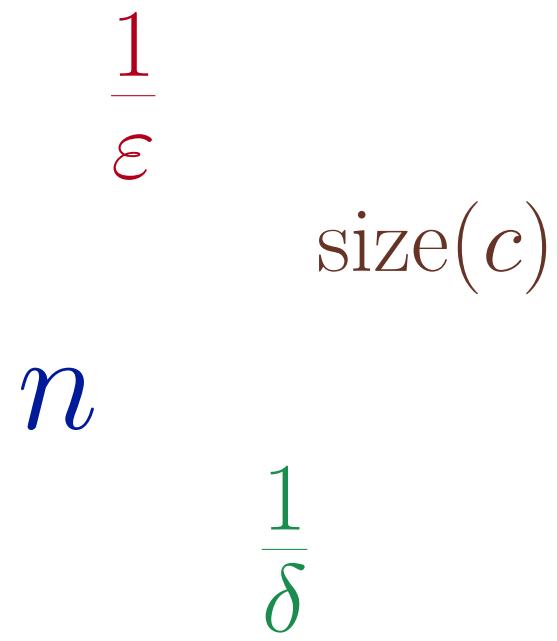
Computational Learning Theory

(Lecture Notes)



Varun Kanade

Contents

Contents i

Preface v

Acknowledgements vii

1 Probably Approximately Correct Learning 1
- 1.1 A Rectangle Learning Game . . . 1
- 1.2 Key Components of the PAC Learning Framework . . . 4
- 1.3 Learning Conjunctions . . . 9
- 1.4 Hardness of Learning 3-term DNF . . . 13
- 1.5 Learning 3-CNF vs 3-TERM-DNF . . . 15
- 1.6 PAC Learning . . . 16
- 1.7 Exercises . . . 17
- 1.8 Chapter Notes . . . 19

2 Consistent Learning and Occam's Razor 21
- 2.1 Occam's Razor . . . 21
- 2.2 Consistent Learning . . . 22
- 2.3 Improved Sample Complexity . . . 24
- 2.4 Exercises . . . 25
- 2.5 Chapter Notes . . . 26

3 The Vapnik Chervonenkis Dimension 27
- 3.1 The Vapnik Chervonenkis (VC) Dimension . . . 27
- 3.2 Growth Function . . . 30
- 3.3 Sample Complexity Upper Bound . . . 32
- 3.4 Sample Complexity Lower Bounds . . . 34
- 3.5 Consistent Learner for Linear Threshold Functions . . . 36
- 3.6 Exercises . . . 37

4 Boosting 39
- 4.1 Weak Learnability . . . 39
- 4.2 The AdaBoost Algorithm . . . 40
- 4.3 Exercises . . . 44

5 Cryptographic Hardness of Learning 45
- 5.1 The Discrete Cube Root Problem . . . 45
- 5.2 A learning problem based on the DCRA . . . 47

5.3 Chapter Notes 48

6 Exact Learning using Membership and Equivalence Queries 51
6.1 Exact Learning with Membership and Equivalence Queries . . 52
6.2 Exact Learning MONOTONE-DNF using MQ + EQ 54
6.3 Learning DFA 56
6.4 Exercises 60
6.5 Chapter Notes 61

7 Statistical Query Learning 63
7.1 Random Classification Noise Model 63
7.2 Statistical Query Model 66
7.3 A hard-to-learn concept class 70
7.4 Exercises 72
7.5 Bibliographic Notes 72

8 Learning Real-valued Functions 75
8.1 Learning Real-Valued Functions 75
8.2 Projected Gradient Descent for Lipschitz functions 79
8.3 Rademacher Complexity 81
8.4 Linear Regression 87
8.5 Generalised Linear Models 90

9 Mistake-Bounded Learning 95
9.1 Online Prediction Framework 95
9.2 Relationships to Other Models of Learning 98
9.3 The Halving Algorithm and Some Examples 99
9.4 Perceptron 100
9.5 The Winnow Algorithm 103

10 Online Learning with Expert Advice 107
10.1 Learning with Expert Advice 107
10.2 Follow The Leader 109
10.3 The Multiplicative Weight Update Algorithm (MWUA) 109
10.4 Application: Boosting Algorithm 112
10.5 Application: von Neumann's Min-Max Theorem 113

A Inequalities from Probability Theory 117
A.1 The Union Bound 117
A.2 Hoeffding's Inequality 117
A.3 Chernoff Bound 117

B Elementary Inequalities 119
B.1 Convexity of exp 119
B.2 Auxilliary Lemmas 119

C Notation 121
C.1 Basic Mathematical Notation 121
C.2 The PAC Learning Framework 121

Bibliography 123

Index **127**

Preface

What is computational learning theory?

Machine learning techniques lie at the heart of many technological applications that are used on a daily basis. When using a digital camera, the boxes that appear around faces are produced using a machine learning algorithm. When streaming portals such as BBC iPlayer or Netflix suggest what a user might like to watch next, they are also using machine learning algorithms to provide these recommendations. In fact, more likely than not, any substantial technology that is in use these days has some component that uses machine learning techniques.

The field of (computational) learning theory develops precise mathematical formulations of the more vague notion of *learning from data*. Having precise mathematical formulations allows one to answer questions such as:

(i) What types of functions are easy to learn?

(ii) Are there types of functions that are hard to learn?

(iii) How much data is required to learn a function of a particular type?

(iv) How much computational power is needed to learn certain types of functions?

Positive as well as negative answers to these questions are of great interest. For example, one of the key considerations is to design and analyse learning algorithms that are guaranteed to learn certain types of functions using modest amount of data and reasonable runnning time. For the most part, we will take the view that as long as the resources used can be bounded by a polynomial function of the problem size, the learning algorithm is efficient. Obviously, as is the case in the analysis of algorithms, there may be situations where just being polynomial time may not be considered efficient enough; the existence of polynomial-time learning algorithms is however a good first step in separating easy and hard learning problems. Some of the algorithms we study will not run in polynomial time at all, but they will still be much better than brute force algorithms.

There is a vast body of literature that is often called *Statistical Learning Theory*. To some extent this distinction between *statistical* and *computational* learning theory is rather artificial and we shall make use of several concepts introduced in that theory such as VC dimension and Rademacher complexity. In this course, greater emphasis will be placed on computational considerations. Research in *computational learning theory* has uncovered interesting phenomena such as the existence of certain types of functions that can be learnt if computational

resources are not a consideration, but cannot be learnt in polynomial time. Other examples demonstrate a tradeoff between the amount of data and the algorithmic running time, i.e. the running time of the algorithm can be reduced by using more data. More importantly, placing the question of learning in a computational framework allows one to reason about other kinds of (computational) resources such as memory, communication, privacy, etc. that may be a consideration for the learning problem at hand.

Acknowledgements

The course materials have been mainly developed by using the book by Michael Kearns and Umesh Vazirani [34] and material taught in lecture courses on learning theory by Leslie Valiant, Adam Tauman Kalai, Avrim Blum, Adam Klivans, Rocco Servedio and several others. I am particularly grateful to Les Valiant and Adam Kalai, without whom I may have never been interested in computational learning theory. Many of the exercises are directly lifted from problem sheets developed at Harvard University over three decades by Les Valiant, Michael Kearns, Scott Decatur, Rocco Servedio, Vitaly Feldman, and several others. My apologies if I have missed anyone from this list.

I am grateful to Ben Worrell for providing the material included in Chapter 6, particularly Section 6.3 on learning DFAs. The lecture notes have benefited greatly from feedback received from students at the University of Oxford between 2017-2023. I'm very grateful to Francisco Marmolejo, Matthias Gerstgrasser, David Martínez-Rubio, Ninad Rajgopal, Alexandros Hollender, Philip Lazos, Tomas Vaškevičius, Amartya Sanyal, Tom Orton, Pascale Gourdeau, Alex Buna Marginean, Satwik Bhattamishra and Sílvia Casacuberta Puig who have been class tutors for this course over the years at Oxford and without whom this course or these lecture notes would not have existed. The cover material is inspired by the submission of one of our undergraduate students who made my job of marking so much more pleasurable by the beautiful designs and aesthetic elegance in their work, but mostly by doing everything correctly.

Any errors that still remain are entirely due to me and I would be grateful to be notified of them, whether they be mathematical, linguistic, or stylistic.

Chapter 1

Probably Approximately Correct Learning

Our goal in this chapter is to gradually build up the probably approximately correct (PAC) learning framework while emphasising the key components of the learning model. We will discuss various model choices in detail; the exercises and some results in later chapters explore the robustness of the PAC learning framework to slight variants of these design choices. As the goal of computational learning theory is to shed light on the phenomenon of automated learning, such robustness is of key importance.

1.1 A Rectangle Learning Game

Let us consider the following rectangle learning game. We are given some points in the Euclidean plane, some of which are labelled positive (+) and others negative (−). Furthermore, we are guaranteed that there is an axis-aligned rectangle such that all the points inside it are labelled positive, while those outside are labelled negative. However, this rectangle itself is not revealed to us. Our goal is to produce a rectangle that is "close" to the *true hidden* rectangle that was used to label the observed data (see Fig. 1.1(a)).

Although the primary purpose of this example is pedagogical, it may be worth providing a scenario where such a (fake) learning problem may be relevant. Suppose that the two dimensions measure the curvature and length of bananas. The points that are labelled positive have *medium* curvature and *medium* length and represent the bananas that would pass "stringent" EU regulations. However, the actual lower and upper limits that "define" *medium* in each dimension are hidden. Thus, we wish to learn some rectangle that will be good enough to predict whether bananas we produce would pass the regulators' tests or not.

Let R be the unknown rectangle used to label the points. We can express the labelling process using a *boolean* function $c_R : \mathbb{R}^2 \to \{+, -\}$, where $c_R(\mathbf{x}) = +$, if $\mathbf{x}$ is inside the rectangle R and $c_R(\mathbf{x}) = -$, otherwise.[1]

[1] We refer to functions whose range has size at most 2 as *boolean* functions. From the point of view of machine learning, the exact values in the range are unimportant. We will frequently use $\{+, -\}$, $\{0, 1\}$ and $\{-1, +1\}$ as the possible options for the range depending on the context (and at times make rather unintuitive transformations between these possibilities).

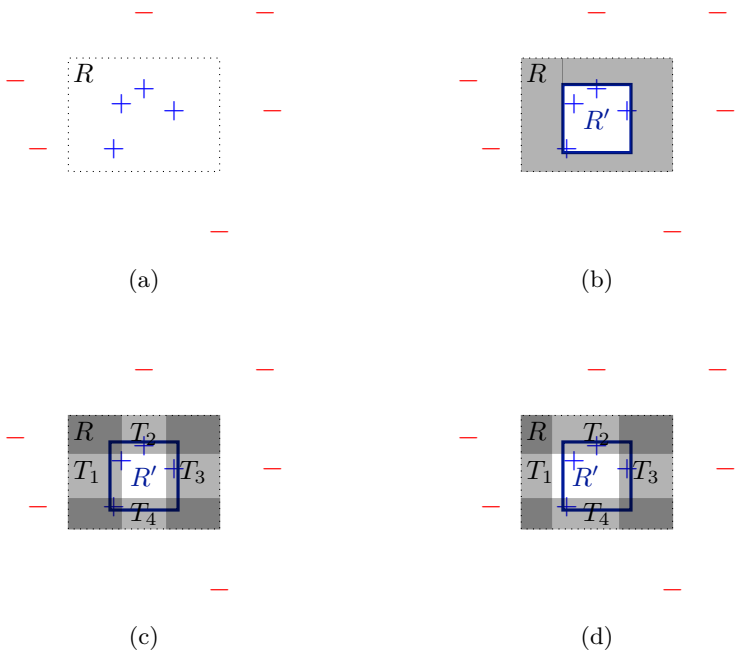


Figure 1.1: (a) Data received for the rectangle learning game. The rectangle R used to generate labels is hidden from the learning algorithm. (b) The *tightest fit* algorithm produces a rectangle R'. (c) & (d) The regions T_1, T_2, T_3 and T_4 contain $\epsilon/4$ mass each under D (for two different distributions D).

Let us consider the following simple algorithm. We consider the *tightest* possible axis-aligned rectangle that can fit all the positively labelled data inside it; let us denote this rectangle by R' (Fig. 1.1(b)). Our prediction function or *hypothesis* $h_{R'} : \mathbb{R}^2 \to \{+, -\}$ is the following: if $\mathbf{x} \in R'$, $h_{R'}(\mathbf{x}) = +$, else $h_{R'}(\mathbf{x}) = -$.[2] Let us consider the following questions:

- Have we learnt the function c_R ?
- How good is our prediction function $h_{R'}$?

Let R denote the *true* rectangle that actually defines the labelling function c_R. Since we've chosen the tightest possible fit, the rectangle R' must be entirely contained inside R. Consider the shaded region shown in Fig. 1.1(b). For any point $\mathbf{x}$ that is in this shaded region, it must be that $h_{R'}(\mathbf{x}) = -$, while $c_R(\mathbf{x}) = +$. In other words, our prediction function $h_{R'}$ would make errors on all of these points. If we had to make predictions on points that mostly lie in this region our hypothesis would be quite bad. This raises an important point that the data that is used to learn a hypothesis should be similar to the data on which the hypothesis will be tested. We will now formalise this notion.

Let D be a probability distribution over $\mathbb{R}^2$; in the ensuing discussion, we will assume that D can be expressed using a density function that is defined

[2]For the sake of concreteness, let us say that points on the sides are considered to be inside the rectangle.

While no knowledge of machine learning is required to complete this course, it would of course be helpful to make connections with the techniques and terminology in machine learning. This discussion appears in coloured boxes and can be safely ignored for those uninterested in applied machine learning.

In machine learning, one makes the distinction between the *training* and *test* datasets; when done correctly the *empirical error* on the test dataset would give an unbiased estimate of what we refer to as error in Eqn. (1.1). In machine learning it is also common to use a *validation set*; this is often done because multiple models are trained on the training set (possibly because of hyperparameters) and one of them needs to be picked. Picking them using their performance on the training set may result in overfitting. In this course, we will adopt the convention that model selection is also part of the learning algorithm and not make a distinction between the training and validation sets. (For example, see Exercise 1.3.)

over all of $\mathbb{R}^2$ and is continuous.[3] The *training data* consists of m points that are drawn independently according to D and then labelled according to the function c_R. We will define the error of a hypothesis $h_{R'}$ with respect to the target function c_R and distribution D as follows:

$$\mathrm{err}(h_{R'}; c_R, D) = \mathbb{P}_{\mathbf{x}\sim D}\left[h_{R'}(\mathbf{x}) \neq c_R(\mathbf{x})\right] \tag{1.1}$$

Whenever the target c_R and distribution D are clear from context, we will simply refer to this as $\mathrm{err}(h_{R'})$.

We will now show that in fact our algorithm outputs an $h_{R'}$ that is quite good, in the sense that given any $\epsilon > 0$ as the target error, with high probability (at least $1 - \delta$), given a sufficiently large training sample, it will output $h_{R'}$ such that $\mathrm{err}(h_{R'}; c_R, D) \leq \epsilon$. Consider four rectangular strips T_1, T_2, T_3, T_4 that are chosen along the sides of the rectangle R (and lying inside R) such that the probability that a random point drawn according to D lands in some T_i is exactly $\epsilon/4$.[4] Note that some of these strips overlap, e.g. T_1 and T_2 (see Fig. 1.1(c)). The probability that a point drawn randomly according to D lies in the set $T_1 \cup T_2 \cup T_3 \cup T_4$ is at most ϵ (a fact that can be proved formally using the union bound (cf. Appendix A.1)). If we can guarantee that the training data of m points contains at least one point from each of T_1, T_2, T_3 and T_4, then the tightest fit rectangle R' will be such that $R \setminus R' \subset T_1 \cup T_2 \cup T_3 \cup T_4$, and as a consequence, $\mathrm{err}(h_{R'}; c_R, D) \leq \epsilon$. This is shown in Fig. 1.1(c); note that if even one of the T_i do not contain any point in the data, this may cause a problem, in the sense that the region of disagreement between R and R' may have probability mass greater than ϵ (see Fig. 1.1(d)).

Let A_1 be the event that when m points are drawn independently according to D, none of them lies in T_1. Similarly, define the events A_2, A_3, A_4 for

[3] This assumption is not required; in the exercises you are asked to show how the assumption can be removed.

[4] Assuming that the distribution D can be expressed using a continuous density function that is defined over all of $\mathbb{R}^2$, such strips always exist. Otherwise, the algorithm is still correct, however, the analysis is slightly more tedious and is left as Exercise 1.1.

T_2, T_3, T_4. Consider the event $\mathcal{E} = A_1 \cup A_2 \cup A_3 \cup A_4$. If $\mathcal{E}$ does not occur, then we have already argued that $\mathrm{err}(h_{R'}; c_R, D) \leq \epsilon$. We will use the union bound to bound $\mathbb{P}[\mathcal{E}]$ (cf. Appendix A.1). To begin, let us compute $\mathbb{P}[A_1]$. The probability that a single point drawn according to D does not land in T_1 is exactly $1-\epsilon/4$; so the probability that after m independent draws from D, none of the points are in T_1 is $\left(1 - \frac{\epsilon}{4}\right)^m$. By a similar argument, $\mathbb{P}[A_i] = \left(1 - \frac{\epsilon}{4}\right)^m$ for $i = 1, \ldots, 4$. Thus, we have

$$\begin{aligned}
\mathbb{P}[\mathcal{E}] &\leq \sum_{i=1}^{4} \mathbb{P}[A_i] && \text{The Union Bound (A.1).} \\
&= 4\left(1 - \frac{\epsilon}{4}\right)^m \\
&\leq 4\exp\left(-\frac{m\epsilon}{4}\right). && \text{As } 1 - x \leq e^{-x} \text{ (B.1).}
\end{aligned}$$

For any $\delta > 0$, picking $m \geq \frac{4}{\epsilon}\log\left(\frac{4}{\delta}\right)$ suffices to ensure that $\mathbb{P}[\mathcal{E}] \leq \delta$. In other words, with probability at least $1 - \delta$, $\mathrm{err}(h_{R'}; c_R, D) \leq \epsilon$.

A couple of remarks are in order. We should think of ϵ as being the accuracy parameter and δ being the confidence parameter. The bound $m \geq \frac{4}{\epsilon}\log\left(\frac{4}{\delta}\right)$ suggests that as we demand higher accuracy (smaller value of ϵ) and higher confidence (smaller value of δ) of our learning algorithm, we need to supply more data.[5] This is indeed a reasonable requirement. Furthermore, the cost of achieving higher accuracy and higher confidence is relatively modest. For example, if we want to halve the error while keeping the confidence parameter constant, say go from $\epsilon = 0.02$ to $\epsilon = 0.01$, the amount of data required (as suggested by the bound) only doubles.[6]

There is another corner case that needs to be considered. What if we observe no positively labelled points? Or only one of them? We will allow the learning algorithm to use degenerate rectangles, which include the empty set, points, and line segments that are parallel to one of the axes. So $h_{R'}$ as produced by our algorithm is still well-defined. It is easy to check that the rest of the analysis remains unchanged. In short, if after drawing $m \geq \frac{4}{\epsilon}\log\left(\frac{4}{\delta}\right)$ points independently from D, we have still not seen a single positively labelled point, then outputting a hypothesis $h_{R'}$ that always predicts negative, does satisfy with probability at least $1 - \delta$, that its error it at most ϵ. As this was our very first example, we discussed the corner cases in detail. Further on in the course, you should convince yourselves that the corner cases indeed pose no problem to our analyses.

1.2 Key Components of the PAC Learning Framework

We will use the insights gleaned from the rectangle learning game to develop key components of a mathematical framework for automatic learning from data. First let us make a few observations:

[5] So far, we have only established sufficient conditions, i.e. upper bounds, on the sample complexity required for learning. In later chapters we will establish necessary conditions, i.e. lower bounds on the amount of data required for learning algorithms.

[6] We are using the word "required" a bit loosely here. All we can say is our present analysis of this particular algorithm suggests that the amount of data required scales linearly as $\frac{1}{\epsilon}$. We will see lower bounds of this nature that hold for any algorithm in later chapters.

1. The learning algorithm does not know the target concept to be learnt (obviously, otherwise there is nothing to learn!). However, the learning algorithm does know the set of possible target concepts. In the rectangle learning game, the unknown target is always an axis-aligned rectangle.
2. The learning algorithm has access to data drawn from some distribution D. We do assume that the observations are drawn independently according to D. However, no assumption is made on the distribution D itself. This reflects the fact that the environments in which learning agents operate may be very complex and it is unrealistic to assume that the observations are generated according to some distribution that is easy to describe.
3. The output hypothesis is evaluated with respect to the same distribution D that generated the training data.
4. We would like learning algorithms to be *statistically efficient*, i.e. they should require a relatively small training sample to guarantee high accuracy and confidence, as well as *computationally efficient*, i.e. they should run in a reasonable amount of time. In general, we shall take the view that learning algorithms for which the training sample size and running time scales polynomially with the size parameters are efficient. However, in some cases we will be more precise and specify the exact running time and sample size.

Let us now formalise a few other concepts related to learning.

Instance Space

Let X denote the set of possible instances; an instance is the *input* part, $\mathbf{x}$, of a training example $(\mathbf{x}, y)$, and y is the target label. In the rectangle learning game, the instances were points in $\mathbb{R}^2$; the instance space was $\mathbb{R}^2$. When considering binary classification problems for images, the instances may be 3 dimensional arrays, containing the RGB values of each pixel. Mostly, we shall be concerned with the case when $X = \{0,1\}^n$ or $X = \mathbb{R}^n$; other instance spaces can be usually mapped to one of these, as is often done in machine learning.

Concept Class

A *concept* c over an instance space X is a boolean function $c : X \to \{0,1\}$. (We will consider learning target functions that are not boolean later in the course.) A concept class C over X is a collection of concepts c over X. In the rectangle learning game, the concept class is the set of all axis-aligned rectangles in $\mathbb{R}^2$. The learning algorithm has knowledge of C, but not of the specific concept $c \in C$ that is used to label the observations. A concept class that contains concepts that are too simple may not be expressive enough to describe the real-world process we are trying to *learn*. On the other hand, considering a concept class that is too large, e.g. all boolean functions, would not allow us to design efficient learning algorithms.

Data Generation

Let D be a probability distribution over X. The training data is obtained as follows. An instance $\mathbf{x} \in X$ is drawn according to the distribution D. If c is

the target concept, the instance $\mathbf{x}$ is labelled accordingly as $c(\mathbf{x})$. The learning algorithm observes the example $(\mathbf{x}, c(\mathbf{x}))$. We will refer to this process as an example oracle, denoted by $\mathsf{EX}(c, D)$. We assume that a learning algorithm can query the oracle $\mathsf{EX}(c, D)$ at unit cost and each query yields an *independent* training example.

1.2.1 PAC Learning: Take I

Let $h : X \to \{0, 1\}$ be some hypothesis; we typically refer to the boolean function output by a learning algorithm as a *hypothesis* to distinguish it from the *target*. For a distribution D over X and a fixed target $c \in C$, the error of h with respect to c and D is defined as:

$$\mathrm{err}(h; c, D) = \mathbb{P}_{\mathbf{x} \sim D}\left[h(\mathbf{x}) \neq c(\mathbf{x})\right]. \tag{1.2}$$

When c and D are clear from context, we will simply refer to this as $\mathrm{err}(h)$.

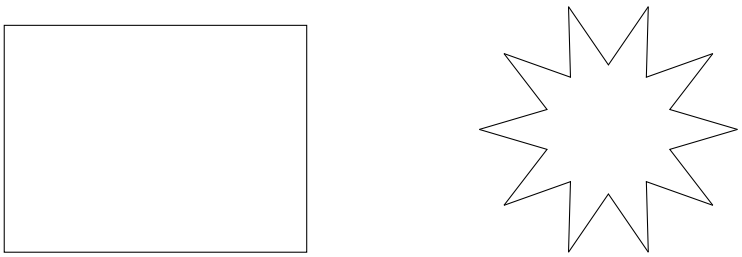
***Definition 1.1* – PAC Learning: Take I.** *Let C be a concept class over X. We say that C is* PAC (take I) learnable *if there exists a learning algorithm L that satisfies the following: for every concept $c \in C$, for every distribution D over X, for every $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $\mathsf{EX}(c, D)$ and inputs ϵ and δ, L outputs a hypothesis $h \in C$ that with probability at least $1 - \delta$ satisfies $\mathrm{err}(h) \leq \epsilon$. The probability is over the random examples drawn from $\mathsf{EX}(c, D)$ as well as any internal randomisation of L. The number of calls made to $\mathsf{EX}(c, D)$* (sample complexity) *must be bounded by a polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.*

We further say that C is efficiently *PAC (take I) learnable if the running time of L is polynomial in $1/\epsilon$ and $1/\delta$.*

The term PAC stands for probably approximately correct. The *approximately correct* part captures the notion that the most that can be guaranteed is that the error of the output hypothesis can be bounded to be below a desired level; demanding higher accuracy (lower ϵ) is possible, but comes at a cost of increased running time and sample complexity. In most cases, achieving *exactly* zero error is infeasible as it is possible that two target concepts may be identical except on one instance which is very unlikely to be drawn according to the distribution D.[7] The *probably* part captures the notion that there is some chance that the algorithm may fail completely. This may happen because the observations are not representative of the underlying data distribution, a low probability event, though very much a possible event. Our confidence (lower δ) in the correctness of our algorithm is increased as we allow more sample complexity and running time.

Based on our analysis of the rectangle learning game in Section 1.1, we have essentially already proved the following theorem.

