorithm are comparable

Table 13.2: Hypothesis complexity (# weights).

	method				
		perceptrons			
domain	BBP	multi-layer	ordinary	2nd-order	pruned
voting	12	651	30	450	12
promoters	41	2267	228	25764	59
protein coding	52	4270	60	1740	37

for all three domains. Recall, however, that for all three tasks, the perceptrons learned by our algorithm had significantly better generalization performance than their similar-sized OBD-pruned counterparts. We contend that the sizes of the perceptrons produced by our algorithm are within the bounds of what humans can readily understand. In the biological literature, for example, linear discriminant functions are frequently used to communicate domain knowledge about sequences of interest. These discriminant functions frequently involve more weights than the perceptrons produced by our algorithm. We conclude, therefore, that our algorithm produces hypotheses that are not only accurate, but also comprehensible.

The results on the protein-coding domain are especially interesting. The input representation for this problem consists of 15 nominal features representing 15 consecutive *bases* (nucleotides) in a DNA sequence. In the regions of DNA that encode proteins (the positive examples in our task), non-overlapping triplets of consecutive bases represent meaningful "words" called *codons*. In previous work [CS93b], it has been found that a feature set that explicitly represents codons results in better generalization than does a representation of just bases. However, we used the bases representation in our experiments in order to investigate the ability of our algorithm to select the "right" second-order features. Interestingly, nearly all of the second-order features included in the perceptrons produced by our algorithm represent conjunctions of bases that are in the same codon. This result suggests that BBP is especially good at selecting relevant features from large feature sets.

Chapter 14

Further Work

While we have shown that DNF is efficiently learnable in a particular model of learning, the question of whether or not DNF is learnable in more general models—and in particular in the PAC model—remains open. A number of interesting intermediate questions also remain unanswered. For example, is it possible to learn DNF (even monotone DNF, in which no variables are negated) with respect to uniform *without* using membership queries?

In Chapter 6 we briefly compared `HS` with earlier Fourier-based learning algorithms. There we pointed out that `HS` has the *potential* to learn representation classes that these previous algorithms might not be able to learn efficiently. Do such classes exist—and in particular, is DNF such a class? Or is it possible that `HS` is no more powerful than earlier algorithms, that is, that the results of this paper are the consequence of an improved analysis rather than an improved algorithm?

On the applications side, our next step will be to further modify the `BBP` algorithm for learning sparse perceptrons so that it can be applied to real-valued as well as discrete feature domains. We then plan to test the algorithm over a much broader spectrum of benchmark tasks than the three considered here. We are also interested in comparing `BBP` with other theory-based algorithms that have been empirically tested, such as Winnow [Lit95] and Weighted Majority [Blu95], as well as other empirical algorithms. Along these lines, the current algorithm is designed to learn within a time-invariant learning framework, and therefore may not perform as well as, say, Weighted Majority in domains where the target function may vary over time. It would be especially nice to have an algorithm that worked well in this on-line setting while producing sparse hypotheses.

Another potential application area for the Harmonic Sieve is within hardware circuit design and analysis. For example, given the truth table for a function over a reasonable number of inputs (say up to 25), the Harmonic Sieve can be used to produce a TOP representation of the function that is at most polynomially larger than the smallest possible TOP. For at least some functions, the TOP produced may be significantly smaller than the optimal DNF expression of the function. Does this occur in practice, and if so, how significant are the size reductions achieved?

Bibliography

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[AM91] William Aiello and Milena Mihail. Learning the Fourier spectrum of probabilistic lists and trees. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 291–299, 1991.

[AHP92] Howard Aizenstein, Lisa Hellerstein, and Leonard Pitt. Read-thrice DNF is hard to learn with membership and equivalence queries. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 523–532, 1992.

[AP91] Howard Aizenstein and Leonard Pitt. Exact learning of read-twice DNF formulas. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 170–179, 1991.

[AP92] Howard Aizenstein and Leonard Pitt. Exact learning of read-k disjoint DNF and not-so-disjoint DNF. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 71–76, 1992.

[ADTss] R. Andrews, J. Diederich, and A. B. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, (in press).

[Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[Ang90] Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[AFP90] Dana Angluin, Michael Frazier, and Leonard Pitt. Learning conjunctions of Horn clauses. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 186–192, 1990.

[AK91] Dana Angluin and Michael Kharitonov. When won't membership queries help? In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 444–454, 1991.