Theorem 1.2. *The concept class of axis-aligned rectangles in $\mathbb{R}^2$ is efficiently PAC (take I) learnable.*



(a) (b)

Figure 1.2: Different shape concepts in $\mathbb{R}^2$.

1.2.2 PAC Learning: Take II

Having proved our first result in PAC learning, let us discuss a couple of issues that we have glossed over so far. The first question concerns that of the complexity of the concepts that we are trying to learn. For example, consider the question of learning rectangles (Fig. 1.2(a)) versus more complex shapes such as shown in Fig. 1.2(b). Intuitively, we believe that it should be harder to learn concepts defined by shapes like in Fig. 1.2(b) than rectangles. Thus, within our mathematical learning framework, an algorithm that learns a more complex class should be allowed more resources (sample size, running time, memory, etc.). In order to represent an axis-aligned rectangle, we only need to store four real numbers, the lower and upper limits in both the x and y directions. The number of real numbers used to represent more complex shapes is higher.[8]

The question of representation is better elucidated by taking the case of boolean functions defined on the *boolean hypercube* $X = \{0,1\}^n$, the set of length n bit vectors. Consider a boolean function $f : X \to \{0,1\}$; there are several ways of representing boolean functions. One option is to keep the entire truth table with 2^n entries. Alternatively, we may represent f as a circuit using $\wedge$ (and), $\vee$ (or) and $\neg$ (not) gates. We may ask that f be represented in disjunctive normal form (DNF), i.e. in the form shown below

$$(z_1 \wedge \bar{z}_3 \wedge z_7 \wedge \cdots) \vee (z_2 \wedge z_4 \wedge \bar{z}_8 \wedge \cdots) \vee \cdots \vee (z_1 \wedge z_3).$$

The choice of representation can make a huge difference in terms of the amount of memory required to store a *description* of the boolean function. You are asked to show this in the case of the parity function $f = z_1 \oplus z_2 \oplus \cdots \oplus z_n$ in Exercise 1.2. There are other possible representations of boolean functions, such as decision lists, decision trees, neural networks, etc., which we will encounter later in the course.

[7] In later chapters, we will consider different learning frameworks under which exact learning, i.e. achieving *zero* error, is possible.

[8] We shall assume that our computers can store and perform elementary arithmetic operations (addition, multiplication, division) on real numbers at unit cost.

Representation Scheme

Abstractly, a representation scheme for a concept class C is an onto function $R : \Sigma^* \to C$, where Σ is a finite alphabet.[9] Any $\sigma \in \Sigma^*$ satisfying $R(\sigma) = c$ is called a representation of c. We assume that there is a function, size : $\Sigma^* \to \mathbb{N}$, that measures the size of a representation. A concept $c \in C$ may in general have multiple representations under R. For example, there are several boolean circuits that compute exactly the same boolean function. We can define the function size on the set C by defining, $\text{size}(c) = \min_{\sigma : R(\sigma)=c} \{\text{size}(\sigma)\}$. When we refer to a concept class, we will assume by default that it is associated with a representation scheme and a size function, so that size(c) is well defined for $c \in C$. In most cases of interest, there will be a natural notion of size that makes sense for the learning problem at hand; however, some of the exercises and coloured boxes encourage you to explore the subtleties involved with representation size in greater detail.

Instance Size

Typically, instances in a learning problem also have a natural notion of size associated with them; roughly we may think of the size of an instance as the amount of memory required to store it. For example, 10×10 black and white images can be represented using 100 bits, whereas 1024×1024 colour images will require over 3 million real numbers. When faced with larger instances, we should expect that learning algorithms will require more time; at the very least they have to read the input data![10] In this course, we will only consider settings where the instance space is either $X_n = \{0,1\}^n$ or $X_n = \mathbb{R}^n$. We denote by C_n a concept class over X_n. We consider the instance space $X = \bigcup_{n \geq 1} X_n$ and the concept class $C = \bigcup_{n \geq 1} C_n$ as representing increasingly larger instances (and concepts on them).

Definition 1.3 – **PAC Learning: Take II.** *For $n \geq 1$, let C_n be a concept class over instance space X_n and let $C = \bigcup_{n \geq 1} C_n$ and $X = \bigcup_{n \geq 1} X_n$. We say that C is* PAC (take II) learnable *if there exists a learning algorithm L that satisfies the following: for every $n \in \mathbb{N}$, for every concept $c \in C_n$, for every distribution D over X_n, for every $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $\mathsf{EX}(c, D)$ and inputs n, size(c), ϵ and δ, L outputs $h \in C_n$ that with probability at least $1 - \delta$ satisfies $\text{err}(h) \leq \epsilon$. The probability is over the random examples drawn from $\mathsf{EX}(c, D)$ as well as any internal randomisation of L. The number of calls made to $\mathsf{EX}(c, D)$* (sample complexity) *must be bounded by a polynomial in n,* size(c), $\frac{1}{\epsilon}$ *and* $\frac{1}{\delta}$.

We further say that C is efficiently *PAC (take II) learnable if the running time of L is polynomial in n,* size(c), $1/\epsilon$ *and* $1/\delta$.

[9] If representing the concept requires using real numbers, such as in the case of rectangles, we may use $R : (\Sigma \cup \mathbb{R})^* \to C$. Representing a real number will assumed to be unit cost.

[10] Assuming we know how the data is stored and that we can access specific parts of the data, in certain cases learning algorithms that do not even have to read the entire data can be designed.

When learning a target concept $c \in C_n$, in general, allowing the learning algorithm resources that increase with size(c) will be necessary. Mostly, we will consider concept classes C_n over X_n for which every size(c) can be bounded for every $c \in C_n$ by some fixed polynomial function of n. Thus, *efficient* PAC learning simply requires designing algorithms that run in time polynomial in n, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

Definition 1.3 is general enough to allow for the existence of "efficient" PAC learning algorithms if an overly verbose representation scheme is chosen. For example, the class of *all boolean functions* is efficiently PAC-learnable when the representation scheme uses truth tables. On the other hand, if we represent boolean functions as *decision trees*, or *boolean circuits*, or even *boolean formulae* in disjunctive normal form (DNF), it is widely believed that the class of boolean functions is *not* efficiently PAC-learnable. We will provide some evidence for this assertion based on cryptographic assumptions and on hardness of learning in the statistical query (SQ) learning model in later chapters.

1.3 Learning Conjunctions

Having formulated a notion of learning, let us consider a second learning problem. Let $X_n = \{0,1\}^n$ represent the instance space of size n; note that each element $\mathbf{x} \in X_n$ denotes a possible assignment to n boolean variables $z_1, \ldots, z_n$; let $X = \bigcup_{n \geq 1} X_n$. Let $\mathsf{CONJUNCTIONS}_n$ denote the concept class of conjunctions over the n boolean variables $z_1, \ldots, z_n$. A *literal* is either a boolean variable z_i or its negation $\bar{z}_i$. A conjunction (sometimes also called a *term*) is simply an *and* ($\wedge$) of literals. An example conjunction φ with $n = 10$ (say) is

$$\varphi = z_1 \wedge \bar{z}_3 \wedge \bar{z}_8 \wedge z_9. \tag{1.3}$$

Formally, a conjunction over $z_1, \ldots, z_n$ can be represented by two subsets $P, N \subset [n]$. Such a pair of sets P, N represents the conjunction $\varphi_{P,N}$ defined as

$$\varphi_{P,N} = \bigwedge_{i \in P} z_i \wedge \bigwedge_{i \in N} \bar{z}_i. \tag{1.4}$$

In (1.4), the sets P and N represent the positive and negative literals that appear in the conjunction $\varphi_{P,N}$ respectively. We have not required that $P \cap N = \emptyset$; this allows us to represent a boolean function that is 0 over the entire hypercube (falsehood), e.g. as $z_1 \wedge \bar{z}_1$. Both P and N could be empty, representing an empty conjunction that is 1 over the entire boolean hypercube (tautology). Formally

$$\mathsf{CONJUNCTIONS}_n = \{\varphi_{P,N} \mid P, N \subset [n]\},$$
$$\mathsf{CONJUNCTIONS} = \bigcup_{n \geq 1} \mathsf{CONJUNCTIONS}_n.$$

When representing a conjunction over n boolean variables, each of the sets P and N can be represented by a *bit-string* of length n; as a result any

Algorithm 1.1: CONJUNCTIONS Learner

Input: n, m, access to $\mathsf{EX}(c, D)$
// initialise hypothesis conjunction with all literals
Set $h = z_1 \wedge \bar{z}_1 \wedge z_2 \wedge \bar{z}_2 \wedge \cdots \wedge z_n \wedge \bar{z}_n$
for $i = 1, \ldots, m$ **do**
    draw $(\mathbf{x}_i, y_i)$ from $\mathsf{EX}(c, D)$
    if $y_i == 1$ **then** // ignore negative examples
        for $j = 1, \ldots n$ **do**
            if $x_{i,j} = 0$ **then** // j^{th} bit of i^{th} instance is 0
                Drop z_i from h
            else // j^{th} bit of i^{th} instance is 1
                Drop $\bar{z}_i$ from h
Output: h

conjunction can be represented using a bit-string of length $2n$. As there are at least 2^n conjunctions (can you count the number of conjunctions exactly?) we should expect to need at least n bits to represent a conjunction. Thus, this representation scheme is fairly succinct. Thus, our goal is to design an algorithm that runs in time polynomial in n, $1/\epsilon$ and $1/\delta$.

Let c denote the target conjunction. The example oracle $\mathsf{EX}(c, D)$ returns examples of the form $(\mathbf{x}, y)$ where $y \in \{0, 1\}$. $y = 1$ if c evaluates to 1 (true) after assigning $z_i = x_i$ for $i = 1, \ldots, n$. In other words, $y = 1$ if $\mathbf{x}$ is a satisfying assignment of the conjunction c, and 0 otherwise.

Algorithm 1.1 is learns the concept class CONJUNCTIONS. We describe the high-level idea before giving the complete proof.

i) The algorithm begins by conservatively constructing a hypothesis h that is a conjunction of all the $2n$ possible literals. Clearly, this conjunction will always output 0 on any given input. The algorithm then makes use of data to remove harmful literals from h.

ii) The algorithm draws m independent examples $(\mathbf{x}_i, y_i)$ from the oracle $\mathsf{EX}(c, D)$; all the *negatively labelled* examples ($y_i = 0$) are ignored. For positively labelled examples, literals that would cause these to be labelled as negative by h are dropped from h. The resulting hypothesis h is returned. Thus, the algorithm outputs the "longest" conjunction (containing the most number of literals) that is consistent with the observed data. This is because only those literals that absolutely cannot be part of the target conjunction (as dictated by the positively labelled data) are dropped.

Theorem 1.4. *Provided* $m \geq \frac{2n}{\epsilon} \log\left(\frac{2n}{\delta}\right)$, *Algorithm 1.1* efficiently *PAC (take II) learns the concept class* CONJUNCTIONS.

Proof. Let c be the target conjunction and D the distribution over $\{0, 1\}^n$. For a literal ℓ (which may be z_i or $\bar{z}_i$), let $p(\ell) = \mathbb{P}_{\mathbf{x} \sim D}\left[c(\mathbf{x}) = 1 \wedge \ell(\mathbf{x}) = 0\right]$; here, we interpret ℓ itself as a conjunction with 1 literal. Thus, if $\ell = z_i$, then $\ell(\mathbf{x}) = x_i$; if $\ell = \bar{z}_i$, then $\ell(\mathbf{x}) = 1 - x_i$. Notice that if $p(\ell) > 0$, then the literal ℓ cannot be present in c; if it were, then there can be no $\mathbf{x}$ such that $c(\mathbf{x}) = 1$ and $\ell(\mathbf{x}) = 0$.

We define a literal ℓ to be *harmful* if $p(\ell) \geq \frac{\epsilon}{2n}$. We will ensure that all harmful literals are eliminated from the hypothesis h. For a harmful literal ℓ, let A_ℓ denote the event that after m independent draws from $\mathsf{EX}(c, D)$, ℓ is not eliminated from h. Note that this can only happen if no $\mathbf{x}$ such that $c(\mathbf{x}) = 1$ but $\ell(\mathbf{x}) = 0$ is drawn. This can happen with probability at most $\left(1 - \frac{\epsilon}{2n}\right)^m$. Let B denote the set of harmful literals and let $\mathcal{E} = \bigcup_{\ell \in B} A_\ell$ be the event that at least one harmful literal survives in h. We shall choose m large enough so that $\mathbb{P}[\mathcal{E}] \leq \delta$. Consider the following,

$$\begin{aligned}
\mathbb{P}[\mathcal{E}] &\leq \sum_{\ell \in B} \mathbb{P}[A_\ell] && \text{By the Union Bound (A.1).} \\
&\leq 2n \left(1 - \frac{\epsilon}{2n}\right)^m && |B| \leq 2n \text{ and for each } \ell \in B, \mathbb{P}[A_\ell] \leq \left(1 - \frac{\epsilon}{2n}\right)^m. \\
&\leq 2n \exp\left(-\frac{m\epsilon}{2n}\right). && \text{As } 1 - x \leq e^{-x} \text{ (B.1).}
\end{aligned}$$

Thus, whenever $m \geq \frac{2n}{\epsilon} \log\left(\frac{2n}{\delta}\right)$, we know that $\mathbb{P}[\mathcal{E}] \leq \delta$. Now, suppose that $\mathcal{E}$ does not occur, i.e. all harmful literals are eliminated from h. Let B^c be the set of literals that are not harmful.

$$\begin{aligned}
\mathrm{err}(h) &= \mathbb{P}_{\mathbf{x} \sim D}\left[c(\mathbf{x}) = 1 \wedge h(\mathbf{x}) = 0\right] \\
&\leq \sum_{\ell \in B^c} \mathbb{P}_{\mathbf{x} \sim D}\left[c(\mathbf{x}) = 1 \wedge \ell(\mathbf{x}) = 0\right] \\
&\leq 2n \cdot \frac{\epsilon}{2n} \leq \epsilon.
\end{aligned}$$

This completes the proof. □

It is worth pointing out that Algorithm 1.1 only makes use of positively labelled examples. The algorithm works correctly even if no positively labelled examples are obtained from the oracle $\mathsf{EX}(c, D)$; this is because if no positive examples are obtained after drawing m independent examples (for a sufficiently large m), then returning a hypothesis h that always predicts 0 is sufficient to achieve low error.

1.3.1 Learning k-CNF

We can generalise Algorithm 1.1 to learn richer classes of boolean functions. A *clause* is a disjunction ($\vee$) of boolean literals. The *length* of a clause is the number of (not necessarily distinct) literals in it. For example, $z_1 \vee \bar{z}_7 \vee \bar{z}_{15}$ is a clause of length 3. Let $\mathsf{clauses}_{n,k}$ denote the set of all clauses of length exactly k on the n boolean variables $z_1, \ldots, z_n$. We define the class of boolean functions that can be written in conjunctive normal form using clauses of length exactly k as:

$$\begin{aligned}
k\text{-}\mathsf{CNF}_n &= \{\bigwedge_i c_i \mid c_i \in \mathsf{clauses}_{n,k}\}, \\
k\text{-}\mathsf{CNF} &= \bigcup_{n \geq 1} k\text{-}\mathsf{CNF}_n.
\end{aligned}$$

A representation of boolean function as in the class k-CNF is called a k-CNF formula. There are at most $(2n)^k$ possible clauses of length k on n boolean variables and so each k-CNF formula over n variables can have at most $(2n)^k$ clauses. (Allowing clauses to have the same literal multiple times and letting the order of literals matter, we shall assume in the rest of this section that there are exactly $(2n)^k$ clauses of length k.)

It is completely straightforward to modify Algorithm 1.1 to start with a hypothesis h that is a k-CNF formula with all $(2n)^k$ clauses and eliminate the clauses that cause positive examples to be labelled negative. This algorithm is *efficient* if we assume the representation scheme to have length $(2n)^k$, and in any case the running time and sample complexity is polynomial in n for any fixed constant k.

Rather than redo the proof of Theorem 1.4, we shall sketch a different approach that also introduces the notion of a reduction between learning problems. Suppose the target function is a k-CNF formula over the boolean variables $z_1, \ldots, z_n$; we create new boolean variables $(z'_{\ell_1,\ldots,\ell_k})$ where each ℓ_i is either some z_j or $\bar{z}_j$. When placed in parentheses, $(z'_{\ell_1,\ldots,\ell_k})$ denotes the set of all possible $(2n)^k$ boolean variables; whereas by itself $z'_{\ell_1,\ldots,\ell_k}$ denotes the specific variable corresponding to the tuple of literals $(\ell_1, \ldots, \ell_k)$. The boolean variable $z'_{\ell_1,\ldots,\ell_k}$ is meant to represent the clause $\ell_1 \vee \cdots \vee \ell_k$. Given an assignment to the boolean variables $z_1, \ldots, z_n$ denoted by some bit-vector $\mathbf{x} \in \{0,1\}^n$, an assignment to $(z'_{\ell_1,\ldots,\ell_k})$ can be uniquely determined, by assigning the variable $z'_{\ell_1,\ldots,\ell_k}$ the value 1 if and only if $\mathbf{x}$ is a satisfying assignment of the clause $\ell_1 \vee \cdots \vee \ell_k$. This yields a bit vector in $\{0,1\}^{(2n)^k}$ that represents the assignment to all $(z'_{\ell_1,\ldots,\ell_k})$. Let us denote this map from $\{0,1\}^n$ to $\{0,1\}^{(2n)^k}$ by f and observe that it is injective.

Now consider the following "natural" bijective map, denoted by g, between k-CNF formulae over $z_1, \ldots, z_n$ and *monotone conjunctions* over $(z'_{\ell_1,\ldots,\ell_k})$: given a k-CNF formula φ, the literal $z'_{\ell_1,\ldots,\ell_k}$ appears in the monotone conjunction $g(\varphi)$ if and only if φ contains the clause $\ell_1 \vee \cdots \vee \ell_k$.[11] (A conjunction is *monotone* if it does not contain any *negated* literals; Algorithm 1.1 modified to start with $h = z_1 \wedge \cdots \wedge z_n$ clearly learns the class of *monotone conjunctions*.)

Let D be a distribution over $\{0,1\}^n$ and let $f(D)$ denote the distribution over $\{0,1\}^{(2n)^k}$ obtained by first drawing $\mathbf{x}$ according to D and then applying f to $\mathbf{x}$. Let $c, h \in k\text{-CNF}_n$, then it can be easily verified that

$$\mathrm{err}(h; c, D) = \mathrm{err}(g(h); g(c), f(D)).$$

The only thing that remains is to observe that the maps f, g and g^{-1} are (trivially) polynomial time computable and that given access to $\mathsf{EX}(c, D)$, the hypothetical example oracle $\mathsf{EX}(g(c), f(D))$ can be simulated in polynomial time. Thus, we have proved the following result.

Theorem 1.5. *The concept class k-CNF is efficiently PAC (take II) learnable.*

[11] We treat this map as purely syntactic. In particular, for truth assignments the order of the variables does not matter; however, for the purpose of the map g, the 2-CNF formulae $(x_1 \vee x_2) \wedge (x_3 \vee x_4)$ and $(x_2 \vee x_1) \wedge (x_4 \vee x_3)$ would be mapped to the (distinct) conjunctions, $z'_{z_1,z_2} \wedge z'_{z_3,z_4}$ and $z'_{z_2,z_1} \wedge z'_{z_4,z_3}$ respectively.

1.4 Hardness of Learning 3-term DNF

Having seen a few examples of concept classes that are PAC (take II) learnable, we shall temper our optimism by proving that a class of boolean functions (not significantly more complex than CONJUNCTIONS) is not PAC (take II) learnable, assuming an unproven, but widely believed, conjecture from computational complexity theory. The class is that of boolean functions that can be expressed as DNF formulae with exactly 3 terms. A term is simply a conjunction over n boolean variables $z_1, \ldots, z_n$. Formally, the class is defined as

$$\text{3-TERM-DNF}_n = \{T_1 \vee T_2 \vee T_3 \mid T_i \in \text{CONJUNCTIONS}_n\},$$
$$\text{3-TERM-DNF} = \bigcup_{n \geq 1} \text{3-TERM-DNF}_n.$$

Note that any DNF formula with 3 terms can be expressed as a bit-string of length at most $6n$—there are three terms, each of which is a boolean conjunction expressible by a boolean string of length $2n$; as a result, the representation size for each $c \in \text{3-TERM-DNF}_n$ can be bounded by $6n$. Thus, an efficient algorithm for learning 3-TERM-DNF needs to run in time polynomial in n, $1/\epsilon$ and $1/\delta$. The next result shows that such an algorithm is, in fact, unlikely to exist. Formally, we'll prove the following theorem.

Theorem 1.6. 3-TERM-DNF *is not efficiently PAC (take II) learnable unless* RP = NP.

Let us first discuss the condition "unless RP = NP". We will briefly define the class RP here, but those unfamiliar with (randomised) complexity classes may wish to refer to standard texts on complexity theory (cf. Chapter Notes in Section 1.8). The class RP consists of languages for which membership can be determined by a randomised polynomial time algorithm that errs on only one side. More formally, a language $L \in \text{RP}$, if there exists a randomised polynomial time algorithm A that satisfies the following

- For string $\sigma \notin L$, $A(\sigma) = 0$
- For string $\sigma \in L$, $A(\sigma) = 1$ with probability at least $1/2$.

The rest of this section is devoted to prove Theorem 1.6. We shall reduce the decision problem for an NP-complete language to the problem of PAC (take II) learning 3-TERM-DNF. Suppose L is a language that is NP-complete. Given an instance (string) σ we wish to decide whether $\sigma \in L$. We will construct a training sample, a set of positive instances S_+ and negative instances S_-, where S_+ and S_- are disjoint. We will show that there exists a 3-term DNF formula φ such that all instances in S_+ are satisfying assignments of φ and that none of the instances in S_- satisfy φ, if and only if $\sigma \in L$. We will ensure that $|S_+ \cup S_-|$ is bounded by some polynomial in $|\sigma|$, and that each example is also of size bounded by a polynomial in $|\sigma|$, so that the reduction is polynomial time. Here $|\sigma|$ simply denotes the length of the instance (string) σ.

Let us see how an efficient algorithm that PAC (take II) learns 3-TERM-DNF can be used to test whether or not $\sigma \in L$. Let $S = S_+ \cup S_-$, where S_+ and S_- are the sets as constructed above, and let D be a distribution that is uniform over S, i.e. a distribution that assigns probability mass $\frac{1}{|S|}$ to every

instance that appears in S, and 0 mass to all other instances. Let $\epsilon = \frac{1}{2|S|}$ and $\delta = 1/2$. Now, let us suppose that $\sigma \in L$, then indeed there does exist 3-term DNF formula, φ, that is consistent with the sample S. So we can simulate a valid example oracle $\mathsf{EX}(\varphi, D)$, by simply returning a random example $(\mathbf{x}, y)$ where $\mathbf{x}$ is chosen uniformly at random from S, and $y = 1$ if $\mathbf{x} \in S_+$, and $y = 0$ otherwise. By the PAC (take II) learning guarantee, with probability at least $1/2$, the algorithm returns $h \in$ 3-TERM-DNF, such that $\mathrm{err}(h) \leq \frac{1}{2|S|}$. However, as there are only $|S|$ instances in S and the distribution is uniform, it must be that h correctly predicts the labels of all instances in S, which implies $\sigma \in L$. Notice that given h, it can easily be checked in polynomial time that h indeed correctly predicts the labels for all instances in S.

On the other hand, if $\sigma \notin L$, there is no 3-term DNF formula that correctly assigns labels to the instances in S. Hence, the learning algorithm cannot output such an $h \in$ 3-TERM-DNF. Again, given the output hypothesis h, checking whether h correctly labels all the instances in S or not, can be easily done in polynomial time. Thus, assuming an efficient PAC (take II) learning algorithm for 3-TERM-DNF exists, we also have a randomised algorithm to solve the decision problem for the NP-complete language L. This in turn implies that RP = NP, something that is widely believed to be untrue.

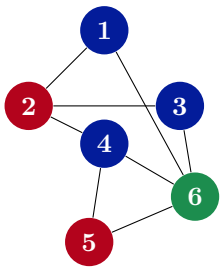
All that is left to do is to identify a suitable NP-complete language and show how to construct a sample S with the desired property. In this case, we will use the fact that *graph* 3-*colouring* is NP-complete.

Graph 3-Colouring reduces to PAC (Take II) Learning 3-TERM-DNF

The language 3-COLOURABLE consists of representations of graphs that can be 3-*coloured*. We say a graph is 3-*colourable* if there is an assignment from the vertices to the set of three colours, $\{r, g, b\}$, such that no two adjacent vertices are assigned the same colour. As already discussed, given a graph G, we only need to produce disjoint sets S_+ and S_- of instances that are positively and negatively labelled respectively, such that the graph G is 3-colourable if and only if there exists a 3-term DNF formula that correctly predicts the labels of all instances in $S_+ \cup S_-$.

For notational convenience, in this section, we will denote the instances as $v(i)$ and $e(i, j)$ rather than the more usual $\mathbf{x}$. Suppose G has n vertices. For vertex $i \in G$, we let $v(i) \in \{0, 1\}^n$ that has a 1 in every position except i. For an edge (i, j) in G, we let $e(\{i, j\}) \in \{0, 1\}^n$ that has a 1 in all positions except i and j. Let $S_+ = \{v(i) \mid i \text{ a vertex of } G\}$ and $S_- = \{e(\{i, j\}) \mid \{i, j\} \text{ an edge of } G\}$; clearly S_+ and S_- are disjoint. Figure 1.3 shows an example of a graph that is 3-colourable along with the sets S_+ and S_-.

First, suppose that G is 3-colourable. Let V_r, V_g, V_b be the set of vertices of G that are labelled red (r), blue (b) and green (g) respectively in some valid 3-colouring. Let $z_1, \ldots, z_n$ denote the n boolean variables (one corresponding to each vertex of G). Let $T_r = \bigwedge_{i \notin V_r} z_i$. T_g and T_b are defined similarly. Consider the 3-term DNF formula $\varphi = T_r \vee T_g \vee T_b$; we will show that all instances in S_+ satisfy φ and that none of the instances in S_- do. First consider $v(i) \in S_+$. Without loss of generality, suppose i is coloured red, i.e. $i \in V_r$. Then, we claim that $v(i)$ is a satisfying assignment of T_r and hence also of φ. Clearly, the literal z_i is not contained in T_r and there are no negative literals in T_r. Since all the bits of $v(i)$ other than the i^{th} position are 1, $v(i)$ is a satisfying



(a) Graph

$v(1)$	$(0,1,1,1,1,1)$
$v(2)$	$(1,0,1,1,1,1)$
$v(3)$	$(1,1,0,1,1,1)$
$v(4)$	$(1,1,1,0,1,1)$
$v(5)$	$(1,1,1,1,0,1)$
$v(6)$	$(1,1,1,1,1,0)$

(b) Positive Examples

$e(\{1,2\})$	$(0,0,1,1,1,1)$
$e(\{1,6\})$	$(0,1,1,1,1,0)$
$e(\{2,3\})$	$(1,0,0,1,1,1)$
$e(\{2,4\})$	$(1,0,1,0,1,1)$
$e(\{3,6\})$	$(1,1,0,1,1,0)$
$e(\{4,5\})$	$(1,1,1,0,0,1)$
$e(\{4,6\})$	$(1,1,1,0,1,0)$
$e(\{5,6\})$	$(1,1,1,1,0,0)$

(c) Negative Examples

Figure 1.3: (a) A graph G along with a valid three colouring. (b) Positive examples of the sample generated using G. (c) Negative examples of the sample generated using G.

assignment of T_r. Now, consider $e(\{i,j\})$. We claim that $e(\{i,j\})$ is not a satisfying assignment of any of T_r, T_g or T_b and hence it also does not satisfy φ. For a colour $c \in \{r,g,b\}$, either i is not coloured c or j isn't. Suppose i is the one that is not coloured c, then T_c contains the literal z_i, but the i^{th} bit of $e(\{i,j\})$ is 0 and so $e(\{i,j\})$ is not a satisfying assignment of T_c. This argument applies to all colours and hence $e(\{i,j\})$ is not a satisfying assignment of φ. This completes the "if" part of the proof.

Next, suppose that $\varphi = T_r \vee T_g \vee T_b$ is a 3-term DNF such that all instances in S_+ are satisfying assignments of φ and none in S_- are. We use φ to assign colours to the vertices of G that represent a valid 3-colouring. For a vertex i, since $v(i)$ is a satisfying assignment of φ, it is also a satisfying assignment of at least one of T_r, T_g or T_b. We assign it a colour based on the term for which it is a satisfying assignment (ties may be broken arbitrarily). Since for every vertex i, there exists $v(i) \in S_+$, this ensures that every vertex is assigned a colour. Next, we need to ensure that no two adjacent vertices are assigned the same colour. Suppose there is an edge $\{i,j\}$ such that i and j are assigned the same colour. Without loss of generality, suppose that this colour is red (r). Since we know that $e(\{i,j\})$ is not a satisfying assignment of φ, $e(\{i,j\})$ also does not satisfy T_r. Also, as i and j were both coloured red, $v(i)$ and $v(j)$ do satisfy T_r. This implies that the literals z_i and z_j are not present in T_r. The fact that $v(i)$ satisfies T_r ensures that the literal $\bar{z}_k$ for any $k \neq i$ cannot appear in T_r. However, if T_r does not contain any negated literal, other than possibly z_i, and if it does not contain the literals z_i and z_j, then $e(\{i,j\})$ satisfies T_r and hence φ, a contradiction. Hence, there cannot be two adjacent vertices that have been assigned the same colour. This completes the proof of the "only if" part and with it also the proof of Theorem 1.6.

1.5 Learning 3-CNF vs 3-TERM-DNF

In Section 1.3.1, we proved that the concept class k-CNF, and hence 3-CNF, is *efficiently* PAC (take II) learnable. On the other hand, Theorem 1.6 shows that under the widely believed assumption that RP $\neq$ NP, the class 3-TERM-DNF is not *efficiently* PAC (take II) learnable. Let us recall the distributive law of

boolean operations

$$(a \wedge b) \vee (c \wedge d) \equiv (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d). \tag{1.5}$$

By applying the rule (1.5), we can express any $\varphi \in$ 3-TERM-DNF as some $\psi \in$ 3-CNF.

$$\varphi = T_1 \vee T_2 \vee T_3 \equiv \bigwedge_{\substack{\ell_1 \in T_1 \\ \ell_2 \in T_2 \\ \ell_3 \in T_3}} (\ell_1 \vee \ell_2 \vee \ell_3) = \psi$$

For any distribution D over $X_n = \{0,1\}^n$, the example oracles $\mathsf{EX}(\varphi, D)$ and $\mathsf{EX}(\psi, D)$ are indistinguishable. Thus, if we use a PAC (take II) learning algorithm for 3-CNF that outputs some $h \in$ 3-CNF, with probability at least $1 - \delta$, we will have

$$\mathrm{err}(h; \varphi, D) = \mathrm{err}(h; \psi, D) \leq \epsilon.$$

What this suggests is that if our goal is simply to *predict* as well as the target concept $\varphi \in$ 3-TERM-DNF, then there is no impediment (in terms of statistical or computational resources) to doing so. The difficulty arises because our definition of PAC (take II) learning requires us to express the output hypothesis as a 3-term DNF formula. Arguably from the point of view of learning, being able to *predict* labels correctly is more important than the exact hypothesis we use to do so. Our final definition of PAC learning in Section 1.6 will allow learning algorithms to output hypothesis that do not belong to the concept class being learnt. We will still need to put some restrictions on what is allowable as an output hypothesis; you are asked to explore the implications of loosening these requirements further in Exercise 2.4. It may also be the case that the computational savings (being able to run in polynomial time) come at a statistical cost, something we will explore in greater detail after having seen some general methods for designing learning algorithms.[12]

1.6 PAC Learning

In our final definition of PAC learning, we shall remove the requirement that the output hypothesis actually belongs to the concept class being learnt. We then have to specify in what form an algorithm may output a hypothesis. As was the case with concept classes, we can define a hypothesis class H_n over the instances X_n (implicitly we assume that there is also a representation scheme for H_n and an associated size function), and consider the hypothesis class $H = \bigcup_{n \geq 1} H_n$. We will wish to place some restrictions on the hypothesis class. (To explore why see Exercise 2.4.) The requirement we add is that the hypothesis class H be *polynomially evaluatable*.

Definition 1.7 **– Polynomially Evaluatable Hypothesis Class.** *A hypothesis class H is* polynomially evaluatable *if there exists an algorithm that on input any instance $\mathbf{x} \in X_n$ and any representation $h \in H_n$, outputs the value $h(\mathbf{x})$ in time polynomial in n and* size(h).

[12]The word "may" has been used in the above sentence because the claim is based only on different upper bounds on the sample complexity of *efficient* learning algorithms. No "non-trivial" lower bound on the sample complexity for a polynomial time algorithm for learning 3-TERM-DNF is known. This will be discussed in greater detail in Chapter 2.

In words, the requirement that H be polynomially evaluatable demands that given the description of the "program" encoding the prediction rule, h, and an instance, $\mathbf{x}$, we should be evaluate $h(\mathbf{x})$ in a reasonable amount of time. Here reasonable means polynomial in the input, i.e. size(h) and $\mathbf{x}$. We now give the final definition of PAC learning and then end by making a few observations.

***Definition 1.8* – PAC Learning.** *For $n \geq 1$, let C_n be a concept class over instance space X_n and let $C = \bigcup_{n\geq 1} C_n$ and $X = \bigcup_{n\geq 1} X_n$. We say that C is* PAC learnable using the hypothesis class H *if there exists an algorithm L that satisfies the following: for every $n \in \mathbb{N}$, for every concept $c \in C_n$, for every distribution D over X_n, for every $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $\mathsf{EX}(c, D)$ and inputs n,* size(c)*, ϵ and δ, L outputs $h \in H_n$ that with probability at least $1-\delta$ satisfies* $\mathrm{err}(h) \leq \epsilon$*. The probability is over the random examples drawn from $\mathsf{EX}(c, D)$ as well as any internal randomisation of L. The number of calls made to $\mathsf{EX}(c, D)$* (sample complexity) *must be bounded by a polynomial in n,* size(c)*, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ and H must be polynomially evaluatable.*

We further say that C is efficiently *PAC learnable using H, if the running time of L is polynomial in n,* size(c)*, $1/\epsilon$ and $1/\delta$.*

Some comments regarding the definition of PAC Learning

i) For efficient PAC learning, although no explicit restriction is put on what size(h) can be, the requirement on the running time of the algorithm ensures that size(h) itself must be bounded by a polynomial in n, size(c), $\frac{1}{\delta}$ and $\frac{1}{\epsilon}$.

ii) When H is not explicitly specified, by *efficient* PAC learning C, we mean that there exists some polynomially evaluatable hypothesis class H, such that C is *efficiently* PAC learnable using H.

iii) In terms for our final definition of PAC learning, PAC (take II) learning C refers to PAC learning C using C. When *efficiency* is a consideration, the learning algorithm has to be efficient and C itself needs to be polynomially evaluatable. In the literature (and in the rest of this course), (efficient) PAC (take II) learning is referred to as (efficient) *proper* PAC learning. Sometimes to distinguish PAC learning from *proper* PAC learning, the word *improper* is added in front of PAC learning.

iv) In the definition of PAC learning (all of them), we do require that the number of calls to $\mathsf{EX}(c, D)$ is bounded by a polynomial in n, size(c), $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. This corresponds to the *sample complexity* or the amount of data used by the learning algorithm. Even when we allow *inefficient* algorithms, we do require the amount of data used to be modest; this is mainly to capture the idea that *automated learning* is about learning the target function using a modest amount of data. When arbitrary computational power is permitted, there is not much to be gained from using more data; this follows from Exercise 2.4.

1.7 Exercises

1.1 This question is about the rectangle learning problem.

a) Modify the analysis of the rectangle learning algorithm to work in the case that D is an arbitrary probability distribution over $\mathbb{R}^2$.

b) The concept class of *hyper-rectangles* over $\mathbb{R}^n$ is defined as follows

$$\mathsf{RECTANGLES}_n = \{\mathbb{1}_{[a_1,b_1]\times\cdots\times[a_n,b_n]} \mid a_i, b_i \in \mathbb{R}, a_i < b_i\}.$$

For a set $S \subset \mathbb{R}^n$, the notation $\mathbb{1}_S$ represents its indicator, i.e. the boolean function that is 1 if $x \in S$ and 0 otherwise. Generalise the algorithm for learning rectangles in $\mathbb{R}^2$ and show that it *efficiently* PAC learns the class of hyper-rectangles. Give bounds on the number of examples required to guarantee that with probability at least $1-\delta$, the error of the output hypothesis at most ϵ. The sample complexity and running time of your algorithm should be polynomial in n, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

1.2 Let $f : \{0,1\}^n \to \{0,1\}$ be the parity function on the n bits, i.e. $f(z_1, \ldots, z_n) = z_1 \oplus z_2 \oplus \cdots \oplus z_n$. In words, when given n bits as input, f evaluates to 1 if and only if an odd number of the input bits are 1.

a) A boolean circuit with n inputs is represented by an annotated directed acyclic graph with exactly n source nodes and 1 sink node. The source nodes contain the inputs $z_1, \ldots, z_n$ (at the time of evaluation each z_i is assigned a value in $\{0,1\}$). Each internal node is labelled with either $\wedge$, $\vee$, or $\neg$; internal nodes labelled by $\wedge$ or $\vee$ have in-degree exactly 2 and internal nodes labelled by $\neg$ have in-degree exactly 1. The nodes labelled by $\wedge$, $\vee$ and $\neg$, compute the logical *and*, *or*, and *not*, of their inputs (the values at the one or two nodes that feed into them) respectively. The sink node represents the output of the circuit which will be either 0 or 1. The *size* of a boolean circuit is defined to be the number of edges in the directed acyclic graph that represents the circuit; the depth of a circuit is the length of the *longest* path from a source node to the sink node. Show that f can be represented as a boolean circuit of size $O(n)$ and depth $O(\log n)$.

b) Show that representing f in disjunctive normal form (DNF) requires at least 2^{n-1} terms.

1.3 Say that an algorithm L *perhaps learns* a concept class C using hypothesis class H, if for every n, for every concept $c \in C_n$, for every distribution D over X_n and for every $0 < \epsilon < 1/2$, L given access to $\mathsf{EX}(c, D)$ and inputs ϵ and size(c), runs in time polynomial in n, size(c) and $1/\epsilon$, and outputs a polynomially evaluatable hypothesis $h \in H_n$, that with probability at least $3/4$ satisfies $\mathrm{err}(h) \leq \epsilon$. In other words, we've set $\delta = 1/4$ in the definition of *efficient* PAC learning. Show that if C is "perhaps learnable" using H, then C is also *efficiently* PAC learnable using H.

1.4 Consider the question of learning boolean threshold functions. Let $X_n = \{0,1\}^n$ and for $\mathbf{w} \in \{0,1\}^n$ and $k \in \mathbb{N}$, $f_{\mathbf{w},k} : X_n \to \{0,1\}$ is a boolean

threshold function defined as follows:

$$f_{\mathbf{w},k}(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i \cdot x_i \geq k \\ 0 & \text{otherwise} \end{cases}$$

Define the concept class of threshold functions as

$$\mathsf{THRESHOLDS}_n = \{f_{\mathbf{w},k} \mid \mathbf{w} \in \{0,1\}^n, 0 \leq k \leq n\},$$
$$\mathsf{THRESHOLDS} = \bigcup_{n \geq 1} \mathsf{THRESHOLDS}_n.$$

Prove that unless $\mathsf{RP} = \mathsf{NP}$, there is no *efficient proper* PAC learning algorithm for $\mathsf{THRESHOLDS}$.

1.8 Chapter Notes

Material in this lecture is almost entirely adopted from Kearns and Vazirani [34, Chap. 1]. The original PAC learning framework was introduced in a seminal paper by Valiant [43].

While we will not make heavy use of deep results from Computational Complexity theory, acquaintance with basic concepts such as NP-completeness, will be necessary. Better understanding of computational complexity will also be beneficial to understand hardness of learning based on the $\mathsf{RP} \neq \mathsf{NP}$ conjecture and other conjectures from cryptography. The classic text by Papadimitriou [40], and the more recent book by Arora and Barak [6], are excellent resources for students wishing to read up further on computational complexity theory.

Chapter 2

Consistent Learning and Occam's Razor

In the previous chapter, we studied a few different learning algorithms. Both the design and the analysis of those algorithms was somewhat *ad hoc*, based on first principles. In this chapter, we'll begin to develop tools that will serve as general methods to design learning algorithms and analyse their performance.

2.1 Occam's Razor

In the first part of this chapter, we'll study an *explanatory framework* for learning. In the PAC learning framework, what is important is a guarantee that, with high probability, the output hypothesis performs well on unseen data, i.e. fresh data drawn from the target distribution D. Here we consider the following question: Given $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)$, where $\mathbf{x}_i \in X_n$ and $y_i \in \{0, 1\}$, can we find some hypothesis, $h : X_n \to \{0, 1\}$ that is consistent with the observed data, i.e. for all i, $h(\mathbf{x}_i) = y_i$.[1]

If there is no restriction on the output hypothesis, then this can be simply achieved by *memorising* the data. In particular, one could output a program of the form, *"if* $\mathbf{x} = \mathbf{x}_1$*, output* y_1*, else if* $\mathbf{x} = \mathbf{x}_2$*, output* y_2*, ..., else if* $\mathbf{x} = \mathbf{x}_m$*, output* y_m*, else output* 0*"*. This output hypothesis is correct on all of the observed data and predicts 0 on all other instances. Clearly, we would not consider this as a form of learning. The basic problem here is that the "explanation" of the data is as long as the data itself. Even if one tries to rule out programmes of this kind, it is easy to see that simple concept classes are rich enough to *essentially* memorise the data (cf. Exercise 2.1).

The condition that we want to impose is that the explanation of the data be *succinct*, at the very least, shorter than the length of the data itself. In computational learning theory, this is referred to as the *Occam Principle* or *Occam's Razor*, named after the medieval philosopher and theologian, William of Ockham, who expounded the principle that "explanations should be not made unnecessarily complex".[2]

[1] In order to avoid absurdities, we will assume that for all $1 \leq i, j \leq m$, it is not the case that $\mathbf{x}_i = \mathbf{x}_j$, but $y_i \neq y_j$.

[2] This is by no means a wholly accurate depiction of the writings of William of Ockham. Those interested in the history are encouraged to look up the original work.

Philosophical Implications*

The notion of *succinct explanations* can be formalised in several ways and has deep connections to various areas of mathematics and philosophy. There are connections to Kolmogorov complexity which leads to the *minimum description length* (MDL) principle. The MDL principle itself can be given a Bayesian interpretation of assigning a larger prior probability to shorter hypotheses. The existence of a short description also implies existence of compression schemes. We will not discuss these issues in detail in this course; the interested student is referred to the following sources as a starting point [23, 30, 28].

Typically, finding the *shortest hypothesis* consistent with the data may be intractable or even uncomputable. In order to get useful results out of this principle, we do not need to find the shortest description or achieve optimal compression. It turns out that it is enough for the description of the output hypothesis to be slightly shorter than the amount of data observed. We'll formalise this notion to derive PAC-learning algorithms from explanatory hypotheses.

2.2 Consistent Learning

We'll first define the notion of a consistent learning algorithm, or consistent learner, for a concept class C.[3]

Definition 2.1 – **Consistent Learner.** *We say that a learning algorithm L is a consistent learner for a concept class C using hypothesis class H, if for all $n \geq 1$, for all $c \in C_n$ and for all $m \geq 1$, given as input the sequence of examples, $(\mathbf{x}_1, c(\mathbf{x}_1)), (\mathbf{x}_2, c(\mathbf{x}_2)), \ldots, (\mathbf{x}_m, c(\mathbf{x}_m))$, where each $\mathbf{x}_i \in X_n$, L outputs $h \in H_n$ such that for $i = 1, \ldots, m$, $h(\mathbf{x}_i) = c(\mathbf{x}_i)$. We say that L is an* efficient consistent learner *if the running time of L is polynomial in n,* size(c) *and m. Furthermore, we shall say that a concept class C is (efficiently) consistently learnable, if there exists a learning algorithm L and a polynomially-evaluatable hypothesis class H, such that L is an (efficient) consistent learner for C using H.*

A consistent learning algorithm is simply required to output a (polynomially evaluatable) hypothesis that is consistent with all the training data provided to it. So far, we have not imposed any requirement on the hypothesis class H. This notion of *consistency* is closely related to the empirical risk minimisation (ERM) principle in the statistical machine learning literature, when the risk is defined using the *zero-one* loss.

The main result we will prove is that if H is "small enough", something that is made precise in the theorem below, then a consistent learner can be used to derive a PAC-learning algorithm. This theorem shows that short explanatory hypotheses do in fact also possess predictive power.

Theorem 2.2 – **Occam's Razor, Cardinality Version.** *Let C be a concept class and H a hypothesis class. Let L be a consistent learner for C using H.*

[3] Starting from this chapter, we will avoid the cumbersome notation of treating a concept class C as $C = \cup_{n \geq 1} C_n$ (likewise $X = \cup_{n \geq 1} X_n$ and $H = \cup_{n \geq 1} H_n$) and shall assume that this is implicitly the case. Where confusion may arise we shall continue to be fully explicit about concept classes that contain concepts defined over instance spaces of increasing sizes.

In statistical machine learning, the general setting is where the *inputs* to the target function come from some space X (which in this course we refer to as the instance space) and the outputs come from some set Y. The case where $Y = \{0, 1\}$ corresponds to binary classification problems, such as the ones we are considering in this course, but in general Y can be other sets. The data is assumed to come from some distribution over $X \times Y$.

A class of hypotheses H consists of functions $h : X \to Y'$, where typically $Y \subseteq Y'$. There is a loss function, $\ell : Y' \times Y \to \mathbb{R}^+$ that indicates the *loss* incurred by outputting $y' \in Y'$, when the true output was $y \in Y$. The risk of a hypothesis with respect to a loss function ℓ and a data distribution D over $X \times Y$ is defined as

$$R(h) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D} \left[\ell(h(\mathbf{x}), y)\right]. \tag{2.1}$$

The *empirical risk* on a sample S of size m drawn from D is

$$\widehat{R}(h) = \frac{1}{m} \sum_{(\mathbf{x},y)\in S} \ell(h(\mathbf{x}), y). \tag{2.2}$$

To be more precise, we should use the notation $R_D(h)$ and $\widehat{R}_S(h)$, however, unless there is possibility of confusion, we shall drop these subscripts. The Empirical Risk Minimisation (ERM) principle suggests that a learning algorithm should pick a hypothesis $h \in H$ that minimises the empirical risk. So far in this course, we have restricted attention to binary classification with $Y = Y' = \{0, 1\}$ and the so-called *zero-one* loss, $\ell(y', y) = \mathbb{1}(y' \neq y)$. In the language of statistical learning, the *realisable* setting is the one where there exists $h \in H$ which has 0 risk; in this case ERM is equivalent to *consistent learning*, and Theorem 2.2 can be applied. We will make further connections to the ERM principle to topics covered in this course in later chapters.

Then for all $n \geq 1$, for all $c \in C_n$, for all D over X_n, for all $0 < \epsilon < 1/2$ and all $0 < \delta < 1/2$, if L is given a sample of size m drawn from $\mathsf{EX}(c, D)$, such that,

$$m \geq \frac{1}{\epsilon}\left(\log |H_n| + \log \frac{1}{\delta}\right), \tag{2.3}$$

then L is guaranteed to output a hypothesis $h \in H_n$ that with probability at least $1 - \delta$, satisfies $\mathrm{err}(h) \leq \epsilon$.

If furthermore, L is an efficient consistent learner, $\log |H_n|$ is polynomial in n and $\mathrm{size}(c)$, and H is polynomially evaluatable, then C is efficiently PAC-learnable using H.

Proof. Fix a target concept $c \in C_n$ and the target distribution D over X_n. Call a hypothesis, $h \in H_n$ "bad" if $\mathrm{err}(h) \geq \epsilon$. Let A_h be the event that

m independent examples drawn from $\mathsf{EX}(c, D)$ are all consistent with h, i.e. $h(\mathbf{x}_i) = c(\mathbf{x}_i)$, for $i = 1, \ldots, m$. Then, if h is bad, $\mathbb{P}[A_h] \leq (1-\epsilon)^m \leq e^{-\epsilon m}$.

Consider the event,

$$\mathcal{E} = \bigcup_{h \in H_n : h \text{ bad}} A_h$$

Then, by a simple application of the union bound (A.1), we have,

$$\mathbb{P}[\mathcal{E}] \leq \sum_{h \in H_n : h \text{ bad}} \mathbb{P}(A_h) \leq |H_n| \cdot e^{-\epsilon m}$$

Thus, whenever m is larger than the bound given in the statement of the theorem, except with probability δ, no "bad" hypothesis is consistent with m random examples drawn from $\mathsf{EX}(c, D)$. However, any hypothesis that is not "bad", satisfies $\mathrm{err}(h) \leq \epsilon$ as required. □

Remark 2.3. *The version of the theorem described above only allows H_n to depend on C_n and n. It is possible to have a much more general version, where instead we consider the hypothesis class $H_{n,m}$ where a consistent learner when given m examples outputs some $h \in H_{n,m}$. As long as $\log |H_{n,m}|$ can be bounded by $\mathrm{poly}(n, \mathrm{size}(c), \frac{1}{\epsilon}, \frac{1}{\delta}) \cdot m^{\beta}$ and for some $\beta < 1$, a PAC-learning algorithm can still be derived from a consistent learner. Exercise 2.2 asks to you prove this more general result. The proof for this version appears in the book by Kearns and Vazirani [34, Chap. 2].*

2.3 Improved Sample Complexity

Learning CONJUNCTIONS

Let us revisit some of the learning algorithms we've seen so far. We derived an algorithm for learning conjunctions. At the heart of the algorithm was, in fact, a consistent learner, obtained only using positive examples. Thus, for the conjunction learning algorithm $C_n = H_n$. Note that the number of conjunctions on n literals is 3^n (each variable may appear as a positive literal, negative literal, or not at all).

Our analysis of the conjunction learning algorithm showed that if the number of examples drawn from $\mathsf{EX}(c, D)$ was at least $\frac{2n}{\epsilon}\left(\log(2n) + \log\frac{1}{\delta}\right)$, the output hypothesis with high probability has error at most ϵ. Theorem 2.2 shows that in fact even a sample of size $\frac{1}{\epsilon}\left(n \log 3 + \log\frac{1}{\delta}\right)$ would suffice.