[AD93] Javed A. Aslam and Scott E. Decatur. General bound on statistical query learning and PAC learning with noise via hypothesis boosting. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 282–291, 1993.

[Bel91] Mihir Bellare. The spectral norm of finite functions. Technical Report TR-495, MIT-LCS, February 1991.

[Bel92] Mihir Bellare. 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, 1992.

[Blu90] Avrim Blum. Separating distribution-free and mistake-bound learning models over the Boolean domain. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 211–218, 1990.

[Blu92] Avrim Blum. Rank-r decision trees are a subclass of r-decision lists. *Information Processing Letters*, 42:183–185, 1992.

[Blu95] Avrim Blum. Empirical support for Winnow and Weighted-Majority based algorithms: results on a calendar scheduling domain. In *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, CA, 1995.

[BFJ+94] Avrim Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 253–262, 1994.

[BKK+94] Avrim Blum, Roni Khardon, Eyal Kushilevitz, Leonard Pitt, and Dan Roth. On learning read-k-satisfy-j DNF. In *Proceedings of the Seventh Annual Workshop on Computational Learning Theory*, pages 110–117, 1994.

[BR92] Avrim Blum and Steven Rudich. Fast learning of k-term DNF formulas with queries. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 382–389, 1992.

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, October 1989.

[Bol85] Béla Bollobás. *Random Graphs*. Academic Press, 1985.

[Bru90] Jehoshua Bruck. Harmonic analysis of polynomial threshold functions. *SIAM Journal of Discrete Mathematics*, 3(2):168–177, May 1990.

[BS90] Jehoshua Bruck and Roman Smolensky. Polynomial threshold functions, AC^0 functions and spectral norms. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 632–641, 1990.

[Bsh] Nader Bshouty. Personal communication.

[Bsh93] Nader H. Bshouty. Exact learning via the monotone theory. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 302–311, 1993.

[Bsh95] Nader H. Bshouty. Simple learning algorithms using divide and conquer. In *Proceedings of the Eighth Annual Workshop on Computational Learning Theory*, pages 447–453, 1995.

[BGGM94] Nader H. Bshouty, Paul W. Goldberg, Sally A. Goldman, and H. David Mathias. Exact learning of discretized geometric concepts. Technical Report WUCS-94-19, Washington University, July 1994.

[BJ95] Nader H. Bshouty and Jeffrey C. Jackson. Learning DNF over the uniform distribution using a quantum example oracle. In *Proceedings of the Eighth Annual Workshop on Computational Learning Theory*, pages 118–127, 1995.

[BT95] Nader H. Bshouty and Christino Tamon. On the Fourier spectrum of monotone functions. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 1995.

[BN92] W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–86, 1992.

[CS93a] M. W. Craven and J. W. Shavlik. Learning symbolic rules using artificial neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 73–80, Amherst, MA, 1993. Morgan Kaufmann.

[CS93b] Mark W. Craven and Jude W. Shavlik. Learning to represent codons: A challenge problem for constructive induction. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1993.

[CS94] Mark W. Craven and Jude W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 37–45, 1994.

[DSS93] H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):705–719, August 1993.

[EH89] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, September 1989.

[Fre] Yoav Freund. Personal communication.

[Fre90] Yoav Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216, 1990.

[Fre92] Yoav Freund. An improved boosting algorithm and its implications on learning complexity. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 391–398, 1992.

[Fre93] Yoav Freund. *Data Filtering and Distribution Modeling Algorithms for Machine Learning*. PhD thesis, University of California at Santa Cruz, September 1993. Available as technical report UCSC-CRL-93-37.

[FS95] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second Annual European Conference on Computational Learning Theory*, 1995.

[FJS91] Merrick L. Furst, Jeffrey C. Jackson, and Sean W. Smith. Improved learning of AC^0 functions. In *Fourth Annual Workshop on Computational Learning Theory*, pages 317–325, 1991.

[FSS81] Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial time hierarchy. In *22nd Annual Symposium on Foundations of Computer Science*, pages 260–270, 1981.

[GGM94] Paul W. Goldberg, Sally A. Goldman, and H. David Mathias. Learning unions of boxes with membership and equivalence queries. In *Proceedings of the Seventh Annual Workshop on Computational Learning Theory*, pages 198–207, 1994.

[GKS93] Sally A. Goldman, Michael J. Kearns, and Robert E. Schapire. Exact identification of circuits using fixed points of amplification functions. *SIAM Journal on Computing*, 22(4):705–726, August 1993. Preliminary version appeared in *Proceedings of the 31st Symposium on Foundations of Computer Science*, pages 193–202, 1990.