Learning 3-TERM-DNF

Let us now consider the question of learning 3-TERM-DNF. We have shown that finding a 3-term DNF formula φ that is consistent with a given sample is NP-complete. On the other hand, we saw that it is indeed possible to find a 3-CNF formula that is consistent with a given sample. Let us compare the sample complexity bounds given by Theorem 2.2 in both of these cases. In order to do that we need good bounds on $|\text{3-TERM-DNF}_n|$ and $|\text{3-CNF}_n|$. Any 3-TERM-DNF formula can be encoded using at most $6n$ bits, each term (or a conjunction) can be represented by a bit string of length $2n$ to indicate whether

a variable appears as a positive literal, negative literal, or not at all. Thus, $|\text{3-TERM-DNF}_n| \leq 2^{6n}$.

Similarly, there are $(2n)^3$ possible clauses with three literals. Thus, each 3-CNF formula can be represented by a bit string of length $(2n)^3$, indicating for each of the possible clauses whether they are present in the formula or not. Thus, $|\text{3-CNF}_n| \leq 2^{8n^3}$. It is also not hard to show that $|\text{3-CNF}_n| \geq 2^{\kappa n^3}$ for some universal constant $\kappa > 0$. Thus, it is the case that $\log |\text{3-CNF}_n| = \Omega(n^3)$. Thus, in order to use a consistent learner that outputs a 3-CNF formula, we need a sample that has size $\Omega\left(\frac{n^3}{\epsilon}\right)$;[4] on the other hand if we had unbounded computational resources and could solve the NP-complete problem of finding a 3-term DNF consistent with a sample, then a sample of size $O\left(\frac{n}{\epsilon}\right)$ is sufficient to guarantee a hypothesis with error at most ϵ (assuming δ is constant). This suggests that there may be tradeoff between running time and sample complexity. However, it does not rule out that there may be another computationally efficient algorithm for learning 3-TERM-DNF that has a better bound in terms of sample complexity. This question is currently open.

2.4 Exercises

2.1 Given $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)$, such that $\mathbf{x}_i \in X_n$ and $y_i \in \{0, 1\}$, and for all $1 \leq i, j \leq m$, it is not the case that $\mathbf{x}_i = \mathbf{x}_j$, but $y_i \neq y_j$, show that there is a DNF formula of length $O(m)$ that is consistent with the observed data.

2.2 Formulate Remark 2.3 as a precise mathematical statement and prove it. Observe that when $H_{n,m}$ instead of H_n is used in Equation (2.3), m appears on both sides of the equation. You should justify that there exists m that is still polynomial in n, size(c), $\frac{1}{\delta}$ and $\frac{1}{\epsilon}$ that satisfies the modified form of Equation (2.3).

2.3 Let $X_n = \{0, 1\}^n$ be the instance space. A parity function, χ_S, over X_n is defined by some subset $S \subseteq \{1, \ldots, n\}$, and takes the value 1 if and odd number of the input literals in the set $\{z_i \mid i \in S\}$ are 1 and 0 otherwise. For example, if $S = \{1, 3, 4\}$, then the function $\chi_S(z_1, \ldots, z_n) = z_1 \oplus z_3 \oplus z_4$ computes the parity on the subset $\{z_1, z_3, z_4\}$. Note that any such parity function can be represented by a bit string of length n, by indicating which indices are part of S. Let PARITIES_n denote the concept class consisting of all 2^n parity functions; observe that the the concept class PARITIES_n has representation size at most n. Show that the class PARITIES, defined as $\text{PARITIES} = \bigcup_{n \geq 1} \text{PARITIES}_n$, is *efficiently proper* PAC learnable. You should clearly describe a learning algorithm, analyse its running time and prove its correctness.

2.4 Recall that in the definition of PAC-learning, we require that the hypothesis output by the learning algorithm be evaluatable in polynomial time. Suppose we relax this restriction, and let H be the class of all Turing machines (not necessarily polynomial time)—so the output of the learning

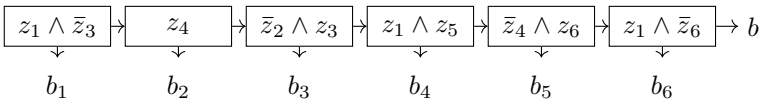
[4] At the very least, this is the lower bound we get if we apply Theorem 2.2. We will see shortly that in fact this is a lower bound on sample complexity for learning 3-CNF, no matter what algorithm is used.

algorithm can be any program. Let C_n be the class of all boolean circuits of size at most $p(n)$ for some fixed polynomial p and having n boolean inputs. Show that $C = \bigcup_{n \geq 1} C_n$ is PAC-learnable using H (under this modified definition). Argue that this solution shows that the relaxed definition trivialises the model of learning.

2.5 A k-decision list over n boolean variables $z_1, \ldots, z_n$, is defined by an ordered list

$$L = (t_1, b_1), (t_2, b_2), \ldots, (t_l, b_l),$$

and a bit b, where each t_i is a term (conjunction) of at most k literals (positive or negative) and each $b_i \in \{0, 1\}$. For $\mathbf{x} \in \{0, 1\}^n$ the value $L(\mathbf{x})$ is defined to be b_j, where j is the smallest index satisfying $t_j(\mathbf{x}) = 1$ and $L(\mathbf{x}) = b$ if no such index exists. Pictorially, a decision list can be depicted as shown below. As we move from left to right, the first time a term is satisfied, the corresponding b_j is output, if none of the terms is satisfied the default bit b is output.



Give an *efficient* consistent learner for the class of decision lists. As a first step, argue that it is enough to just consider the case where all the terms have length 1, i.e. in fact they are just literals.

2.5 Chapter Notes

The material covered in this chapter mostly follows the paper by Blumer et al. [13]; it is worth reading this short paper directly. The material also draws from Kearns and Vazirani [34, Chapter 2].

For a wider applications of this principle in computational learning theory and beyond, the reader may refer to [28, 30, 36].

Chapter 3

The Vapnik Chervonenkis Dimension

We have studied how a consistent learner can be used to design a PAC-learning algorithm, provided the output hypothesis comes from a class that is not too large, in particular as long as the logarithm of the size of the hypothesis class can be bounded by a polynomial in the required factors. However, when the concept class or hypothesis class is infinite, this result cannot be applied at all. Concept classes that are uncountably infinite are often used in machine learning, linear threshold functions, also referred to as linear halfspaces, being the most common one. We have already studied the class of axis-aligned rectangles, and proved the correctness of a PAC-learning algorithm for this class using first principles. In this chapter, we'll study a specific *capacity measure* called the Vapnik Chervonenkis (VC) dimension of a concept class, and show that provided this can be bounded, a consistent learner can be used to design PAC-learning algorithms. In particular, the VC dimension can be finite even for concept classes that are uncountably infinite.

3.1 The Vapnik Chervonenkis (VC) Dimension

In order to keep the notational overhead to a minimum, we will elide the use of the subscript n indicating the instance size. However, it should be clear that the discussion applies to a concept class defined as $\bigcup_{n\geq 1} C_n$, where C_n is a class of concepts over X_n. Let $S \subset X$ be a finite set of instances. For a concept $c : X \to \{0,1\}$, we can consider the restriction of c to S, $c|_S : S \to \{0,1\}$, where $c|_S(\mathbf{x}) = c(\mathbf{x})$ for $\mathbf{x} \in S$. We define the following:

$$\Pi_C(S) = \{c|_S \mid c \in C\}. \tag{3.1}$$

The set $\Pi_C(S)$ is the class of distinct restrictions of concepts in C defined by the set S. Alternatively, if $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, we can associate each element of $\Pi_C(S)$ with the function values at each of the m points,

$$\Pi_C(S) = \{(c(\mathbf{x}_1), \ldots, c(\mathbf{x}_m)) \mid c \in C\} \tag{3.2}$$

Thus, the set $\Pi_C(S)$ can also be viewed as the set all possible dichotomies on S induced by C. Clearly for a set S of size m, $|\Pi_C(S)| \leq 2^m$, as C consists

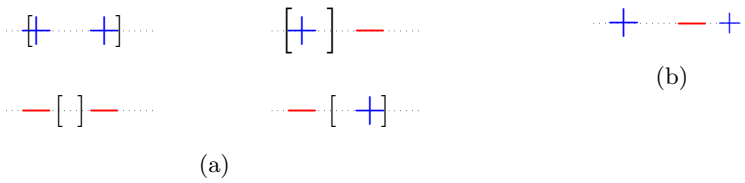


Figure 3.1: (a) All possible dichotomies on 2 points can be realised using intervals. (b) A dichotomy on three points that cannot be realised by intervals.

of boolean functions. If for a set S of size m, $|\Pi_C(S)| = 2^m$, we say that S is *shattered* by C.

Definition 3.1 **– Shattering.** *We say that a finite set $S \subset X$ is shattered by C, if $|\Pi_C(S)| = 2^{|S|}$. In other words, S is shattered by C if all possible dichotomies over S can be realised by C.*

We can now define a notion of *dimension* for a concept class C, called the Vapnik-Chervonenkis dimension, named after the authors of the seminal paper that introduced this notion to statistical learning theory.

Definition 3.2 **– Vapnik Chervonenkis (VC) Dimension.** *The Vapnik-Chervonenkis dimension of C denoted as $\mathsf{VCD}(C)$ is the cardinality d of the largest finite set S shattered by C. If C shatters arbitrarily large finite sets, then $\mathsf{VCD}(C) = \infty$.*

3.1.1 Examples

The language used to define VC-dimension is a bit different from that commonly used in machine learning. Let us use some examples to clarify this idea. The notion of shattering can be phrased as follows, given a finite set of points $S \subset X$, if we assign labels 0 or 1 (or $+$ or $-$) to the points in S arbitrarily, is there a concept $c \in C$ that is consistent with the labels? If the answer is always yes, then the set S is shattered by C, otherwise it is not.

Intervals in $\mathbb{R}$

Let $X = \mathbb{R}$ and let $C = \{c_{a,b} \mid a, b \in \mathbb{R}, a < b\}$ be the concept class of intervals, where $c_{a,b} : \mathbb{R} \to \{0, 1\}$ is defined as $c_{a,b}(x) = 1$ if $x \in [a, b]$ and 0 otherwise. What is $\mathsf{VCD}(C)$? It is easy to see that any subset $S \subset R$ of size 2 can be shattered by C, but not a set of size 3 as shown in Figure 3.1. Given a set of size three, if the middle point is labelled negative and the other two positive, there is no interval consistent with the labelling. Thus, $\mathsf{VCD}(C) = 2$.

Rectangles in $\mathbb{R}^2$

Let $X = \mathbb{R}^2$ and let C be the concept class of axis-aligned rectangles. Figure 3.2(a) shows a set of size 4 that can be shattered, Fig. 3.2(b) shows a set of size 4 that cannot be shattered, by providing an explicit labelling that cannot be achieved. However, the definition of VC dimension only requires the existence of one set of a certain size that is shattered. It is possible to show that no set of size 5

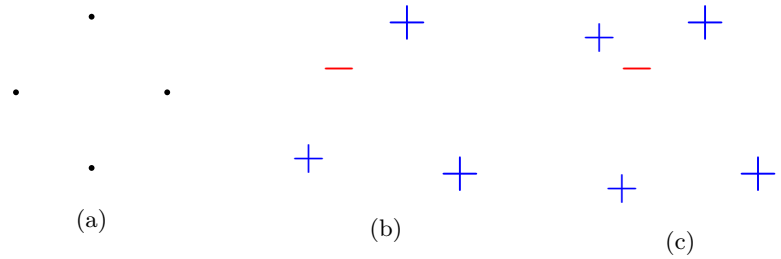


Figure 3.2: (a) A set of 4 points on which all dichotomies can be realised using rectangles. (b) A set of 4 points with a dichotomy that cannot be realised by rectangles. (c) Any set of 5 points always has a dichotomy that cannot be realised using rectangles.

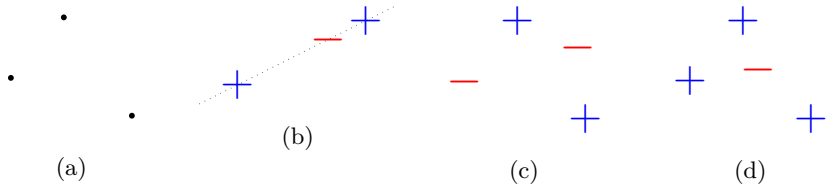


Figure 3.3: (a) A set of 3 points shattered by linear threshold functions. (b) A dichotomy on a set of 3 points that cannot be realised by linear threshold functions. (c) & (d) No set of 4 points can be shattered by linear threshold functions.

can be shattered. The reason being that there must be one of the five points that is not the extreme left, right, bottom or top point (at least not uniquely so). If this point is labelled as negative and all the other (extreme) points are labelled as positive, then there is no rectangle that can achieve this dichotomy.

Linear Threshold Functions or Linear Halfspaces

The concept class of linear threshold functions is widely used in machine learning applications. Let us show that the class of linear threshold functions in $\mathbb{R}^2$ has VC-dimension 3. Fig. 3.3(a) shows a set of size 3 that can be shattered by linear threshold functions; Fig. 3.3(b) shows a set of size 3 that cannot be shattered by linear threshold functions. No set of size 4 can be shattered by linear threshold functions. There are two possibilities, either the convex hull has four vertices in which case if the opposite ends of the quadrilateral are given the same labels, but adjacent vertices are given opposite ones, then no linear threshold function can achieve this labelling (Fig. 3.3 (c)). If on the other hand the convex hull only contains three vertices, if the vertices of the convex hull are labelled positive and the point in the interior is labelled negative, this labelling is not consistent with any linear threshold function (see Fig. 3.3 (d)). The degenerate case when three or more points lie on a line can be treated easily (e.g. as in the case of Fig. 3.3(b)). Exercise 3.1 asks the reader to show that the VC dimension of linear halfspaces in $\mathbb{R}^n$ is $n + 1$.

3.2 Growth Function

Let C be a concept class over an instance space X. The *growth function* captures the maximum number of dichotomies of a set of size m that can be realised by C. Clearly if C can shatter some set of size m, then all 2^m dichotomies can be realised—this is the case for any $m \leq \mathsf{VCD}(C)$. We are interested in understanding the maximum possible growth of the number of dichotomies for $m \geq \mathsf{VCD}(C)$. We will show that this growth can be bounded by a polynomial in m of degree $\mathsf{VCD}(d)$, rather than exponential in m.

Formally, define the *growth function*, as follows:

Definition 3.3 **– Growth Function.** *For any natural number m, define,*

$$\Pi_C(m) = \max\{|\Pi_C(S)| \mid S \subset X, |S| = m\}.$$

The goal of this section is to prove Lemma 3.4, known as the Sauer-Shelah Lemma.

Lemma 3.4 **– Sauer-Shelah Lemma.** *Let C be a concept class over X with $\mathsf{VCD}(C) = d$, then for $m \geq d$, $\Pi_C(m) \leq \left(\frac{me}{d}\right)^d$.*

In order to prove the Sauer-Shelah Lemma, it will be helpful to define a function $\Phi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defined below.

Definition 3.5. *For any $m, d \in \mathbb{N}$, define the function,*

$$\Phi(m, d) = \sum_{i=0}^{d} \binom{m}{i}. \tag{3.3}$$

From the definition of Φ, it is immediate that $\Phi(m, 0) = \Phi(0, d) = 1$ for all $m, d \in \mathbb{N}$. Furthermore, Φ is monotonically increasing in both m and d. We will make use of two additional properties of Φ which are established in the lemma below.

Lemma 3.6. *Consider the function Φ defined in Definition 3.5. The following hold:*

$$\Phi(m, d) = \Phi(m-1, d) + \Phi(m-1, d-1), \tag{3.4}$$

$$\Phi(m, d) \leq \left(\frac{me}{d}\right)^d. \qquad \text{for } m \geq d \tag{3.5}$$

Proof. The proofs are quite elementary. We will prove (3.4) first.

$$\Phi(m, d) = \sum_{i=0}^{d} \binom{m}{i} = \binom{m}{0} + \sum_{i=1}^{d} \binom{m}{i}.$$

Using the fact that $\binom{m}{0} = \binom{m-1}{0}$ and the combinatorial identity $\binom{m}{r} = \binom{m-1}{r} + \binom{m-1}{r-1}$,

$$= \binom{m-1}{0} + \sum_{i=1}^{d} \left(\binom{m-1}{i} + \binom{m-1}{i-1} \right)$$

$$= \sum_{i=0}^{d} \binom{m-1}{i} + \sum_{i=0}^{d-1} \binom{m-1}{i} = \Phi(m-1, d) + \Phi(m-1, d-1).$$

Next, we prove (3.5). Using the fact that $d/m \leq 1$, we have

$$\begin{aligned}\Phi(m,d) &= \sum_{i=0}^{d} \binom{m}{i} = \left(\frac{m}{d}\right)^d \sum_{i=0}^{d} \binom{m}{i} \cdot \left(\frac{d}{m}\right)^d \\ &\leq \left(\frac{m}{d}\right)^d \sum_{i=0}^{d} \binom{m}{i} \cdot \left(\frac{d}{m}\right)^i \leq \left(\frac{m}{d}\right)^d \sum_{i=0}^{m} \binom{m}{i} \cdot \left(\frac{d}{m}\right)^i \\ &= \left(\frac{m}{d}\right)^d \left(1+\frac{d}{m}\right)^m \leq \left(\frac{me}{d}\right)^d,\end{aligned}$$

where above we used the fact that for $d/m \leq 1$ and $i \leq d$, $(d/m)^d \leq (d/m)^i$, and that $1 + (d/m) \leq e^{d/m}$. (As an aside, observe that for $m \leq d$, $\Phi(m,d) = 2^m$.) □

Finally Lemma 3.7 together with Lemma 3.6 completes the proof of the Sauer-Shelah Lemma (Lemma 3.4).

Lemma 3.7. *For any concept class C with $\mathsf{VCD}(C) = d$, $\Pi_C(m) \leq \Phi(m,d)$.*

Proof. We will prove this by induction on m and d simultaneously. We first check the base cases. If $d = 0$, then no non-empty finite set can be shattered, so C contains at most one concept. Thus, for all m, $\Pi_C(m) = 1 = \Phi(m,0)$. If $m = 0$, since there is only one dichotomy of the empty set, clearly $\Pi_C(0) \leq \Phi(0,m)$. Now, suppose that the result holds for all $d' \leq d$ and $m' \leq m$, when at least one of the inequalities is strict. We also observe that the function Φ is monotonically increasing in both m and d.

Let S be any set of size m. Let x be a distinguished point of S. Then, by using the induction hypothesis,

$$|\Pi_C(S \setminus \{x\})| \leq \Pi_C(m-1) \leq \Phi(m-1,d). \tag{3.6}$$

Let us look at the difference between $\Pi_C(S)$ and $\Pi_C(S \setminus \{x\})$. Consider the set

$$C' = \{c \in \Pi_C(S) \mid c(x) = 0, \exists \widetilde{c} \in \Pi_C(S), \widetilde{c}(x) = 1, \forall z \in S \setminus \{x\}, c(z) = \widetilde{c}(z)\}.$$

In words, we look at a dichotomy in $\Pi_C(S \setminus \{x\})$ and see whether this can be extended in two distinct ways in $\Pi_C(S)$, i.e. whether we can keep the assignments on points in $S \setminus \{x\}$ as they were and still retain the choice to label x as either 1 or 0. Then, we have

$$|\Pi_C(S)| = |\Pi_C(S \setminus \{x\})| + |\Pi_{C'}(S \setminus \{x\})|. \tag{3.7}$$

The first term accounts for all the dichotomies on $S \setminus \{x\}$, and the second one accounts for the dichotomies on $S \setminus \{x\}$ that can be extended to two distinct dichotomies on S.

It suffice to show that $\mathsf{VCD}(C') \leq d-1$, to complete the proof by induction and (3.4). Note that the concept class C' is only defined over the set S. Let $S' \subseteq S \setminus \{x\}$ be shattered by C'. (Note that x cannot be included in any set shattered by C' since $c'(x) = 0$ for all $c' \in C'$.) Then, by definition $S' \cup \{x\}$ is shattered by C, so it must be the case that $|S'| \leq d-1$. This completes the proof. □

3.3 Sample Complexity Upper Bound

In this section, we'll prove that the VC dimension plays a role analogous to that played by $\log |H_n|$ in the case of finite hypothesis classes. Provided the learning algorithm outputs a consistent hypothesis from some hypothesis class H which has bounded VC dimension, say d, and the sample size is sufficiently large as a function of the d, $1/\epsilon$ and $1/\delta$ (though while still being polynomially bounded), this yields a PAC-learning algorithm.

Theorem 3.8. *Let C be a concept class. Let $H \supseteq C$ be a hypothesis class with $\mathsf{VCD}(H) = d$, where $1 \leq d < \infty$. Let L be a consistent learner for C that outputs a hypothesis $h \in H$. Then for every $0 < \epsilon, \delta \leq 1/2$, L is a PAC-learning algorithm for C provided it is given as input a random sample of size m drawn from $\mathsf{EX}(c, D)$, for*

$$m \geq \kappa_0 \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{d}{\epsilon} \log \frac{1}{\epsilon} \right),$$

for some universal constant κ_0.

Proof. For two boolean functions f and g, denote by $f \oplus g$ the boolean defined as follows:

$$(f \oplus g)(x) = \begin{cases} 1 & \text{if } f(x) \neq g(x) \\ 0 & \text{if } f(x) = g(x) \end{cases}.$$

Now suppose that $c \in C$ is the target concept and D is the target distribution over the instance space X. For any hypothesis $h \in H$, $\mathrm{err}(h; c, D) = \mathbb{P}_{x \sim D}\left[(c \oplus h)(x) = 1\right]$.

Let $H \oplus c = \{h \oplus c \mid h \in H\}$. It is easy to show that $\mathsf{VCD}(H \oplus c) = \mathsf{VCD}(H)$ (see Exercise 3.4). We say that a finite set $S \subset X$ is an ϵ-net for $H \oplus c$ with respect to distribution D, if for every $h \oplus c \in H \oplus c$, such that $\mathbb{P}_{x \sim D}\left[(h \oplus c)(x) = 1\right] \geq \epsilon$, there exists some $x \in S$, such that $(h \oplus c)(x) = 1$.

We observe that a hypothesis, h, for which $\mathbb{P}_{x \sim D}\left[(h \oplus c)(x) = 1\right] \geq \epsilon$ is problematic, as $\mathrm{err}(h; c, D) \geq \epsilon$. We want to ensure that the consistent learner does not output any such hypothesis. Any S that is an ϵ-net for $H \oplus c$ with respect to D, rules out such hypotheses being output by a consistent learner, as they would not be consistent! Thus, it suffices to show that a random sample of size m drawn from $\mathsf{EX}(c, D)$ actually yields an ϵ-net for $H \oplus c$ with respect to D.

The rest of the proof is essentially a clever argument about the probabilities of certain events set up by doubling the sample size and symmetrising. This idea appears in most proofs related to sample complexity bounds, and is our first introduction to this proof technique.

We will draw a sample S of size $2m$ in two phases. First draw a sample S_1 of size m from $\mathsf{EX}(c, D)$. Let A be the event that S_1 (actually, the input part of S_1 obtained by ignoring the labels) is not an ϵ-net for $H \oplus c$ with respect to D.[1] Now, suppose the event A occurs, then there exists $\widetilde{h} \in H$ such that $(\widetilde{h} \oplus c)(x) = 0$ for all $x \in S_1$ and $\mathbb{P}_{x \sim D}\left[(\widetilde{h} \oplus c)(x) = 1\right] \geq \epsilon$. Fix such a $\widetilde{h} \in H$ and draw a second sample S_2 of size m. Now, let us obtain a *lower*

[1] Actually S_1 can be multiset, e.g. if D has point masses.

bound on the number of elements x in S_2 that satisfy $(\widetilde{h} \oplus c)(x) = 1$. Let X_i denote the random variable that takes value 1 if the i^{th} element of S_2 satisfies $(\widetilde{h} \oplus c)(x) = 1$ and X_i takes value 0 otherwise. Thus, if $X = \sum_{i=1}^{m} X_i$, then X is the (random) number of such points in S_2. Note that, $\mathbb{E}[X] \geq \epsilon m$, so by using a Chernoff bound from Eq. (A.3), we have

$$\mathbb{P}\left[X < \epsilon m/2\right] \leq \mathbb{P}\left[X \leq \mathbb{E}[X]\left(1 - \frac{1}{2}\right)\right] \leq \exp\left(-\frac{\epsilon m}{16}\right)$$

Provided $\epsilon m \geq 16$ (which our final bound will ensure), the probability that $|\{x \in S_2 \mid (\widetilde{h} \oplus c)(x) = 1\}| \geq \epsilon m/2$ is at least $1/2$.

Now consider the event B defined as follows: A sample $S = S_1 \cup S_2$ of size $2m$ with $|S_1| = |S_2| = m$ is drawn from $\mathsf{EX}(c, D)$, there exists a $\widetilde{h} \oplus c \in \Pi_{H \oplus c}(S)$, such that $|\{x \in S \mid (\widetilde{h} \oplus c)(x) = 1\}| \geq \epsilon m/2$ and $(\widetilde{h} \oplus c)(x) = 0$ for all $x \in S_1$. We have slightly abused notation and used $\widetilde{h} \oplus c$ to denote the function in $H \oplus c$ and its restriction to the set S in $\Pi_{H \oplus c}(S)$. Note that $\mathbb{P}[B] \geq \frac{1}{2}\mathbb{P}[A]$, since if S_1 fails to be an ϵ-net for $H \oplus c$ with respect to D, then as argued above, the probability of there being a $(\widetilde{h} \oplus c) \in \Pi_{H \oplus c}(S)$ such that $|\{x \in S_1 \mid \widetilde{h} \oplus c(x) = 1\}| = 0$ and $|\{x \in S_2 \mid \widetilde{c}(x) = 1\}| \geq \epsilon m/2$ is at least $1/2$. Thus, $\mathbb{P}[A] \leq 2\mathbb{P}[B]$.

We will now bound $\mathbb{P}[B]$ which is a purely combinatorial problem. Let

$$\Pi^{\epsilon}_{H \oplus c}(S) = \left\{h \oplus c \in \Pi_{H \oplus c}(S) \mid |\{x \in S \mid (h \oplus c)(x) = 1\}| \geq \epsilon m/2\right\}.$$

In defining the event B, we can first imagine the entire sample S of size $2m$ being drawn from D, denoted by $S \sim D^{2m}$, and then for the fixed sample S a uniformly random partition into S_1 and S_2 being made. Note that the distribution over S_1 obtained by first drawing S and then randomly partitioning is exactly the same as that obtained by drawing m examples directly from $\mathsf{EX}(c, D)$. For any fixed $h \oplus c \in \Pi^{\epsilon}_{H \oplus c}(S)$, let $B_{h \oplus c}|S$ denote the event (conditioned on S) that $|\{x \in S_1 |\ (h \oplus c)(x) = 1\}| = 0$. Then calculating the probability of $B_{h \oplus c}|S$ is equivalent to the following question: Given $2m$ balls out of which $r \geq \epsilon m/2$ are red and the remaining are black, if we divided them into two sets of size m each, without seeing the colours, what is the probability that the first set has no red balls and the second set has all of them? This probability is simply given by $\binom{m}{r}/\binom{2m}{r}$. We can bound this as follows:

$$\frac{\binom{m}{r}}{\binom{2m}{r}} = \prod_{i=0}^{r-1} \frac{m-i}{2m-i} \leq \frac{1}{2^r}.$$

Note that the above bound is still valid for $r > m$ as the probability of $B_{h \oplus c}|S$

is 0 in that case. We can then bound the probability of the event B as follows:

$$\begin{aligned}\mathbb{P}_{S\sim D^{2m}}[B] &= \mathbb{P}_{S\sim D^{2m}}\left[\mathbb{P}_{S_1,S_2}\left[\bigcup_{h\oplus c\in\Pi^{\epsilon}_{H\oplus c}} B_{h\oplus c} \mid S\right]\right] \\ &\leq \mathbb{P}_{S\sim D^{2m}}\left[\sum_{h\oplus c\in\Pi^{\epsilon}_{H\oplus c}} \mathbb{P}_{S_1,S_2}\left[B_{h\oplus c} \mid S\right]\right] \\ &\leq |\Pi^{\epsilon}_{H\oplus c}| \cdot 2^{-\epsilon m/2} \leq \left(\frac{2em}{d}\right)^{d} 2^{-\epsilon m/2}.\end{aligned}$$

Since we have $\mathbb{P}[A] \leq 2\mathbb{P}[B]$, we have that,

$$\mathbb{P}[A] \leq 2 \cdot \left(\frac{2em}{d}\right)^{d} 2^{-\epsilon m/2}.$$

It remains to be shown that $\mathbb{P}[A] \leq \delta$ for the value of m in statement of the theorem. Although it is a standard calculation, it is worth spelling out in full at least once. We first observe that it suffices to show that,

$$m \geq \frac{2d}{\epsilon \cdot \log 2} \log \frac{2em}{d} + \frac{2}{\epsilon \cdot \log 2} \log \frac{2}{\delta}.$$

Thence it suffices for $\frac{m}{2} \geq \frac{2d}{\epsilon\cdot\log 2} \log \frac{2em}{d}$ and $\frac{m}{2} \geq \frac{2}{\epsilon\cdot\log 2} \log \frac{2}{\delta}$ to both hold. The first is equivalent to showing $\frac{m}{d} \geq \frac{4}{\epsilon\cdot\log 2} \log \frac{2em}{d}$, which holds for $m \geq \frac{32d}{\epsilon\cdot\log 2} \log \frac{4}{\epsilon\cdot\log 2}$ by appealing to Lemma B.1 (noting that $4/(\epsilon \log 2) \geq e$ for all $\epsilon \leq 1$ and that $2+2\log(2e) \leq 8$). The second clearly holds for $m \geq \frac{4}{\epsilon\cdot\log 2} \log \frac{2}{\delta}$. Hence, picking

$$m \geq \frac{4}{\epsilon \cdot \log 2} \max\left\{8d \log\left(\frac{4}{\epsilon \cdot \log 2}\right), \log \frac{2}{\delta}\right\},$$

is sufficient as stated in the statement of the theorem. □

Theorem 3.8 could of course be applied with $H = C$ in the context of *proper* PAC learning. However, the more general result will allow us to consider scenarios where efficient consistent learners could be designed by allowing the learning algorithm to output a hypothesis from a class that is larger than C. As we shall see next, the VC dimension essentially completely captures the *statistical complexity* of learning.

3.4 Sample Complexity Lower Bounds

In this section, we will show sample complexity lower bounds for any learning algorithm in terms of the VC dimension. This is a purely information-theoretic result; no assumption is made about the running time of the algorithm. There is also no requirement that the algorithm output a hypothesis from the concept class C.

Theorem 3.9. *Let C be a concept class with $\mathsf{VCD}(C) \geq d$, where $d \geq 25$.*[2] *Then any PAC-learning algorithm (not necessarily efficient) for learning C using $H \supseteq C$ requires at least $\max\{\frac{d-1}{32\epsilon}, \frac{1}{4\epsilon}\log\frac{1}{4\delta}\}$ examples.*

Proof. In order to show that such an algorithm doesn't exist, we need to show that for every learning algorithm L, there exists a target distribution D and a target concept c, such that with probability strictly greater than δ, the output hypothesis has error strictly greater than ϵ.

Suppose for contradiction such a learning algorithm does exist. Note that the learning algorithm L may itself be randomised, however, we know that there is an upper bound $m = \max\{\frac{d-1}{32}, \frac{1}{4\epsilon}\log\frac{1}{\delta}\}$ on the number of examples it uses. Thus, if the learning algorithm output a hypothesis from $H \supseteq C$, we can view any fixed sample S of size m as defining a distribution over H. We will use the *probabilistic method* to derive a contradiction (see e.g. [2]).

We will first show that m must be at least $\frac{d-1}{32\epsilon}$. Let T be a set of size d that is shattered by C. Suppose $T = \{x_1, x_2, \ldots, x_d\}$, and let D be a distribution defined as follows: $D(x_1) = 1 - 8\epsilon$, and $D(x_j) = 8\epsilon/(d-1)$ for $j = 2, \ldots, d$. Since the distribution is only supported on the finite set T, we only need to be concerned with concepts in $\Pi_C(T)$. Suppose the learning algorithm receives a sample S of size $m = (d-1)/(32\epsilon)$ examples drawn according to D and labelled according to some target $c \in \Pi_C(T)$. We may assume without loss of generality that if $x_1 \in S$, then L outputs some h such that $h(x_1) = c(x_1)$. (If not, we can design an algorithm L' that runs L to obtain h and chooses $h' \in H$ which satisfies $h'(x_1) = c(x_1)$ and $h'(x_i) = h(x_i)$ for $i \geq 2$.)

We first note that the probability that $(x_1, c(x_1)) \notin S$ is small, in particular at most $(1-8\epsilon)^{(d-1)/(32\epsilon)} \leq e^{-(d-1)/4} \leq e^{-6}$. We also show that with a reasonable probability S contains fewer than half the examples from the set $T \setminus \{x_1\} = \{x_2, \ldots, x_d\}$. Let Z_i be the random variable that is 1 if the i^{th} example drawn from D is in the set $T \setminus \{x_1\}$ and 0 otherwise. Then $Z_i = 1$ with probability 8ϵ and 0 with probability $1 - 8\epsilon$. Let $Z = \sum_{i=1}^{m} Z_i$ be the number of examples seen from the set $T \setminus \{x_1\}$ (possibly with repetitions). Then $\mathbb{E}[Z] = \frac{d-1}{4}$ and using a Chernoff bound from Eq. (A.4),

$$\mathbb{P}\left[Z \geq \frac{d-1}{2}\right] \leq \mathbb{P}\left[Z \geq 2 \cdot \mathbb{E}[Z]\right] \leq \exp\left(-\frac{d-1}{12}\right) \leq e^{-2}.$$

Let us denote the event that $(x_1, c(x_1)) \in S$ and $Z < \frac{d-1}{2}$ as $\mathcal{E}$. It can be easily checked that $\mathbb{P}[\mathcal{E}] > \frac{1}{2}$. Now suppose the target concept c was chosen uniformly at random from $\Pi_C(T)$. As T is shattered, $|\Pi_C(T)| = 2^d$. We can compute the conditional expectation (on the event $\mathcal{E}$) of the error of h output by the learning algorithm. Conditioned on $\mathcal{E}$, we know that $|S| \leq Z + 1 < \frac{d+1}{2}$. Thus the conditional (on $\mathcal{E}$) distribution over the target distributions is uniform over a set of size $2^{d-|S|} > 2^{(d-1)/2}$. (In words, the assignment to the examples observed by the learning algorithm is fixed, but any assignment on the unseen examples is equally likely.) Then observe that for any fixed hypothesis, h, a random c conditioned on $\mathcal{E}$ results in

$$\mathrm{err}(h; c, D) = \frac{8\epsilon}{2(d-1)}(|T| - |S|) > 2\epsilon.$$

[2] The condition $d \geq 25$ is not really necessary, with a slightly improved argument this can be shown for any $d \geq 2$.

Putting everything together we have that,

$$\mathop{\mathbb{E}}_{c\sim\Pi_C(T)}\left[\mathop{\mathbb{E}}_{\substack{S\sim\mathsf{EX}(c,D)^m\\ h\sim L}}\left[\mathrm{err}(h;c,D)|\mathcal{E}\right]\right] > 2\epsilon.$$

Thus, conditioned on the event $\mathcal{E}$, there must exist a target concept $c \in C$ for which $\mathop{\mathbb{E}}_{h\sim L}\left[\mathrm{err}(h;c,D)|\mathcal{E}\right] > 2\epsilon$. Now because $h(x_1) = c(x_1)$ conditioned on the event $\mathcal{E}$, we also have that $\mathrm{err}(h;c,D) \leq 8\epsilon$ whenever $\mathcal{E}$ occurs. As a result, it is easy to see that conditioned on event $\mathcal{E}$, the probability that $\mathrm{err}(h;c,D) \leq \epsilon$ is at most $6/7$. Otherwise, we would have,

$$\mathop{\mathbb{E}}_{h\sim L}\left[\mathrm{err}(h;c,D)|\mathcal{E}\right] \leq \frac{8\epsilon}{7} + \frac{6}{7}\cdot\epsilon \leq 2\epsilon.$$

This completes the proof of $\frac{d-1}{32\epsilon}$ as a lower bound.

In order to show that $m = \frac{1}{4\epsilon}\log\frac{1}{4\delta}$ as a lower bound, we can use a very similar (but simpler) argument. In fact, we only need a set $T = \{x_1, x_2\}$, and two concepts $c_1, c_2 \in C$, such that $c_1(x_1) = c_2(x_1) = 0$ and $c_1(x_2) \neq c_2(x_2)$; such a set T and concepts c_1, c_2 exist as $\mathsf{VCD}(C) \geq 2$. Now define a distribution D over T such that $D(x_1) = \exp(-4\epsilon)$ and $D(x_2) = 1 - \exp(-4\epsilon)$. It is easy to see that with probability at least 4δ, a sample of size $m = \frac{1}{4\epsilon}\log\frac{1}{4\delta}$ from D will not have the point x_2. Conditioned on that event, again denoted by $\mathcal{E}$, we can conclude that the expected error, $\mathop{\mathbb{E}}_{h\sim L}\left[\mathrm{err}(h;c,D)|\mathcal{E}\right] \geq (1 - e^{-4\epsilon})/2$. It is a straightforward calculation, e.g. using Taylor's theorem, that for $\epsilon < 1/4$, $(1 - e^{-4\epsilon}) > 2\epsilon$. Then, the same argument as above shows that conditioned on $\mathcal{E}$, with probability $> 1/4$ the error of h output by L will be $> \epsilon$. That completes the proof. □

3.5 Consistent Learner for Linear Threshold Functions

To end this chapter, we will look at an application to learning linear threshold functions. Recall that the class of linear threshold functions over $\mathbb{R}^n$ is defined as

$$\mathsf{LTF}_n = \{\mathbf{x} \mapsto \mathbb{1}_{\geq 0}(\mathbf{w}\cdot\mathbf{x} + w_0) \mid \mathbf{w} \in \mathbb{R}^n, \|\mathbf{w}\|_2 = 1, w_0 \in \mathbb{R}\}, \tag{3.8}$$

where $\mathbb{1}_{\geq 0}(z) = 1$ if $z \geq 0$ and 0 otherwise.

Exercise 3.1 asks you to show that the VC dimensions of this class is $n + 1$. Thus in order to apply Theorem 3.8, we would like to design an *efficient* consistent learner for the class LTF_n. The problem is the following: Given $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{0, 1\}$, such that there exists $\mathbf{w}^* \in \mathbb{R}^n$, $\|\mathbf{w}^*\|_2 = 1$ and $w_0^* \in \mathbb{R}$ such that $y_i = \mathbb{1}_{\geq 0}(\mathbf{w}^*\cdot\mathbf{x}_i + w_0^*)$, find some $\mathbf{w}$, w_0, such that $y_i = \mathbb{1}_{\geq 0}(\mathbf{w}\cdot\mathbf{x}_i + w_0)$. This problem can be formulated as a linear program and hence solved in polynomial time.

We consider the following linear program with variables, $w_0, w_1, \ldots, w_n$. The objective function is constant, so we are in fact only looking for a feasible point. The constraints are given by:

$$w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots w_n x_{in} \geq 0 \quad \text{For all } i \text{ such that } y_i = 1 \tag{3.9}$$

$$-w_0 - w_1 x_{i1} - w_2 x_{i2} - \cdots w_n x_{in} \geq 1 \quad \text{For all } i \text{ such that } y_i = 0 \tag{3.10}$$

Let us first discuss the second inequality (3.10). An example is classified as negative if $w^* \cdot x + w_0^* < 0$; however, we cannot include strict inequalities as part of the linear program as we need the resulting set to be closed. The choice of 1 is arbitrary, we could have used any strictly positive real number. Let us show that the above linear program has a feasible solution; given that a feasible solution exists, there are known polynomial time algorithms to find one.

Let the target linear threshold function be defined by $\mathbf{w}^*, w_0^*$. Let $\alpha = \min\{-(\mathbf{w}^* \cdot \mathbf{x}_i + w_0^*) \mid y_i = 0\}$; we know that $\alpha > 0$. Consider $\frac{\mathbf{w}^*}{\alpha} \in \mathbb{R}^n$, $\frac{w_0}{\alpha} \in \mathbb{R}$; it can be checked that this is a feasible solution to the constraints defined by (3.9) and (3.10).

3.6 Exercises

3.1 Show that the concept class of linear halfspaces over $\mathbb{R}^n$ defined in Section 3.5 has VC-dimension $n+1$ by proving the following.

i) Give a set of $n+1$ points in $\mathbb{R}^n$ that is shattered by the class of linear halfspaces.

ii) Show that no set of $m = n+2$ points in $\mathbb{R}^n$ can be shattered by the class of linear halfspaces. For this you can use Radon's theorem, the statement of which appears below.

iii) Prove Radon's theorem.

Radon's Theorem

Given a set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathbb{R}^n$, the convex hull of S is the set

$$\{\mathbf{z} \in \mathbb{R}^n \mid \exists \lambda_1, \ldots, \lambda_m \in [0,1],\ \sum_{i=1}^{m} \lambda_i = 1,\ \mathbf{z} = \sum_{i=1}^{m} \lambda_i \mathbf{x}_i\}.$$

Let $m \geq n+2$, then S must have two disjoint subsets S_1 and S_2 whose convex hulls intersect.

3.2 Prove that for any $d \in \mathbb{N}$, there is a concept class C such that $\mathsf{VCD}(C) = d$, and that for any $m \in \mathbb{N}$, $\Pi_C(m) = \Phi_d(m)$.

3.3 In this question we will consider the learnability of convex sets. Let us consider the domain to be $X = [0,1]^2$, the unit square in the plane. For $S \subset X$ a convex set, let $c_S : X \to \{0,1\}$, where $c_S(x) = 1$ if $x \in S$ and 0 otherwise. Let $C = \{c_S \mid S \text{ convex subset of } X\}$ be the concept class defined by convex sets of $[0,1]^2$.

i) Show that the VC dimension of C is ∞. This shows that the concept class of convex sets of $[0,1]^2$ cannot be learnt by an algorithm (efficiently or otherwise) that uses a sample whose size is bounded by a polynomial in $1/\epsilon$ and $1/\delta$ alone.

ii) We will consider a restriction of PAC-learning where the learning algorithm is only required to work for a specific distribution D over X. Show that if D is the uniform distribution over $[0,1]^2$, then the concept class of convex sets is *efficiently* PAC-learnable in this restricted sense, where efficiency means running time (and sample

complexity) bounded by a polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

Hint: Consider the algorithm that simply outputs the convex hull of positive points as the output hypothesis. You may use the fact that the perimeter of any convex set in the unit square can be at most 4*.*

3.4 Show that $\mathsf{VCD}(H \oplus c) = \mathsf{VCD}(H)$.

Chapter 4

Boosting

4.1 Weak Learnability

Let us revisit the definition of PAC-learning. Definition 1.8 places quite stringent requirements on a learning algorithm that (efficient) PAC learns a concept class. The learning algorithm has to work for all target concepts in the class, for all input distributions, and for any setting of accuracy (ϵ) and confidence (δ) parameters. It is worthwhile considering what happens when we relax some of these requirements. In Exercise 1.3, we have seen that fixing the confidence parameter to be a constant, e.g. $\delta = 1/4$, leaves the notion of PAC-learnability unchanged. On the other hand, if we only require the learning algorithm to succeed with respect to certain input distributions, then PAC-learning is possible for concept classes that are not learnable (efficiently or otherwise) using a sample size that is polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ in the distribution-free sense, i.e. algorithms that have to work with respect to all distributions.[1] Exercise 3.3 explores such a concept class. In this chapter, we focus on the accuracy parameter, ϵ. The problem of learning is trivial if $\epsilon \geq 1/2$ as we can make a random prediction on an input $x \in X$ and achieve an error of $1/2$.[2] The question we are interested in is what happens when $\epsilon = 1/2 - \gamma$, for some $\gamma > 0$? For example, one may wonder if it is possible to learn some concept class up to error $1/4$, but not to an arbitrarily small ϵ?

Surprisingly, the answer to the question above is *no*, i.e. if we can learn a concept class up to error at most $1/2 - \gamma$, then we can learn this class up to error bounded by any $\epsilon > 0$. This method is known as *boosting*, as we take a "weak learning" algorithm and boost it to produce a "strong learning" algorithm.

[1] What is most important here is the order of quantifiers. The notion of PAC-learning requires a single learning algorithm to work regardless of the input distribution. Of course, the learning algorithm may be adaptive in the sense that depending on what examples it has received it can change its behaviour.

[2] If the output hypothesis is allowed to be randomised, that is it takes as input $x \in X$, and also has access to random coin tosses when making a prediction, then it is immediately clear that the outlined approach works. Otherwise, we would need that $\epsilon > \frac{1}{2} + \gamma$; then we know that one of the two constant hypotheses, always predicting 1, or always predicting 0, gives error at most $1/2$, and we can with high confidence determine which one can be guaranteed to have error at most ϵ by using a sample of size $O\left(\frac{1}{\gamma^2}\right)$.

γ-Weak Learner

Let us define the notion of weak learning formally. We will let the parameter γ for weak learning be a function of the instance size n, and the representation size of the target concept, size(c).

Definition 4.1 **– γ-Weak Learning.** *For $\gamma(\cdot,\cdot)$ with $\gamma > 0$, we say that L is a γ-weak PAC learning algorithm for concept class C using hypothesis class H, if for any $n \geq 0$, any $c \in C_n$, any D over X_n, and $0 < \delta < 1/2$, L given access to $\mathsf{EX}(c, D)$ and inputs* size(c)*, δ and γ, outputs $h \in H_n$ that with probability at least $1 - \delta$, satisfies,* $\mathrm{err}(h) \leq \frac{1}{2} - \gamma(n, \mathrm{size}(c))$.

We say that L is an efficient γ-weak PAC learner if H is polynomially evaluatable, $1/\gamma(n, \mathrm{size}(c))$ *is bounded by some polynomial in n and* size(c)*, and the running time of L is polynomial in n, $1/\delta$, and* size(c).

Boosting: A Short History

Boosting has an interesting history and is a prominent example of how a suitable theoretical question has led to some very practical algorithms. The notion of weak learning first appeared in the work of Kearns and Valiant [32], who showed that certain concept classes were hard to learn even when the requirement was only to output a hypothesis that was slightly better than random guessing. Shortly thereafter, Freund [25] and Schapire [42] showed that in the distribution-free setting weak and strong learning are in fact equivalent. The early boosting algorithms were not easy to implement in practice; Freund and Schapire [26] designed an improved boosting algorithm, called Adaboost, which while retaining strong theoretical guarantees was very easy to implement in practice. Adaboost has enjoyed a remarkable practical success and implementations of Adaboost and its variants appear in most machine learning libraries.

4.2 The AdaBoost Algorithm

The central idea of the boosting approach is the following. Initially, we can use a weak learning algorithm that gives us a hypothesis that performs slightly better than random guessing. We could repeatedly run this weak learning algorithm, though it may return the same hypothesis. However, if we modify the distribution so that the hypothesis already returned is no longer valid, i.e. under the new distribution it has error exactly $1/2$, then the weak learning algorithm is required to provide us with a different hypothesis.[3] By doing this repeatedly, we can combine several hypotheses to produce one that has low error. All boosting algorithms make use of this high-level approach. The AdaBoost (for adaptive boosting) algorithm exploits the fact that some hypotheses may be much better than others and aggressively modifies the distribution to account for this. Initially, we will concentrate on proving that AdaBoost succeeds in finding a hypothesis that has training error 0 on a given sample.

The AdaBoost algorithm is described in Alg. 4.1. We assume that AdaBoost has access to the weak learning algorithm, WeakLearn. WeakLearn gets

[3] If the error of h is much larger than $1/2$ under the modified distribution, then the weak learning algorithm may simply return $1 - h$, which is not of much use, since we already have h.

Algorithm 4.1: AdaBoost

Inputs:
  Training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ drawn from $\mathsf{EX}(c, D)$,
  T (#iterations), δ (confidence parameter)
  Weak learning algorithm WeakLearn(D, δ)
// uniform initial distribution over training data
Set $D_1(i) = 1/m$
for $t = 1, \ldots, T$ **do**
  // examples drawn from D_t are passed to WeakLearn
  Obtain $h_t \leftarrow$ WeakLearn$(D_t, \delta/T)$
  Set $\epsilon_t = \mathbb{P}_{(\mathbf{x},y)\sim D_t}\left[h_t(\mathbf{x}) \neq y\right]$ // $\epsilon_t \leq 1/2 - \gamma$, w.p. $\geq 1 - \frac{\delta}{T}$
  Set $\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  // Z_{t+1} is the normalising constant
  Update $D_{t+1}(i) = D_t(i) \cdot \exp(-\alpha_t y_i h_t(\mathbf{x}_i))/Z_{t+1}$
Set $\widetilde{h} = \sum_{i=t}^{T} \alpha_t h_t$
Output: hypothesis $\widehat{h} : X \to \{-1, 1\}$, where $\widehat{h}(\mathbf{x}) = \text{sign}(\widetilde{h}(\mathbf{x}))$

labelled (according to some concept $c \in C$) data from some distribution D and takes a confidence parameter δ. It guarantees that with probability at least $1 - \delta$, the error of the returned hypothesis, h, is at most $1/2 - \gamma$. AdaBoost receives a training sample of m examples drawn from $\mathsf{EX}(c, D)$. It defines a distribution D_t over this sample at each iteration and hence can simulate the example oracle for the weak learning algorithm. To make the mathematical analysis simpler, we will assume that the labels y_i are in $\{-1, 1\}$ rather than $\{0, 1\}$. This is a transformation that is frequently used in machine learning and readers should convince themselves that this does not make any difference. We assume that sign : $\mathbb{R} \to \{-1, 1\}$, with $\text{sign}(z) = -1$ if $z < 0$ and $\text{sign}(z) = 1$ if $z \geq 0$.