[GHR92] M. Goldmann, J. Hastad, and A. Razborov. Majority gates vs. general weighted threshold gates. In *Proceedings of the Seventh IEEE Conference on Structure in Complexity Theory*, pages 2–13, 1992.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[Han91] Thomas R. Hancock. Learning 2μDNF formulas and $k\mu$ decision trees. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 199–209, 1991.

[Han93] Thomas R. Hancock. Learning $k\mu$ decision trees on the uniform distribution. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 352–360, 1993.

[HB90] S. J. Hanson and D. J. Burr. What connectionist models learn: Learning and representation in connectionist networks. *Behavioral and Brain Sciences*, 13:471–518, 1990.

[Has86] J. Hastad. *Computational Limitations for Small Depth Circuits.* PhD thesis, MIT Press, 1986.

[Hau88] David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, pages 177–221, 1988.

[HK93] L. Hunter and T. Klein. Finding relevant biomolecular features. In *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, pages 190–197, Bethesda, MD, 1993. AAAI Press.

[Jac94] Jeffrey Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 42–53, 1994.

[JS68] K. Jogdeo and S. M. Samuels. Monotone convergence of binomial probabilities and a generalization of Ramanujan's equation. *Annals of Math. Statistics*, 39:1191–1195, 1968.

[KLPV87] Michael Kearns, Ming Li, Leonard Pitt, and Leslie Valiant. On the learnability of Boolean formulae. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 285–295, 1987.

[KV89] Michael Kearns and Leslie G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 433–444, 1989.

[Kea93] Michael J. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 392–401, 1993.

[Kha92] Michael Kharitonov. Cryptographic lower bounds for learnability of Boolean functions on the uniform distribution. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 29–36, 1992.

[Kha93] Michael Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 372–381, 1993.

[KP94] Matthias Krause and Pavel Pudlák. On the computational power of depth 2 circuits with threshold and modulo gates. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 48–57, 1994.

[KM93] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, December 1993. Earlier version appeared in *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 455–464, 1991.

[KR93] Eyal Kushilevitz and Dan Roth. On learning visual concepts and DNF formulae. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 317–326, 1993.

[LDS89] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605. Morgan Kaufmann, San Mateo, CA, 1989.

[Lev93] Leonid A. Levin. Randomness and non-determinism. *Journal of Symbolic Logic*, 58(3):1102–1103, 1993.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *Journal of the ACM*, 40(3):607–620, July 1993. Earlier version appeared in *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 574–579, 1989.

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[Lit95] N. Littlestone. Comparing several linear-threshold algorithms on tasks involving superfluous attributes. In *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, CA, 1995.

[LW90] Philip M. Long and Manfred K. Warmuth. Composite geometric concepts and polynomial predictability. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 273–287, 1990.

[Man92] Yishay Mansour. An $O(n^{\log\log n})$ learning algorithm for DNF under the uniform distribution. In *Fifth Annual Workshop on Computational Learning Theory*, pages 53–61, 1992.

[MP88] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1988.

[Pom91] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Ris89] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific, 1989.

[Riv87] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[Ros62] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, 1962.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, volume 1, chapter 8, pages 318–362. MIT Press, 1986.

[SM94] Yoshifumi Sakai and Akira Maruoka. Learning monotone log-term DNF formulas. In *Proceedings of the Seventh Annual Workshop on Computational Learning Theory*, pages 165–172, 1994.

[Sch90] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[SF86] J. C. Schlimmer and D. Fisher. A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 496–501, Philadelphia, PA, 1986.

[SKLM94] M. Sternberg, R. King, R. Lewis, and S. Muggleton. Application of machine learning to structural molecular biology. *Phil. Trans. R. Soc. Lond. B*, 344:365–371, 1994.

[Tes94] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master- level play. *Neural Computation*, 6(2):215–19, March 1994.

[Thr93] Sebastian B. Thrun. Extracting provably correct rules from artificial neural networks. Technical Report IAI-TR-93-5, University of Bonn, May 1993.

[TS94] Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, page to appear, 1994.

[TSN90] G. Towell, J. Shavlik, and M. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866, Boston, MA, 1990. AAAI/MIT Press.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[Val85] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 560–566, 1985.

[VC71] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16(2):264–280, 1971.

[Ver90] Karsten Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 314–326, 1990.