Theorem 4.2. *Assuming that* WeakLearn *is a γ-weak learner for the concept class C, after T iterations, with probability at least $1 - \delta$, the training error of the hypothesis output by AdaBoost (Alg. 4.1) is 0, provided $T \geq \frac{\log 2m}{2\gamma^2}$.*

Proof. As further notation, let $\mathbb{1}(\cdot)$ be the indicator of the predicate inside the parentheses, which takes the value 1 if the predicate is true and 0 otherwise. Observe that $\mathbb{1}(\text{sign}(\widetilde{h}(\mathbf{x})) \neq y) \leq e^{-y\widetilde{h}(\mathbf{x})}$ for $y \in \{-1, 1\}$.

$$\mathbb{P}_{(\mathbf{x},y)\sim D_1}\left[\text{sign}(\widetilde{h}(\mathbf{x})) \neq y\right] = \sum_{i=1}^{m} D_1(i) \cdot \mathbb{1}(\text{sign}(\widetilde{h}(\mathbf{x}_i) \neq y_i) \tag{4.1}$$

$$\leq \sum_{i=1}^{m} D_1(i) \cdot e^{-y_i \widetilde{h}(\mathbf{x}_i)} \tag{4.2}$$

We introduce some additional notation. Let $\widetilde{h}_t = \sum_{s=t}^{T} \alpha_s h_s$ be the weighted sum of the hypotheses returned in iterations t through T; and thus, $\widetilde{h}_t = \alpha_t h_t + \widetilde{h}_{t+1}$. We will allow the overall algorithm to fail if any of the calls to WeakLearn on Line 9 fail. By a simple union bound, all of these calls succeed

with probability at most $1-\delta$. We will assume this is the case in the rest of the proof allowing the algorithm a failure probability δ. Then consider the following:

$$\sum_{i=1}^{m} D_1(i) \cdot e^{-y_i \widetilde{h}(\mathbf{x}_i)} = \sum_{i=1}^{m} D_1(i) \cdot e^{-y_i \widetilde{h}_1(\mathbf{x}_i)} \tag{4.3}$$

$$= \sum_{i=1}^{m} D_1(i) \cdot e^{-\alpha_1 y_i h_1(\mathbf{x}_i)} \cdot e^{-y_i \widetilde{h}_2(\mathbf{x}_i)} \tag{4.4}$$

$$= Z_2 \cdot \sum_{i=1}^{m} D_2(i) \cdot e^{-y_i \widetilde{h}_2(\mathbf{x}_i)} \tag{4.5}$$

$$= Z_2 \cdot \sum_{i=1}^{m} D_2(i) \cdot e^{-\alpha_2 y_2 h_2(\mathbf{x}_i)} \cdot e^{-y_i \widetilde{h}_3(\mathbf{x}_i)} \tag{4.6}$$

$$= Z_2 \cdot Z_3 \cdot \sum_{i=1}^{m} D_3(i) \cdot e^{-y_i \widetilde{h}_3(\mathbf{x}_i)} \tag{4.7}$$

We use in (4.3) $\widetilde{h} = \widetilde{h}_1$, in (4.4) $\widetilde{h}_1 = \alpha_1 h_1 + \widetilde{h}_2$, in (4.5) $D_2(i) = D_1(i) \cdot e^{-\alpha_1 y_i h_1(\mathbf{x}_i)}/Z_2$, in (4.6) $\widetilde{h}_2 = \alpha_2 h_2 + \widetilde{h}_3$, and in (4.7) $D_3(i) = D_2(i) \cdot e^{-\alpha_2 y_i h_2(\mathbf{x}_i)}/Z_3$. Continuing this way, we obtain,

$$\sum_{i=1}^{m} D_1(i) \cdot e^{-y_i \widetilde{h}(\mathbf{x}_i)} = Z_2 \cdot Z_3 \cdots Z_T \cdot \sum_{i=1}^{m} D_T(i) \cdot e^{-y_i \widetilde{h}_T(\mathbf{x}_i)}$$

And thus,

$$\sum_{i=1}^{m} D_1(i) \cdot e^{-y_i \widetilde{h}(\mathbf{x}_i)} = \prod_{t=2}^{T+1} Z_t \tag{4.8}$$

Let us now obtain a bound on Z_{t+1}, for $t = 1, \ldots, T$. We have,

$$\begin{aligned} Z_{t+1} &= \sum_{i: h_t(\mathbf{x}_i) = y_i} D_t(i) \cdot e^{-\alpha_t} + \sum_{i: h_t(\mathbf{x}_i) \neq y_i} D_t(i) \cdot e^{\alpha_t} \\ &= (1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} = 2\sqrt{\epsilon_t (1-\epsilon_t)} \end{aligned}$$

Above we substituted $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$. Letting $\gamma_t = \frac{1}{2} - \epsilon_t$ and using the fact that $\sqrt{1-x} \leq e^{-x/2}$, we get,

$$Z_{t+1} = \sqrt{1 - 4\gamma_t^2} \leq e^{-2\gamma_t^2} \tag{4.9}$$

Now, by the guarantee on the weak learning algorithm, $\gamma_t \geq \gamma$ for $t = 1, \ldots, T$. Thus, $\prod_{t=2}^{T+1} Z_t \leq e^{-2T\gamma^2}$. Provided $T \geq \log(2m)/(2\gamma^2)$, the training error is at most $1/(2m)$ and hence must in fact be 0 (as error on any point causes the error to be at least $1/m$). □

It is worth understanding what the algorithm is doing in each iteration. In Line 13 the algorithm assigns higher weight to examples that were misclassified

by the hypothesis h_t in the t^{th} iteration and lower weight to examples that were correctly classified. Equation (4.8) shows that the product of the normalising constants Z_t is an upper bound on the training error of the classifier after T iterations. Each Z_t is guaranteed to be strictly less than 1 because of the assumption that the weak learner outputs a hypothesis with error at most $1/2 - \gamma$. The choice of α_t in Line 10 is chosen to minimise the value of Z_t at that iteration. It is in this sense that the algorithm is *adaptive* and hence its name. It is worth observing that α_t is also the weight that the hypothesis h_t gets in the final threshold classifier; the more accurate h_t is, the greater the value of α_t. You are asked to further explore some of the behaviour of this algorithm in Exercise 4.1.

4.2.1 Bounding the Generalisation Error

One way to bound the generalisation error of AdaBoost is by ensuring that the VC-dimension of the hypothesis class used by the weak learning algorithm is finite.[4] Suppose the weak learning algorithm, WeakLearn, outputs hypotheses from H and $\mathsf{VCD}(H) = d$. Denote by $\mathsf{THRESHOLDS}_k(H)$ the class of functions given by

$$\mathsf{THRESHOLDS}_k(H) = \left\{ \mathbf{x} \mapsto \operatorname{sign}\left(\sum_{i=1}^{k} \alpha_k h_k(\mathbf{x}) \right) \mid h_i \in H, \alpha_i \in \mathbb{R} \right\}.$$

Lemma 4.3. *If* $\mathsf{VCD}(H) = d$, *then* $\mathsf{VCD}(\mathsf{THRESHOLDS}_k(H)) = O(kd \log(k))$.

Proof. We count the number of dichotomies realised by functions in $\mathsf{THRESHOLDS}_k(H)$ on a set of size m. Let S be a set of size m. We fix the choices $h_1, \ldots, h_k$, then this results in m points in $\{-1, 1\}^k$ that are then fed into a linear threshold function. The VC dimension of linear threshold functions in $\mathbb{R}^k$ is $k+1$, so the number of dichotomies realised (by LTFs that vary depending on the choice of the α_is) are at most $\left(\frac{em}{k+1}\right)^{k+1}$. However, we need to also account for the number of possible realisations of a set of size in $\{-1, 1\}^k$ that are determined by the choice of the h_i. Note that this number is bounded by $\left(\frac{em}{d}\right)^{dk}$, as the $\mathsf{VCD}(H) = d$ and at most k hypothesis from H are used. Thus, if we choose m large enough so that,

$$\left(\frac{em}{k+1}\right)^{k+1} \cdot \left(\frac{em}{d}\right)^{dk} < 2^m, \tag{4.10}$$

then m gives an upper bound on the $\mathsf{VCD}(\mathsf{THRESHOLDS}_k(H))$.

It is then straightforward to show that for a suitable constant c_0, for all $m \geq c_0 kd \log k$, Eq. (4.10) holds. □

Combining 4.3 and Theorem 3.8 we can obtain a generalisation bound on the error of the hypothesis output by Adaboost, whenever the weak learning algorithm, WeakLearn, outputs hypothesis from a class H with $\mathsf{VCD}(H) = d < \infty$.

[4] This is not a stringent requirement on a weak learning algorithm. However, this condition is not necessary and in fact it can be shown that AdaBoost generalises even without such a condition on the weak learning algorithm.

4.3 Exercises

4.1. Consider the AdaBoost algorithm described in Algorithm 4.1.

a) Show that the error of h_t with respect to the distribution D_{t+1} is exactly $1/2$.

b) What is the maximum possible value of $D_t(i)$ for some $1 \leq t \leq T$ and $1 \leq i \leq m$?

c) Fix some example, say i, let t_i be the first iteration such that $h_{t_i}(\mathbf{x}_i) = y_i$. How large can t_i be?

4.2. Give a formal proof of the statement right after Lemma 4.3.

4.3. Consider the instance space $X_n = \{0,1\}^n$ and the following hypothesis class

$$H_n = \{0, 1, z_1, \bar{z}_1, z_2, \bar{z}_2, \ldots, z_n, \bar{z}_n\}.$$

The hypothesis class, H_n, contains $2n + 2$ functions. The functions "0" and "1" are constant and predict 0 and 1 on all instances in X_n. The function "z_i" evaluates to 1 on any $\mathbf{x} \in \{0,1\}^n$ satisfying $x_i = 1$ and 0 otherwise. Likewise, the function "$\bar{z}_i$" evaluates to 1 on any $\mathbf{x} \in \{0,1\}^n$ satisfying $x_i = 0$ and 0 otherwise. Thus a single bit of the input determines the value of these functions; for this reason these functions are sometimes referred to as *dictator* functions.

a) Show that the class CONJUNCTIONS is $\frac{1}{10n}$-weak learnable using H.

Hint: The factor 10 is not particularly important, just a sufficiently large constant.

b) Let CONJUNCTIONS_k denote the class of conjunctions on at most k literals. Give an algorithm that PAC-learns CONJUNCTIONS_k and has sample complexity polynomial in k, $\log n$, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. What would be the sample complexity if you had used the algorithm for learning CONJUNCTIONS discussed in the lectures?

Hint: First show that the weak learning algorithm in the previous part can be modified to be a $\frac{1}{10k}$-weak learner in this case.

c) Show that there is no weak learning algorithm for PARITIES using H.

Chapter 5

Cryptographic Hardness of Learning

We have seen a few *efficient* learning algorithms in the PAC learning framework for concept classes such as conjunctions, decision lists and linear halfspaces. We've also studied the Occam principle that "short consistent hypotheses" generalise well on unseen data. For finite hypothesis classes the sample complexity scales polynomially with $\log |H|$, $1/\epsilon$ and $1/\delta$. When using infinite hypothesis classes, the Vapnik Chervonenkis (VC) dimension plays a similar role as $\log |H|$. We've seen upper and lower bounds on sample complexity in terms of VC-dimension that almost match. In particular, provided one can identify a hypothesis that is consistent with the observed data (as long as the sample size is large enough as a function of the VC dimension, ϵ and δ), we obtain a hypothesis that has error bounded by ϵ with respect to the target concept and distribution.

Thus, in a sense the VC-dimension captures the notion of *learnability*, if sample complexity is the only thing we care about. However, when we consider computational complexity the picture is considerably different. We've already shown that there are concept classes for which finding *proper* consistent learners is hard unless $\mathsf{RP} = \mathsf{NP}$. In the case of 3-term DNF formulae, we can avoid this hardness by choosing the output hypothesis from a larger concept class, that of 3-CNF formulae. One may wonder, whether this is always the case, i.e. can we always identify a larger hypothesis class from which we can identify a consistent learner in polynomial time?

In this chapter, we'll answer this question in the negative, provided a certain widely believed assumption in cryptography holds. We will show that there are concept classes that cannot be *efficiently* PAC-learnt, even in the case of *improper* learning, where the output hypothesis is allowed to come from any polynomially evaluatable hypothesis class.

5.1 The Discrete Cube Root Problem

Let p and q be two large primes that require roughly the same number of bits to represent. Furthermore, we'll assume that these primes are of the form $3k + 2$. Let $N = pq$ be the product of these primes. It is widely believed that factoring such an N, when p and q are chosen to be random n bit primes, cannot be

performed in time polynomial in n.[1] Let φ denote Euler's totient function, then we have $\varphi(N) = (p-1)(q-1)$. As p and q are chosen to be of the form $3k+2$, 3 does not divide $\varphi(N)$.

Let $\mathbb{Z}_N^* = \{i \mid 0 < i < N, \gcd(i, N) = 1\}$. It is well-known that $\mathbb{Z}_N^*$ forms a group under the operation of multiplication modulo N. We consider a function $f_N : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined as $f_N(y) \equiv y^3 \mod N$. As 3 does not divide $\varphi(N)$, it is straightforward to observe that f_N is a bijection. As $\gcd(3, \varphi(N)) = 1$, there exist $d, d' \geq 1$ such that $3d = \varphi(N)d' + 1$ (the existence of d, d' can be shown by a constructive proof of Euclid's algorithm to obtain the gcd). Then, we have,

$$(f_N(y))^d \equiv y^{3d} \equiv y^{\varphi(N)d'+1} \equiv y \mod N.$$

The last equality follows from Euler's Theorem, which states that $y^{\varphi(N)} \equiv 1 \mod N$ for all $y \in \mathbb{Z}_N^*$. Readers unfamiliar with elementary number theory may follow any basic text (e.g. [18]); however, most of the material in this chapter can also be understood just starting from Definition 5.2 without fully understanding the discrete cube root problem.

Definition 5.1 – **Discrete Cube Root Problem.** *Let p and q be two n-bit primes of the form $3k+2$, and let $N = pq$. Let $\varphi(N) = (p-1)(q-1)$ and note that 3 does not divide $\varphi(N)$. Given N and $x \in \mathbb{Z}_N^*$ as input, output $y \in \mathbb{Z}_N^*$, such that $y^3 \equiv x \mod N$.*

Observe that if we can factorise N, the discrete cube root problem is easy to solve. We can simply obtain $\varphi(N)$ and then find d such that $3d \equiv 1 \mod \varphi(N)$ using Euclid's algorithm. However, factoring N is believed to be hard in general and in fact, no polynomial-time algorithm that finds the discrete cube root is known. Note that in this case, polynomial-time means polynomial in n, not N. The discrete cube root problem is also widely believed to be computationally intractable. We define the formal hardness assumption and use this to show that there are concept classes that are computationally hard to learn.

Definition 5.2 – **Discrete Cube Root Assumption (DCRA).** *For any polynomial $P(\cdot)$, there does not exist any (possibly randomised) algorithm, A, that runs in time $P(n)$ and on input N and x, where N is the product of two random n bit primes of the form $3k+2$ and x is chosen uniformly at random from $\mathbb{Z}_N^*$, outputs $y \in \mathbb{Z}_N^*$ that with probability at least $1/P(n)$ satisfies $y^3 \equiv x \mod N$. The probability is over the random draws of p, q, x and any internal randomisation of A.*

Although this is not the main objective here, let us quickly observe how this assumption may be used in public-key cryptography. The integer N is the public key, and any message that can be encoded as an element of $\mathbb{Z}_N^*$ can be encrypted simply by taking its cube modulo N. Under the discrete cube root assumption, this cannot be decrypted, except if one has access to d, such that $3d \equiv 1 \mod \varphi(N)$. The integer d is the private key. The holder of the private key generates two random primes, using which they can obtain d; they only publicly release N.

[1] Note that factoring can easily be done in time polynomial in N. However, the input size is $O(n) = O(\log N)$ if the number N is provided in binary.

5.2 A learning problem based on the DCRA

Let us try to phrase the question of finding the cube root of $x \in \mathbb{Z}_N^*$ as a learning question. Let us suppose that we have access to a training sample,

$$S = \left\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\right\},$$

where $y_i^3 \equiv x_i \mod N$ for $i = 1, \ldots, m$, and x_i are drawn uniformly at random from $\mathbb{Z}_N^*$. The learning question is: Given such examples, can we obtain $h : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$, such that for x drawn uniformly at random from $\mathbb{Z}_N^*$, it holds with probability at least $1 - \delta$, that $\mathbb{P}\left[(h(x))^3 \not\equiv x \mod N\right] \leq \epsilon$?

How can these pairs (x_i, y_i) help? It is easy to see that they cannot help, as it is easy to generate these pairs ourselves. This is because, although finding the cube root is hard, finding the cube is easy. As f_N is a bijection, we can choose $y_i \in \mathbb{Z}_N^*$ uniformly at random, and then pick $x_i \equiv y_i^3 \mod N$ from $\mathbb{Z}_N^*$. Note that this implies that the distribution of x_i is uniform over $\mathbb{Z}_N^*$. Thus, clearly access to random examples of the form (x_i, y_i) where x_i is drawn from the uniform distribution over $\mathbb{Z}_N^*$ and $y_i^3 \equiv x_i \mod N$, can't help to find $h : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$, that satisfies $\mathbb{P}\left[(h(x))^3 \neq x \mod N\right] \not\equiv \epsilon$, under the DCRA.

This almost fits into our notion of PAC learning, except that the output of the target function is not in $\{0, 1\}$. This can be fixed rather easily. We know that the output of f_N^{-1} is some $2n$ bit string. Thus, we can consider $2n$ different target functions, $(f_{N,i}^{-1})_{i=1}^{2n}$, where $f_{N,i}^{-1}$ is a function which outputs the i^{th} bit of the function f_N^{-1}. If we could learn all the function, $f_{N,i}^{-1}$ to accuracy $\frac{\epsilon}{2n}$, then we can reconstruct f_N^{-1} to accuracy ϵ. Thus, if learning f_N^{-1} is hard, then at least one of the boolean functions $(f_{N,i}^{-1})_{i=1}^{2n}$ must also be hard to learn.

5.2.1 A hard-to-learn concept class

So far, we've established that if we choose random n bit primes p and q of the form $3k + 2$, there exists a boolean function, $f_{N,i}^{-1}$, such that if we get labelled examples from a *specific* distribution D over $2n$ bit strings, viz. the uniform distribution over bit representations of elements in $\mathbb{Z}_N^*$, we cannot output a (polynomially evaluatable) hypothesis h, such that $\mathbb{P}_{x \sim D}\left[f_{N,i}^{-1}(x) \neq h(x)\right] \leq \frac{\epsilon}{2n}$. If we can identify a class, C, such that $f_{N,i}^{-1} \in C_{2n}$, then this also implies that the class C is not PAC-learnable.

Let us try and understand what such a concept class could be. First, we note that if d is known, there is a rather simple polynomial time algorithm to output $f_N^{-1}(x)$. All we need to do is perform the operation $x^d \mod N$. Naïvely computing x^d is not efficient as d may be as large as $\varphi(N)$, i.e. d may itself be $2n$ bit long. The first thing we need to ensure is that all operations are repeatedly performed modulo N; this way none of the representations get too large. The second is that we start by computing, $x \mod N, x^2 \mod N, x^4 \mod N, x^8 \mod N, \ldots, x^{2^{\lfloor \log \varphi(N) \rfloor}} \mod N$, i.e. we compute $x^{2^i} \mod N$ for $i = 0, 1 \ldots, \lfloor \log \varphi(N) \rfloor$. To obtain $x^d \mod N$, we simply take the product of the terms $x^{2^i} \mod N$ such that the i^{th} bit of d is 1. This shows that there exists a circuit of polynomial size that computes f_N^{-1} where d is hard-wired into the circuit itself. In particular, this also implies that there exist polynomial-size circuits for $f_{N,i}^{-1}$ for all $i = 1, \ldots, 2n$. This gives us the following result.

Theorem 5.3. *There exists a polynomial $P(\cdot)$, such that class of concepts, C, where C_n consists of circuits of size at most $P(n)$, is not efficiently PAC-learnable under the discrete cube root assumption.*

5.2.2 Reducing the depth

We've established that under DCRA, PAC-learning polynomial-sized circuits is hard. However, in a way, this is the weakest such result one could hope for. Polynomial-sized circuits are in some sense the most expressive class that we could ever hope to learn. The question is whether there exist significantly smaller concept classes that are also hard to learn. We will outline a proof that one such class, that of concepts representable by circuits whose depth is only logarithmic in the number of inputs, is also hard to PAC-learn under DCRA. The complete proof requires showing how low-depth circuits for repeated multiplication and division can be designed; we will not see these constructions here as they are quite involved. We will highlight the basic idea and point to the appropriate references for complete details.

One of the reasons why the circuit described above is deep (it has depth $\Theta(n)$) is that it requires computing the powers $x^{2^i} \mod N$ for each value of $i = 0, 1, \ldots, \lfloor \log \varphi(N) \rfloor$; the reason being that $x^{2^{i+1}} \mod N$ can only be computed after $x^{2^i} \mod N$ has been computed. However, notice that this computation does not require the knowledge of d, the secret key, at all! So, if instead of being given the input x, we were given a longer string of length $(2n)^2$ as input, the concatenation of $(x \mod N, x^2 \mod N, x^4 \mod N, \ldots, x^{2^{\lfloor \log \varphi(N) \rfloor}} \mod N)$, the question of learning $f_{N,i}^{-1}$ would still remain intractable under the DCRA, as the additional input just represents something we could have computed ourselves in polynomial time. Note that a hard to learn target function and distribution now would involve having a distribution over strings of length $(2n)^2$, where the first $2n$ bits represent $x \in \mathbb{Z}_N^*$ chosen at the uniformly at random and the remaining bits are the powers, $x^{2^i} \mod N$, $i = 1, 2, \ldots, \lfloor \log \varphi(N) \rfloor$. It is relatively straightforward to show that there exists a circuit of depth $O((\log n)^2)$ that when given as input $(x \mod N, x^2 \mod N, \ldots, x^{2^{\lfloor \log \varphi(N) \rfloor}} \mod N)$ outputs $f_N^{-1}(x)$. This is because in order to compute $x^d \mod N$, we need to compute a product of at most $2n$ terms from the input, which can be done pairwise in parallel to have multiplication depth $O(\log n)$; however, multiplying two $2n$ bit numbers itself requires a circuit of depth $O(\log n)$, yielding an overall circuit depth of $O((\log n)^2)$. To show that there is a circuit of depth $O(\log n)$ requires more work using the techniques of Beame et al. [9]. To summarise, we have the following theorem.

Theorem 5.4. *There exists a constant $\kappa_0 > 0$, such that the class of circuits, C, where C_n consists of circuits on n inputs with depth bounded by $\kappa_0 \log n$ is not efficiently PAC-learnable under DCRA.*

5.3 Chapter Notes

The material covered in this chapter is presented in greater detail in the textbook by Kearns and Vazirani [34, Chap. 6]. Further details, including complete proofs and reductions showing cryptographic hardness of learning

other concept classes such as finite automata appear in the original paper by Kearns and Valiant [33]. The proof that the class of polynomial-size circuits cannot be efficiently PAC-learnt if one-way functions exist, even when membership queries are allowed (cf. Chap. 6.1), first appeared in the work of Goldreich et al. [27]. The assumption that one-way functions exist is much weaker than the discrete cube root assumption; indeed, it is hardly conceivable to have any cryptography at all if one-way functions don't exist. On the other hand, if the discrete cube root assumption were to be untrue, it would simply call into question the security of the RSA cryptosystem, but not rule out the existence of other kinds of cryptosystems.

In general, making stronger assumptions leads to stronger hardness results for efficient PAC-learning results. Recently, there has been work using different kinds of assumptions, not directly related to cryptography, to establish hardness of PAC-learning of much smaller classes such as DNF [22, 21, 20]. However, it is worth bearing in mind that not all of these assumptions have been as thoroughly tested; for example, recently Allen et al. [1] showed that one of the assumptions made by Daniely et al. [22] was in fact not true.

Chapter 6

Exact Learning using Membership and Equivalence Queries

In the PAC learning framework, we receive labelled examples $(\mathbf{x}, c(\mathbf{x}))$, where $\mathbf{x}$ is drawn from some distribution over the instance space X, and $c(\mathbf{x}) \in \{0, 1\}$ is the target label. Thus far, we have focused on two different questions—the first regarding sample complexity asks how much data is necessary and sufficient for learning, the second regarding computational complexity asks when learning algorithms run in polynomial time. For questions concerning sample complexity, we have seen that capacity measures such as the VC dimension give an answer that is essentially tight, in that the lower and upper bounds on sample complexity in terms of the VC dimension match each other up to constant and some logarithmic factors. On the question of computational complexity, we can make a distinction between *proper learning*, where the learning algorithm is required to output a hypothesis from the concept class that is being learnt, and *improper learning*, where the learning algorithm may output any polynomially evaluatable hypothesis. For proper learning, we have already established that even relatively simple concept classes such as 3-TERM-DNF are hard to efficiently PAC-learn unless RP = NP. However, as we have seen it may be possible to learn these classes if the learning algorithm is allowed to output hypotheses from larger classes. In the case of *improper learning*, we established in Chapter 5 that the class of log-depth circuits is not efficiently PAC-learnable under the discrete cube root assumption.

In this chapter, we will consider a richer model of learning that allows the learning algorithm to be more *active*. In addition to requesting random labelled examples from the target distribution and concept, we'll allow the learning algorithm to pick an instance $\mathbf{x} \in X$ and request the label $c(\mathbf{x})$. We'll investigate this model in greater detail and show that there are concept classes that can be *efficiently* learnt under this more powerful model of learning, that are not *efficiently* PAC-learnable under the discrete cube root assumption.[1] For stylistic reasons, it will be easier to define a new model of *exact* learning that does not require the existence of a distribution over the instance space, and also

[1] All hardness results of this kind that we can establish are conditional; they rely on assumptions such as the discrete cube root assumption or something else.

allows us to drop the accuracy parameter, ϵ and the confidence parameter, δ, from consideration. Exercise 6.2 relates this model of *exact learning* defined in Section 6.1 to an enriched model of PAC learning. We will see two algorithmic results in this new model in this chapter.

6.1 Exact Learning with Membership and Equivalence Queries

We will consider learning algorithms that are allowed to make two different types of queries: *membership queries* or *value queries*,[2] and *equivalence queries*. As we did in the case of PAC learning in Chapter 1, it is convenient to define the model in terms of oracles which may be queried by a learning algorithm.

Definition 6.1 – **Membership (Value) Query Oracle, $\mathsf{MQ}(c)$.** *A membership (or value) query oracle for a concept $c : X \to \{0,1\}$, $\mathsf{MQ}(c)$, when queried with an instance $x \in X$ returns the value $c(x)$.*

Next we define the *equivalence oracle* which takes as input (a representation of) $h : X \to \{0,1\}$ and either agrees that h is *equivalent* to the target concept c, or returns a "counterexample" x such that $h(x) \neq c(x)$, a proof that c and h are not equivalent. Formally, we define:

Definition 6.2 – **Equivalence Oracle, $\mathsf{EQ}(c)$.** *An equivalence oracle for a concept $c : X \to \{0,1\}$, $\mathsf{EQ}(c)$, when queried with a representation of a hypothesis, $h : X \to \{0,1\}$, either returns* equivalent *indicating that h and c are equivalent as boolean functions, or a counterexample $x \in X$, such that $c(x) \neq h(x)$.*

We can now define a model of *exact learning* using membership and equivalence query oracles. The goal of the learning algorithm is to obtain a hypothesis h, such that $h(x) = c(x)$ for every $x \in X$. Thus the distribution over the instance space does not play a role, and indeed in the model we consider we will not have access to any *random examples* at all. The goal of *exact* learning and the lack of a target distribution over X renders the accuracy parameter, ϵ, irrelevant. As there is no inherent randomness, we will also not include the confidence parameter, δ. However, one could make the distinction between deterministic learning algorithms and randomised learning algorithms, and in that case one would have to reintroduce the confidence parameter, δ, to account for the failure of the algorithm due to its internal random choices rather than the external randomness in the data. For simplicity, we will only define *efficient* exact learning, however, in principle one could individually account for the number of queries made by the learning algorithm and the its running time. This would allow us to investigate tradeoffs between "information complexity" and "computational complexity" of learning in the exact learning framework, as we have done in the PAC learning framework.

[2]The name membership query originated from the fact that boolean functions may be viewed as subsets of the instance space; the instances that evaluate to 1 are members of the set and those that evaluate to 0 are not. Querying the value of a boolean function at a point can be thought of as querying the membership of this point in this set. The name *value query* may be more suitable as it can be applied to non-boolean functions as well.

Definition 6.3 – **Exact Learning with $\mathsf{MQ} + \mathsf{EQ}$.** *We say that a concept class C is efficiently* exactly learnable *from membership and equivalence queries, if there exists a polynomially evaluatable hypothesis class H, a polynomial $p(\cdot, \cdot)$ and a learning algorithm L, such that for all $n \geq 1$, for all $c \in C_n$, L when given access to the oracles $\mathsf{MQ}(c)$ and $\mathsf{EQ}(c)$, and inputs n, $\text{size}(c)$, halts in time $p(n, \text{size}(c))$ and outputs a hypothesis $h \in H_n$, such that for each $x \in X_n$, $h(x) = c(x)$, i.e. h is equivalent to c. Furthermore, we required that every query made by L to $\mathsf{EQ}(c)$ is with some $h \in H_n$.*

The model of exact learning may seem quite far removed from the practice of machine learning, and in some ways it is. However, it is designed to allow us to isolate interesting results and design learning algorithms. The *key addition* is the ability to make membership queries. From a point of view of practical machine learning, one may imagine that we can identify *expert* human labellers to provide responses that would simulate an $\mathsf{MQ}(c)$ oracle;[3] it is harder to expect humans to be able to simulate an $\mathsf{EQ}(c)$ oracle. The latter however is primarily defined for mathematical convenience. Exercise 6.2 shows how any algorithm designed in the *exact learning* with $\mathsf{MQ} + \mathsf{EQ}$ framework allows one to design a PAC learning algorithm provided the algorithm has access to the membership oracle $\mathsf{MQ}(c)$. Thus, in short if one is willing to settle for a PAC-guarantee, i.e. $\text{err}(h) \leq \epsilon$ with probability at least $1 - \delta$, then access to the equivalence oracle $\mathsf{EQ}(c)$ is not necessary. The notion of PAC+MQ learning is formally defined below as Definition 6.4.

Definition 6.4 – **PAC+MQ Learning.** *For $n \geq 1$, let C_n be a concept class over instance space X_n and let $C = \bigcup_{n \geq 1} C_n$ and $X = \bigcup_{n \geq 1} X_n$. We say that C is* learnable using the hypothesis class H *in the PAC+MQ framework, if there exists an algorithm L that satisfies the following: for every $n \in \mathbb{N}$, for every concept $c \in C_n$, for every distribution D over X_n, for every $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $\mathsf{EX}(c, D)$ and $\mathsf{MQ}(c)$, and inputs n, $\text{size}(c)$, ϵ and δ, L outputs $h \in H_n$ that with probability at least $1 - \delta$ satisfies $\text{err}(h) \leq \epsilon$. The probability is over the random examples drawn from $\mathsf{EX}(c, D)$ as well as any internal randomisation of L. The number of calls made to $\mathsf{EX}(c, D)$ and $\mathsf{MQ}(c)$* (sample complexity) *must be bounded by a polynomial in n, $\text{size}(c)$, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ and H must be polynomially evaluatable.*

We further say that C is efficiently *PAC+MQ learnable using H, if the running time of L is polynomial in n, $\text{size}(c)$, $1/\epsilon$ and $1/\delta$.*

We will now focus on learning algorithms for two concept classes. In Section 6.2, we show that the class of monotone DNF formulae is efficiently exactly learnable using membership and equivalence queries. In Section 6.3, we show how to learn languages that are recognisable by deterministic finite automata (DFA), which is also the class of regular languages. In that section, it will be convenient to move somewhat away from the specific learning framework introduced above, but we will limit those changes and that discussion to that section.

[3] Recently this has become easier using various online labelling services that allow interaction with humans. There are of course still practical considerations, such as whether one should expect examples constructed by learning algorithms to be classifiable by humans. However, this distinction is not unlike many others when comparing theory and practice. Willingness to ignore such considerations is a price that we must pay in order to be able to develop the suitable theory.

6.2 Exact Learning MONOTONE-DNF using MQ + EQ

In this section, we show that the concept class MONOTONE-DNF that is not known to be PAC-learnable is in fact *exact learnable* using membership and equivalence queries. Exercise 6.1 asks you to show that the problem of learning DNF formulae and MONOTONE-DNF formulae are equivalent (up to polynomial time reductions) in the PAC Learning framework discussed in Chapter 1. However, while the results in this section together with Exercise 6.2 show that MONOTONE-DNF formulae are learnable in the PAC learning framework with access to an MQ oracle, the same is not known for learning DNF formulae. In fact, a result of Angluin and Kharitonov [4] suggests that for learning DNF formulae, access to a membership query oracle, MQ(c), does not help for learning DNF formulae under certain plausible assumptions used in cryptography. Taken together this suggests a *separation* with regards to polynomial time learnability between the PAC learning framework, or the PAC learning framework with access to an MQ oracle. However, the main evidence we have for the *hardness* of learning DNF formulae is our inability to have come up with a polynomial time learning algorithm.[4] Let us formally define the class of concepts that are represented by *monotone* DNF formulae. A term is a conjunction over the literals; we say that a term is *monotone* if the conjunction only contains *positive* literals, i.e. literals that are variables (but not their negations). A monotone DNF formula is a disjunction of monotone terms. The class MONOTONE-DNF consists of concepts that can be expressed as monotone DNF formulae. Let c be a monotone DNF formula that contains s terms. Any term T_i that is part of c can be associated with a subset $S_i \subseteq [n]$, i.e. $T_i \equiv \bigwedge_{j \in S_i} x_j$. We assume that c is of the form that if T_i and T_j are both terms of c, with S_i and S_j being the corresponding subsets of variables appearing in them, then it is not the case that $S_i \subseteq S_j$. If it were the case, dropping T_j from c would yield a formula that represents the same boolean function. We will refer to this as the *minimal* representation of c; it is not hard to show the uniqueness of *minimal* representations for monotone DNF formulae, justifying the use of the definite article "the".

The class MONOTONE-DNF$_{n,s}$ over $\{0,1\}^n$ contains boolean functions that can be represented as DNF formulae with at most s terms over n variables, and where each term only contains positive literals. Then define,

$$\text{MONOTONE-DNF} = \bigcup_{n \geq 1} \bigcup_{s \geq 1} \text{MONOTONE-DNF}_{n,s}.$$

Alg. 6.1 presents an algorithm for learning MONOTONE-DNF using membership and equivalence queries. We will prove the following theorem.

Theorem 6.5. *The class* MONOTONE-DNF *is efficiently exactly learnable using* MQ + EQ.

[4] There have been recent attempts to establish the hardness of learning DNF formulae based on assumptions from average case complexity theory. See Section 5.3 for some discussion about this. The fact that parity functions on $\log n$ bits can be represented as DNF formulae with at most n terms together with Theorem 7.6 in Chapter 7 establishes the hardness of learning DNF formulae in the *statistical query* framework. This can be viewed as further evidence of the hardness of learning DNF formulae.

Algorithm 6.1: Learning MONOTONE-DNF using MQ + EQ oracles

1 Let $\varphi \equiv 0$ // Always predict false
2 Let $s \leftarrow$ false // Determine whether we have succeeded
3 **while** $s =$ false **do**
4 Let ans be the response of EQ(c) to query φ
5 **if** ans = equivalent **then**
6 $s \leftarrow$ true
7 break
8 **else**
9 Let $\mathbf{x}$ = ans be the counterexample
10 // It must be that $\varphi(\mathbf{x}) = 0$ and $c(\mathbf{x}) = 1$
11 Let $S = \{i \mid x_i = 1\}$
12 **for** $j \in S$ **do**
13 $\mathbf{x}' \leftarrow \mathbf{x}$
14 $x'_j \leftarrow 0$
15 Let y be response to MQ(c) with query $\mathbf{x}'$
16 **if** $y = 1$ **then**
17 $\mathbf{x} \leftarrow \mathbf{x}'$
18 $T \leftarrow \{i \mid x_i = 1\}$
19 $\varphi \leftarrow \varphi \vee \left(\bigwedge_{j \in T} z_i \right)$
20 **Output:** Hypothesis φ

Proof. Let c be the target monotone DNF formula. Let n denote the number of variables and let s be the number of terms in the minimal representation of c, i.e. there isn't any term in c that implies another. The learning algorithm is allowed running time that is polynomial in n and s.

We argue that every iteration of the **while** loop on Line 3 of Alg. 6.1 finds a term T that is present in the target monotone DNF formula c, that we have not yet included in φ. First, we establish that if $\varphi(\mathbf{x}) = 1$ at any stage in the algorithm, then $c(\mathbf{x}) = 1$. Clearly, it is the case at the beginning of the algorithm; we'll show that if it holds at the beginning of the **while** loop (Line 3 of Alg. 6.1), then it continues to hold at the next iteration of the **while** loop.

Since, $\varphi(\mathbf{x}) = 1$ implies $c(\mathbf{x}) = 1$, any counterexample $\mathbf{x}$ that establishes that $\varphi \not\equiv c$ must be such that $c(\mathbf{x}) = 1$ and $\varphi(\mathbf{x}) = 0$. Let $P = \{j \mid T_j(\mathbf{x}) = 1\}$ denote the indices of terms in the minimal representation of c that are satisfied by $\mathbf{x}$. As $c(\mathbf{x}) = 1$, we know that P is non-empty. We claim that when the **for** loop on Line 11 of Alg. 6.1 ends, it is the case that there is exactly one index j in P is such that $T_j(\mathbf{x}) = 1$, where $\mathbf{x}$ is now the assignment updated in the **for** loop. Clearly, that there is at least one such index j is ensured by the **if** statement on Line 15, as $\mathbf{x}$ will never be modified to be such that $c(\mathbf{x}) = 0$. On the other hand, if there were two such indices, say j and j', then let $i \in [n]$ be the smallest index such that z_i is a literal in T_j, but not in $T_{j'}$. Setting $x_i = 0$ would have continued to have satisfied $T_{j'}$, and hence this is what would have happened in the **if** clause of Line 15. A similar argument also shows that all bits of $\mathbf{x}$ that could have been set to 0 and still have allowed $\mathbf{x}$ be a satisfying

assignment of some term T_j for $j \in P$, would have been set to 0. Thus, Line 18 finds a new term that appears in the minimal representation of c, but is not (yet) in φ and adds it to φ. This also shows that at the end of the **while** loop, it continues to be the case that $\varphi(\mathbf{x}) = 1$ implies $c(\mathbf{x}) = 1$. Since each new term added is a term that is actually a term in the *minimal* monotone DNF formula representing c, this can happen at most s times, after which the algorithm has exactly identified c.

Thus, the algorithm makes at most $s+1$ queries to $\mathsf{EQ}(c)$, and at most $n \cdot s$ queries to $\mathsf{MQ}(c)$. Clearly, the running time of the algorithm is polynomial in n and s. □

6.3 Learning DFA

In this section, we will consider the problem of learning Deterministic Finite Automata (DFA). We will deviate from the notation introduced in Definition 6.3 slightly, in that we will not require the input instances to be all strings of the same length and the output hypothesis will itself be a DFA which will recognise a language over some finite alphabet Σ.

We will introduce some notation that will be primarily restricted to this section and the corresponding exercises. Let ε denote the empty string which has length 0 and let Σ be a finite alphabet. For $i \geq 0$, we denote by Σ^i finite strings of length i using the alphabet Σ, and let Σ^* denote the set of finite strings over Σ, i.e.,

$$\Sigma^* = \bigcup_{i \geq 0} \Sigma^i.$$

We refer to elements of Σ^* as strings or *words*. Given two words $u, v \in \Sigma^*$, the word uv denotes the word obtain by concatenating the two strings.

A deterministic finite automaton is defined by a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is the finite alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the starting state, and $F \subseteq Q$ is the set of accepting states. A DFA can be represented by a directed graph with nodes denoting the states and *labelled* directed edges representing the transition function. The label associated with an edge is a letter in Σ and the (directed) out-degree of every node is exactly $|\Sigma|$. The starting state, q_0 is marked with an incoming arrow unconnected to any other node, and the accepting states are marked by using nodes with double circles. Figure 6.1 shows an example of a DFA; the DFA accepts words over the alphabet $\{0, 1\}$ where the number of times 1 appears is an integer that is 3 modulo 4. We will use this example to illustrate the algorithm in this section.

6.3.1 Access Words and Test Words

Let L be the language over Σ recognised by some DFA A. Let $|A|$ denote the number of states of A and suppose that A is the smallest such DFA recognising L. The learning algorithm will maintain a pair of sets, (Q, T), where $Q \subseteq \Sigma^*$ contains *access* words and $T \subseteq \Sigma^*$ contains *test* words. We define the notion of T-equivalence given a non-empty $T \subseteq \Sigma^*$.

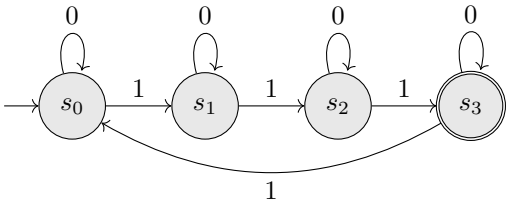


Figure 6.1: A DFA that accepts words in $\{0,1\}^*$ in which the number of times 1 appears is 3 modulo 4.

Definition 6.6. *Given a non-empty set $T \subseteq \Sigma^*$ and a language L, we say that two words $v, w \in \Sigma^*$ are T-equivalent for L, denoted by $v \equiv_T w$, if $vu \in L$ if and only if $wu \in L$ for every $u \in T$.*

It is easy to see that T-equivalence is indeed an equivalence relation as all the three properties: reflexivity, symmetry, and transitivity, are immediate. As the target language L we are seeking to learn is fixed, we will not make any dependence on L explicit. However, implicitly, all notions defined also depend on the language L.

We next define what it means for a pair (Q, T) of access and test words to be separable and closed.

Definition 6.7. *A pair (Q, T) of access and test words is* separable *if there are no two words $q, q' \in Q$ such that $q \equiv_T q'$.*

Definition 6.8. *A pair (Q, T) of access and test words is* closed *if for every $q \in Q$ and every $a \in \Sigma$, there exists some $q' \in Q$, such that $qa \equiv_T q'$.*

6.3.2 Constructing a Hypothesis Automaton

Suppose (Q, T) is a pair of access and test words and further suppose that $\varepsilon \in Q$ and that the pair (Q, T) is both *separable* and *closed*. Then, we can construct hypothesis automaton $A_{(Q,T)}$, defined to be $(Q, \Sigma, \delta, q_0, F)$, where

- $\delta(q, a) = q'$ where q' is the unique word in Q such that $qa \equiv_T a'$. The existence of q' follows from the fact that (Q, T) is closed; the uniqueness follows from the fact that (Q, T) is separable.
- $q_0 = \varepsilon$.
- $F = \{q \in Q \mid q \in L\}$. Note that F can easily be determined using access to the membership oracle, MQ for L.

Figure 6.2 shows the construction of the corresponding automata $A_{(Q,T)}$ whenever the pair is separable and closed in the simulation of Algorithm 6.2.

6.3.3 Properties of Access and Test Words

We now prove a few lemmas that will help us prove the correctness and termination in polynomial time of Algorithm 6.2.

Lemma 6.9. *Suppose that A is an automaton with the minimum number of states recognising L. Let (Q, T) be a pair of access and test words that is separable. Then $|Q| \leq |A|$.*

Proof. Suppose for the sake of contradiction $|Q| > |A|$, then there must exist two words $q, q' \in Q$ such that q and q' reach the same state in the automaton A. However, this means that for any $u \in \Sigma^*$, qu and $q'u$ will reach the same state in A. This must mean that $q \equiv_T q'$, a contradiction. □

The next lemma shows that provided the pair (Q, T) is separable, if it is not also closed, we can add some new word q' so that $(Q \cup \{q'\}, T)$ remains separable.

Lemma 6.10. *Let (Q, T) be a pair of access and test words that is separable, but not closed. Then there exists $q' \notin Q$, such that $(Q \cup \{q'\}, T)$ is separable. Furthermore such a q' can be identified in time polynomial in $|Q|$, $|T|$ and $|\Sigma|$ using access to the membership oracle, MQ, for L.*

Proof. Given set T and any set of words W, we observe that whether or not any two words in W are T-equivalent can be determined by making $|T| \cdot |W|$ membership queries. We can consider the set of words $Q \cup \{qa \mid q \in Q, a \in \Sigma\}$ which has size no greater than $|Q| \cdot (|\Sigma| + 1)$. Thus, by making at most $|Q| \cdot |T| \cdot (|\Sigma| + 1)$ queries, and in time polynomial in $|Q|$, $|T|$ and $|\Sigma|$, we can find a $q' \notin Q$, such that $(Q \cup \{q'\}, T)$ remains separable. □

Finally, the next lemma shows that if (Q, T) is separable and closed, and if the resulting automaton, $A_{(Q,T)}$ is not identical to the target automaton A, then using a counterexample, $w \in \Sigma^*$, we can find words q' and t', such that $(Q \cup \{q'\}, T \cup \{t'\})$ remains separable.

Lemma 6.11. *Suppose (Q, T) is separable and closed. If $w \in \Sigma^*$ is such the behaviour of $A_{(Q,T)}$ is different from the behaviour of A on w, i.e. w is a counterexample, then in time polynomial in $|w|$ and using at most $|w|$ membership queries, we can identify $q' \notin Q$ and t', such that $(Q \cup \{q'\}, T \cup \{t'\})$ is separable.*

Proof. Let $n = |w|$ and we consider the state transitions of $A_{(Q,T)}$ on w. We thus have,

$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \xrightarrow{w_3} \cdots \xrightarrow{w_i} q_i \xrightarrow{w_{i+1}} q_{i+1} \xrightarrow{w_{i+2}} \cdots \xrightarrow{w_n} q_n.$$

Above the states q_i, represented by words in Q, may not be distinct, so we use the pair (q_i, i) to distinguish possible multiple occurrences of the same state. We say that the pair (q_i, i) is *correct* if $q_i w_{i+1} \cdots w_n \in L$ if and only if $w \in L$. The fact that $q_0 = \varepsilon$ implies that $(q_0, 0)$ is correct, and the fact that w is a counterexample implies that (q_n, n) is not correct. Thus, there must be some $i \in \{0, \ldots, n-1\}$, such that (q_i, i) is correct and $(q_{i+1}, i+1)$ is incorrect. We can identify one such i by making no more than $|w|$ membership queries.

Having identified such an i, let $q' = q_i w_{i+1}$ and let $t' = w_{i+2} \cdots w_n$. Setting $Q' = Q \cup \{q'\}$ and $T' = T \cup \{t'\}$, in order to show that (Q', T') is separable, we need to show that no two words in Q' are T'-equivalent. Obviously, since $T \subseteq T'$ and since (Q, T) was separable, no two words in Q can be T'-equivalent.

Also, for the same reason, the only word $q \in Q$ that can possibly be T'-equivalent to q', is q_{i+1}. This is because we know that $q_i w_{i+1} \equiv_T q_{i+1}$, so if $q_i w_{i+1} \equiv_{T'} q_j$, then $q_j \equiv_T q_{i+1}$, which can only happen if $j = i + 1$ by separability of (Q, T). However, the addition of t' to T to obtain T', ensures that $q_i w_{i+1} \not\equiv_{T'} q_{i+1}$, as (q_i, i) was correct and $(q_{i+1}, i + 1)$ was not. Thus, (Q', T') is separable as required. □

6.3.4 The L^* Algorithm

Algorithm 6.2: L^*: Learning DFA using MQ + EQ oracles

```
1  Let Q ← {ε}, T ← {ε}
2  while true do
3      while (Q, T) not separable and closed do
4          Use Lemma 6.10 to obtain q'
5          Q ← Q ∪ {q'}
6      // (Q, T) now separable and closed
7      Let A_(Q,T) be hypothesis automaton
8      Let ans ← EQ(A_(Q,T))
9      if ans = equivalent then
10         break
11     else
12         Let w ∈ Σ* be the counterexample
13         Use Lemma 6.11 to obtain q', t'
14         Q ← Q ∪ {q'}
15         T ← T ∪ {t'}
16 Output: Hypothesis A_(Q,T)
```

Algorithm 6.2 shows the L^* algorithm of Angluin [3]. Figure 6.2 shows the simulation of this algorithm when the target language is the one recognised by the automaton in Figure 6.1.

The main result we prove here is the following.

Theorem 6.12. *Let $L \subseteq \Sigma^*$ be a regular language over Σ and let A be an automaton with the fewest states recognising L. Algorithm 6.2 when given access to a membership oracle, MQ, and equivalence oracle, EQ, for L, outputs a hypothesis automaton that recognises L. The running time of the algorithm is polynomial in $|A|$, Σ and n, where n is the length of the longest counterexample returned by EQ.*

Proof. We first prove the termination condition. Line numbers referred to are the ones in Algorithm 6.2. We note that as long as (Q, T) is not separable and closed, the loop in Line 2 increases the size of $|Q|$ by at least 1. Likewise if we are in the **else** part on Line 10, then the size of $|Q|$ increases by at least 1. Lemma 6.9 shows that the size of $|Q| \leq |A|$, as a result the algorithm must terminate correctly with automaton $A_{(Q,T)}$ exactly recognising the target language L.

The running time argument follows from Lemma 6.10 and Lemma 6.11. □

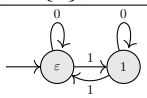
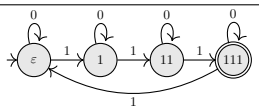
#	State	Action
1	$Q = \{\varepsilon\}, T = \{\varepsilon\}$	
2	Receive counterexample 111	Set $Q = Q \cup \{1\}, T = T \cup \{11\}$
3	$Q = \{\varepsilon, 1\}, T = \{\varepsilon, 11\}$	
4	Receive counterexample 111	Set $Q = Q \cup \{11\}, T = T \cup \{1\}$
5	$Q = \{\varepsilon, 1, 11\}, T = \{\varepsilon, 1, 11\}$ Make (Q, T) closed	Set $Q = Q \cup \{111\}, T = T$
6	$Q = \{\varepsilon, 1, 11, 111\}, T = \{\varepsilon, 1, 11\}$	

Figure 6.2: Simulation of the L^* algorithm (Alg. 6.2) with the target automaton from Fig. 6.1

.

6.4 Exercises

6.1 The class $\mathsf{MONOTONE\text{-}DNF}_{n,s}$ over $\{0,1\}^n$ contains boolean functions that can be represented as DNF formulae with at most s terms over n variables, and where each term only contains positive literals. Then define,

$$\mathsf{MONOTONE\text{-}DNF} = \bigcup_{n \geq 1} \bigcup_{s \geq 1} \mathsf{MONOTONE\text{-}DNF}_{n,s}.$$

The class DNF is defined analogously, except that the literals in the terms may also be negative. An efficient learning algorithm is allowed time polynomial in n, s, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Show that if the class $\mathsf{MONOTONE\text{-}DNF}$ is efficiently PAC-learnable, then so is DNF.

6.2 Let C be a concept class that is exactly efficiently learnable using membership and equivalence queries. We will consider the learnability of C in the standard PAC framework. Prove that if in addition to access to the example oracle, $\mathsf{EX}(c, D)$, the learning algorithm is allowed to make membership queries, then C is *efficiently* PAC-learnable. Formally, show that there exists a learning algorithm that for all $n \geq 1$, $c \in C_n$, D over X_n, $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, that with access to the oracle $\mathsf{EX}(c, D)$ and the membership oracle for c and with inputs ϵ, δ and size(c), outputs h that with probability at least $1 - \delta$ satisfies $\mathrm{err}(h) \leq \epsilon$. The running time of L should be polynomial in n, size(c), $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ and the h should be from a hypothesis class H that is polynomially evaluatable.

6.5 Chapter Notes

The L^* algorithm was first presented by Angluin [3]. The presentation we have used is largely based on [34, Chap. 8]. We have not attempted to give an efficient implementation beyond being polynomial time, however, considerable savings can be achieved by moving away from naïve implementations of the lemmas in Section 6.3.3. For details the reader may refer to [34, Chap. 8].

In Chapter 5, it was shown that the class of functions that can be represented using circuits of depth $O(\log n)$ are not efficiently PAC learnable under the DCRA. Through a sequence of reductions, it can be shown that DFA, even ones where all accepting paths have a fixed length n, cannot be efficiently under the DCRA. Thus, the result in Section 6.3 shows that the PAC+MQ model is strictly more powerful from the point of view of *efficient learning* under the DCRA.

Theorem 6.5 shows that MONOTONE-DNF is efficiently exactly learnable using membership and equivalence queries, and hence also efficiently PAC+MQ-learnable. On the other hand, there is no known algorithm for PAC-learning DNF even when membership queries are allowed. In fact, under a suitable cryptographic assumption, it has been shown that PAC-learning DNF with or without membership queries is equivalent [4]. In the absence of membership queries, Exercise 6.1 shows that learning DNF can be reduced to learning MONOTONE-DNF. These observations provide further evidence that allowing membership queries is powerful, in the sense that concept classes that are likely not *efficiently* learnable in the PAC framework without membership queries, become *efficiently* learnable when membership queries are allowed.

Chapter 7

Statistical Query Learning

The learning frameworks we've studied so far have assumed that the learning algorithms have access to *perfectly labelled data*. This is clearly not an accurate reflection of what one would encounter when accessing data in the real-world. In this chapter, we shall study a model that seeks to capture noise or imperfections in the observed data. Some of the learning algorithms we've studied are not robust to noise; we'll study modifications to these algorithms that make them robust to noise. For some concept classes, there is evidence that while these are learnable in the absence of noise, there may not exist any efficient algorithms for learning these classes in the presence of noise.

7.1 Random Classification Noise Model

We begin by studying what can be considered as a *uniform label noise* model. The data we receive has labels flipped independently with probability η. Let us define the example oracle with random classification noise formally.

Definition 7.1 **– RCN Example oracle, $\mathsf{EX}^{\eta}(c, D)$.** *An example oracle with random classification noise rate η, for a concept c over instance space X and for distribution D over X, denoted by $\mathsf{EX}^{\eta}(c, D)$, when queried does the following: $\mathbf{x} \in X$ is drawn according to the distribution D, independently of all other random choices, with probability $1 - \eta$, it returns $(\mathbf{x}, c(\mathbf{x}))$ and with probability η returns $(\mathbf{x}, 1 - c(\mathbf{x}))$.*

Before we study algorithms that can be implemented using such a noisy example oracle, let us make a couple of observations. The noise is assumed to be independent in each example; the case when noise could be correlated or a function of the instance can be much harder to deal with. Clearly, no learning algorithm can succeed when the noise rate, η, equals $\frac{1}{2}$, as the labels have nothing to do with the target concept in that case. We'll assume that $\eta < \frac{1}{2}$; if $\eta > \frac{1}{2}$, we are just trying to learn the flipped concept. The measure of interest is $1 - 2\eta$: the smaller the value, the harder the learning problem. Thus, we will allow the sample complexity and running time of our algorithms to depend polynomially on $\frac{1}{1-2\eta}$. The requirement for the output hypothesis h remains exactly the same as in the PAC-learning framework, we want $\mathrm{err}(h; c, D) = \mathbb{P}_{\mathbf{x} \sim D}\left[h(\mathbf{x}) \neq c(\mathbf{x})\right] \leq \epsilon$, i.e. we are comparing ourselves to the *true* target

function (not with a noisy target). Let us formally define the model of PAC-learning in the presence of random classification noise.

***Definition 7.2* – PAC Learning with Random Classification Noise.** *For $n \geq 1$, let C_n be a concept class over instance space X_n and let $C = \bigcup_{n\geq 1} C_n$ and $X = \bigcup_{n\geq 1} X_n$. We say that C is* PAC learnable with Random Classification Noise (RCN) *using the hypothesis class H if there exists an algorithm L that satisfies the following: for every $n \in \mathbb{N}$, for every concept $c \in C_n$, for every distribution D over X_n, for every $0 < \epsilon < 1/2$, for every $0 < \delta < 1/2$, and for every $0 \leq \eta < 1/2$, if L is given access to $\mathsf{EX}^{\eta}(c, D)$ and inputs n, $\mathrm{size}(c)$, ϵ, δ, and η_0, such that $0 \leq \eta \leq \eta_0 < 1/2$, L outputs $h \in H_n$ that with probability at least $1 - \delta$ satisfies $\mathrm{err}(h; c, D) \leq \epsilon$. The probability is over the random examples drawn from $\mathsf{EX}^{\eta}(c, D)$ as well as any internal randomisation of L. The number of calls made to $\mathsf{EX}(c, D)$* (sample complexity) *must be bounded by a polynomial in n, $\mathrm{size}(c)$, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ and $\frac{1}{1-2\eta_0}$ and H must be polynomially evaluatable.*

We further say that C is efficiently *PAC learnable with RCN using H, if the running time of L is polynomial in n, $\mathrm{size}(c)$, $1/\epsilon$, $1/\delta$ and $1/(1 - 2\eta_0)$.*

In the definition above, instead of the (perfect) example oracle $\mathsf{EX}(c, D)$ introduced in the PAC learning framework, the learning algorithm has access to the noisy oracle, $\mathsf{EX}^{\eta}(c, D)$. The learning algorithm is also given as input a parameter $\eta_0 < \frac{1}{2}$, which is an upper bound on the noise rate. It is straightforward to show that this input, η_0, is not really required, the learning algorithm can simply try all possible upper bounds in a systematic way and then test which of the produced hypotheses is the (almost) best one (see Exercise 5.4 in [34, Chap. 5]).

7.1.1 Learning Conjunctions

Let us revisit one of the the first algorithms we studied, Algorithm 1.1 to learn CONJUNCTIONS from Section 1.3, when there was no noise in the data. The algorithm starts with a hypothesis consisting of a conjunction of all $2n$ literals; thus it begins with a hypothesis that always predicts 0. Then for every positive example $(\mathbf{x}, 1)$, it deletes the literals present in its hypothesis that cause this example to be classified as negative. As long as sufficiently many examples are used, it is guaranteed that this algorithm outputs a hypothesis that has error at most ϵ.

This algorithm does not work when there is noise in the data. For instance, if an example that was a negative example was observed with label 1 (due to noise), the algorithm may drop several literals from the hypothesis that are actually required. The decisions made by the algorithm are not *robust* as they are based on a single example. We will design a more robust algorithm for learning conjunctions. To begin with, let us continue to assume that the data we receive is noise-free; later, we'll discuss how this more robust algorithm can also be used when the data is noisy.

Let c be the target conjunction and let ℓ be a literal that appears in c. We will use the notation $\ell(\mathbf{x}) = 1$ to indicate that the literal ℓ evaluates to 1 (true) on the instance $\mathbf{x} \in X$. For any literal ℓ that is present in the target conjunction, it holds that $\mathbb{P}_{\mathbf{x}\sim D}\left[\ell(\mathbf{x}) = 0 \wedge c(\mathbf{x}) = 1\right] = 0$. We would like to identify all such literals and put them in the output hypothesis. However, the

literals that are truly *harmful* are those that would cause us to make lots of errors. Let us make this idea more concrete.

- A literal ℓ is harmful if $\mathbb{P}_{\mathbf{x}\sim D}\left[\ell(\mathbf{x}) = 0 \wedge c(\mathbf{x}) = 1\right] \geq \frac{\epsilon}{8n}$

Let h be a hypothesis that is a conjunction of all literals that are not harmful. By definition, literals that appear in c are not harmful so they all appear in h. Thus, we are once again in the scenario there $h(\mathbf{x}) = 1$ implies $c(\mathbf{x}) = 1$. So the only possible errors occur when $h(\mathbf{x}) = 0 \wedge c(\mathbf{x}) = 1$. Any such error must be due to some literal, ℓ, and the corresponding event $\ell(\mathbf{x}) = 0 \wedge c(\mathbf{x}) = 1$. Thus, we have,

$$\mathrm{err}(h) = \mathbb{P}_{\mathbf{x}\sim D}\left[h(\mathbf{x}) = 0 \wedge c(\mathbf{x}) = 1\right]$$

If $h(\mathbf{x}) = 0$, but $c(\mathbf{x}) = 1$, it must be due to a literal ℓ that is not harmful which was added to h.

$$\begin{aligned}\mathrm{err}(h) &\leq \sum_{\ell \text{ not harmful}} \mathbb{P}_{\mathbf{x}\sim D}\left[\ell(\mathbf{x}) = 0 \wedge c(\mathbf{x}) = 1\right] \\ &\leq 2n \cdot \frac{\epsilon}{2n} = \epsilon\end{aligned}$$

We haven't said how exactly to find significant and harmful literals; however, it is easy to see that they could be identified *almost* correctly with high probability. The size of the sample required to guarantee correctness can be obtained using Hoeffding's Inequality (A.2), and it is polynomial in n, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. More precisely, what this boils down to is using the observation that we can get an empirical estimate, $\widehat{p}$, of the true probability, p, of some event with the guarantee that with probability at least $1 - \delta'$, $|\widehat{p} - p| \leq \tau$, by using a sample of size $\Theta\left(\frac{1}{\tau^2}\log\frac{1}{\delta'}\right)$. The events in consideration are $(\ell(\mathbf{x}) = 0 \wedge c(\mathbf{x}) = 1)$ for all possible ℓ. Ignoring the failure case whose probability is bounded by δ', we know that if $\widehat{p} < \epsilon/(8n)$, then $p < \epsilon/(8n) + \tau$. Say a literal is *approximately harmful* if $\widehat{p} \geq \epsilon/(8n)$; clearly if a literal is not approximately harmful, it is also not harmful if $\tau \leq \epsilon/(16n)$. On the other hand if $\tau \leq \epsilon/(16n)$ any literal that is not deemed to be *approximately harmful* is also not in the target conjunction c. Thus, provided we chose $\tau \leq \epsilon/(16n)$, if the algorithm outputs a hypothesis h that is a conjunction of all literals that are not *approximately harmful*, we still have $\mathrm{err}(h) \leq \epsilon$ with high probability. We will choose $\delta' = \delta/(2n)$ so that even after considering a union over all failure events, the probability of the combined failure event is bounded by δ. From now on, we will avoid spelling out such intricacies in full detail every time; it will be assumed that the reader can fill in the details. Compared to Algorithm 1.1 from Section 1.3, this new algorithm is more robust. A single example is not used to determine whether or not a literal should be present in the target hypothesis; this decision is made using aggregate statistics.

It is not at all hard to show that if we know the noise rate η, these probability estimates can be obtained even when receiving samples from $\mathsf{EX}^{\eta}(c, D)$, rather than $\mathsf{EX}(c, D)$. The sample complexity (and hence also the computational complexity) will be worse by a factor of $1/(1-2\eta)^2$. However, rather that showing how these estimates can be obtained with only access to $\mathsf{EX}^{\eta}(c, D)$ in each case separately, we will study a model that formalises this idea of using

statistics to design learning algorithms and show that any algorithm in this framework can always be simulated with access to the noisy example oracle, $\mathsf{EX}^{\eta}(c, D)$.

7.2 Statistical Query Model

In the statistical query model, the learning algorithm is not given any access to examples at all, but instead is given access to a *statistical query* oracle, $\mathsf{STAT}(c, D)$. As was the case in PAC-learning, let X be the instance space, $c : X \to \{0, 1\}$ the target concept, and D the target distribution over X. A statistical query is a tuple, (χ, τ), where $\chi : X \times \{0, 1\} \to \{0, 1\}$ is a boolean function that takes as input an instance $\mathbf{x} \in X$ and a bit $b \in \{0, 1\}$ (one of the two possible labels of the instance), and τ is the tolerance parameter. The response of the oracle, $\mathsf{STAT}(c, D)$, to the query (χ, τ), is a value $\widehat{v} \in [0, 1]$, such that,

$$\left| \mathop{\mathbb{E}}_{\mathbf{x} \sim D} \left[\chi(\mathbf{x}, c(\mathbf{x})) \right] - \widehat{v} \right| \leq \tau.$$

Learning Conjunctions using Statistical Queries

Before, we formally define the notion of learning using statistical queries, let us see how the algorithm we described above can be implemented using statistical queries. The main task was to identify literals that are approximately harmful. This can be done easily, for a literal, ℓ, define the query function, $\chi_\ell : X \times \{0, 1\} \to \{0, 1\}$, as follows:

$$\chi_\ell(\mathbf{x}, b) = \begin{cases} 1 & \text{if } \ell(\mathbf{x}) = 0 \text{ and } b = 1 \\ 0 & \text{otherwise} \end{cases}$$

Thus, $\chi_\ell(\mathbf{x}, b) = \mathbb{1}(\ell(\mathbf{x}) = 0 \wedge b = 1)$, and hence,

$$\mathop{\mathbb{E}}_{\mathbf{x} \sim D} \left[\chi_\ell(\mathbf{x}, c(\mathbf{x})) \right] = \mathbb{P}_{\mathbf{x} \sim D} \left[\ell(\mathbf{x}) = 0 \wedge c(\mathbf{x}) = 1 \right].$$

We set the tolerance parameter $\tau = \frac{\epsilon}{16n}$. Let $\widehat{v}_\ell$ be the response received from $\mathsf{STAT}(c, D)$ to $(\chi_\ell, \frac{\epsilon}{16n})$, then we treat all literals for which $\widehat{v}_\ell \geq \frac{\epsilon}{8n}$ as approximately harmful. This guarantees that any literal that we call as not *approximately harmful* satisfies

$$\mathbb{P}_{\mathbf{x} \sim D} \left[\ell(\mathbf{x}) = 0 \wedge c(\mathbf{x}) = 1 \right] \leq \frac{\epsilon}{16n}.$$

Thus, the $\mathsf{STAT}(c, D)$ oracle can be used to identify approximately harmful literals. The exact details of choosing the tolerance values and completing the proof are left as an exercise.

7.2.1 Statistical Query Learnability

Let us formally define the notion of learning with statistical queries.

Definition 7.3 – **Efficient Statistical Query (SQ) Learnability.** *Let C be a concept class and H a polynomially evaluatable hypothesis class. We say*

that C is efficiently learnable *from* statistical queries *using H, if there exists a learning algorithm, L, and polynomials $p(\cdot,\cdot,\cdot)$, $q(\cdot,\cdot,\cdot)$, and $r(\cdot,\cdot,\cdot)$, such that, for all $n \in \mathbb{N}$, for every target concept $c \in C_n$, for every distribution D over X_n, for $0 < \epsilon < \frac{1}{2}$, L with access to the statistical query oracle, $\mathsf{STAT}(c, D)$, and inputs n, ϵ and $\mathrm{size}(c)$, satisfies the following:*

- *For any query (χ, τ) made by L, the predicate χ can be evaluated in time $q(n, \mathrm{size}(c), \frac{1}{\epsilon})$ and $\frac{1}{\tau}$ is bounded by $r(n, \mathrm{size}(c), \frac{1}{\epsilon})$*
- *L halts in time bounded by $p(n, \mathrm{size}(c), \frac{1}{\epsilon})$*
- *L outputs $h \in H$, such that $\mathrm{err}(h; c, D) \leq \epsilon$*

The confidence parameter, δ, doesn't appears in the definition above; this is because we require the statistical query oracle, $\mathsf{STAT}(c, D)$, to return a value that is within the tolerance with probability 1. We could extend the definition of statistical query learnability to allow randomised algorithms, in which case the confidence parameter would be required to bound the failure of the algorithm itself. Let us first establish the relatively simple result that any concept class that is efficiently SQ-learnable is also efficiently PAC-learnable. We'll only provide a sketch of the proof, but the main idea is that with access to the example oracle, $\mathsf{EX}(c, D)$, the algorithm can simulate $\mathsf{STAT}(c, D)$ with high probability.

Theorem 7.4. *If C is efficiently SQ-learnable using H, then C is efficiently PAC-learnable using H.*

Proof. Let A be the algorithm that learns C using H in the SQ model. Let k be an upper bound on the total number of queries made to the $\mathsf{STAT}(c, D)$ oracle by A. We simulate the algorithm A; every time a query (χ, τ) is made, we draw $m = \Theta(\frac{1}{\tau^2} \log \frac{k}{\delta})$ fresh examples from $\mathsf{EX}(c, D)$, say $(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, c(\mathbf{x}_m))$, and return $\frac{1}{m} \sum_{i=1}^{m} \chi(\mathbf{x}_i, c(\mathbf{x}_i))$. Using the Hoeffding's Inequality (A.2), we know that with probability at least $1 - \frac{\delta}{k}$, the following holds:

$$\left| \frac{1}{m} \sum_{i=1}^{m} \chi(\mathbf{x}_i, c(\mathbf{x}_i)) - \mathop{\mathbb{E}}_{\mathbf{x} \sim D} \left[\chi(\mathbf{x}, c(\mathbf{x})) \right] \right| \leq \tau.$$

When A halts, we simply output the hypothesis h that was produced by A. By using the union bound and as A makes at most k queries, we know that with probability at least $1 - \delta$, all simulations of the statistical query oracle used to provide responses to A are valid, and hence the output hypothesis h satisfies, $\mathrm{err}(h; c, D) \leq \epsilon$. □

The more surprising result that we prove next is that the statistical query oracle can be simulated even with access to noisy examples, i.e. the oracle $\mathsf{EX}^{\eta}(c, D)$. This automatically implies that all concept classes learnable in the SQ framework are also PAC-learnable with random classification noise. This formalises the intuition that *robust* algorithms that make decisions on the basis of statistics rather than individual examples can be adapted to work with noisy data.

7.2.2 Simulating $\mathsf{STAT}(c, D)$ using $\mathsf{EX}^{\eta}(c, D)$

In order to make the mathematical manipulations simpler, we'll assume that the output of boolean functions are in the set $\{-1, 1\}$, rather than the more common $\{0, 1\}$. For reasons that will be clear later, we'll map 0 to 1, and 1 to -1. We will also require that the query, χ, is defined as, $\chi : X \times \{-1, 1\} \to \{-1, 1\}$. Note that this still allows us to compute, $\mathbb{P}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, c(\mathbf{x})) = -1\right]$, which is what we want as part of the statistical query learning framework, rather easily. To see this, observe that:

$$\mathbb{P}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, c(\mathbf{x})) = -1\right] = \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\frac{1 - \chi(\mathbf{x}, c(\mathbf{x}))}{2}\right] = \frac{1}{2} - \frac{1}{2}\mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, c(\mathbf{x}))\right]$$

Furthermore, for any query, $\chi : X \times \{-1, 1\} \to \{-1, 1\}$, we can express this as follows:

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, c(\mathbf{x}))\right] = \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, 1) \cdot \mathbb{1}(c(\mathbf{x}) = 1)\right] + \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, -1) \cdot \mathbb{1}(c(\mathbf{x}) = -1)\right]$$

As $c(\mathbf{x}) \in \{-1, 1\}$, $\mathbb{1}(c(\mathbf{x}) = 1) = \frac{1+c(\mathbf{x})}{2}$; similarly, $\mathbb{1}(c(\mathbf{x}) = -1) = \frac{1-c(\mathbf{x})}{2}$. Thus,

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, c(\mathbf{x}))\right] = \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, 1) \cdot \frac{1 + c(\mathbf{x})}{2}\right] + \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, -1) \cdot \frac{1 - c(\mathbf{x})}{2}\right]$$

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, c(\mathbf{x}))\right] = \frac{1}{2}\left(\mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, 1)\right] + \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, -1)\right] + \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, 1)c(\mathbf{x})\right] - \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\chi(\mathbf{x}, -1)c(\mathbf{x})\right]\right)$$

We observe that $\chi(\cdot, 1)$ and $\chi(\cdot, -1)$ are simply functions from $X \to \{-1, 1\}$. The first two expectations on the RHS above don't depend on the target concept at all; the last two compute the correlation between some function from $X \to \{-1, 1\}$ and the target. Formally, we allow the learning algorithm to make two kinds of queries—*target independent* and *correlational*. A target-independent query is of the form (ψ, τ), where $\psi : X \to \{-1, 1\}$ and $\tau \in (0, 1)$, and it receives an answer $\widehat{v}_{\psi}$ from $\mathsf{STAT}(c, D)$, such that $|\mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\psi(\mathbf{x})\right] - \widehat{v}_{\psi}| \leq \tau$. A correlational query is also of the form (φ, τ), where $\varphi : X \to \{-1, 1\}$ and $\tau \in (0, 1)$; in this case, however, the response $\widehat{v}_{\varphi}$ of $\mathsf{STAT}(c, D)$, satisfies, $\left|\mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\varphi(\mathbf{x})c(\mathbf{x})\right] - \widehat{v}_{\varphi}\right| \leq \tau$. Thus, it suffices to show that we can simulate responses of $\mathsf{STAT}(c, D)$ to target-independent and correlational queries using the noisy oracle $\mathsf{EX}^{\eta}(c, D)$. For target-independent queries, the label noise is irrelevant, as the response to the query doesn't depend on the target at all. We only need examples drawn from the distribution D; these are obtained by simply ignoring the labels of the observed data. We will now show how to simulate the responses of $\mathsf{STAT}(c, D)$ using examples from $\mathsf{EX}^{\eta}(c, D)$ for correlational queries.

Simulating responses to correlational queries

Let $B(\eta)$ denote a random variable taking values in $\{-1, 1\}$, which takes the value 1 with probability $1 - \eta$ and -1 with probability η. Let $(\mathbf{x}, c(\mathbf{x}))$ denote

a random example drawn from $\mathsf{EX}(c, D)$, let $\xi_{\mathbf{x}} \sim B(\eta)$ be a random variable independent of everything else, then $(\mathbf{x}, c(\mathbf{x})\xi_{\mathbf{x}})$ is distributed as a random example drawn from $\mathsf{EX}^{\eta}(c, D)$. Consider the following:

$$\begin{aligned}
\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathsf{EX}^{\eta}(c,D)}\left[\varphi(\mathbf{x})\cdot y\right] &= \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\mathop{\mathbb{E}}_{\xi_{\mathbf{x}}\sim B(\eta)}\left[\varphi(\mathbf{x})c(\mathbf{x})\xi_{\mathbf{x}}\right]\right] \\
&= \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\varphi(\mathbf{x})c(\mathbf{x})\mathop{\mathbb{E}}_{\xi_{\mathbf{x}}\sim B(\eta)}\left[\xi_{\mathbf{x}}\right]\right] \quad \text{As } \xi_{\mathbf{x}} \text{ is independent} \\
&= (1-2\eta)\cdot\mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\varphi(\mathbf{x})c(\mathbf{x})\right]
\end{aligned}$$

Suppose we draw m examples from $\mathsf{EX}^{\eta}(c, D)$, say $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ and let $\widehat{v} = \frac{1}{m}\sum_{i=1}^{m}\varphi(\mathbf{x}_i)y_i$. Let m be chosen to be large enough such that

$$|\widehat{v} - \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathsf{EX}^{\eta}(c,D)}\left[\varphi(\mathbf{x})y\right]| \leq \tau_1$$

with probability at least $1-\delta$. Suppose that we don't have access to the exact value of η, but only to some $\widehat{\eta} \leq \eta_0$ (where η_0 is an upper-bound on the noise rate) such that $|\widehat{\eta} - \eta| \leq \Delta/2$. We have the following:

$$\begin{aligned}
&\left|\frac{\widehat{v}}{1-2\widehat{\eta}} - \mathop{\mathbb{E}}_{\mathbf{x}\sim D}\left[\varphi(\mathbf{x})c(\mathbf{x})\right]\right| \\
&\quad\leq \left|\frac{\widehat{v}}{1-2\widehat{\eta}} - \frac{\widehat{v}}{1-2\eta} + \frac{\widehat{v}}{1-2\eta} - \frac{\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathsf{EX}^{\eta}(c,D)}\left[\varphi(\mathbf{x})y\right]}{1-2\eta}\right| \\
&\quad\leq |\widehat{v}|\cdot\left|\frac{1}{1-2\widehat{\eta}} - \frac{1}{1-2\eta}\right| + \frac{1}{1-2\eta}\cdot\left|\widehat{v} - \mathop{\mathbb{E}}_{(x,y)\sim\mathsf{EX}^{\eta}(c,D)}\left[\varphi(x)y\right]\right| \\
&\quad\leq \frac{2\Delta}{(1-2\eta_0)^2} + \frac{\tau_1}{1-2\eta_0}
\end{aligned}$$

If Δ and τ_1 are chosen so that the RHS above is at most τ, then we have successfully simulated the response of the statistical query oracle. This can be achieved, for instance by setting $\Delta = \tau(1-2\eta_0)^2/4$ and $\tau_1 = \tau(1-2\eta_0)/2$. Choosing a sample size m that is polynomially large in n, $\frac{1}{1-2\eta_0}$, $\frac{1}{\delta}$ and $\frac{1}{\tau}$ suffices to achieve this with high probability. In order to find a suitable $\widehat{\eta}$, we simply run the algorithm with all possible values of $\widehat{\eta} = i\Delta$ for $i = 1, \ldots, \lfloor\frac{\eta_0}{\Delta}\rfloor$. Clearly, one of the values among these satisfies the properties that we require of $\widehat{\eta}$. When we run the procedure for that particular value, we will obtain a h that with high probability satisfies, $\mathrm{err}(h; c, D) \leq \epsilon$. We obtain $h_1, h_2, \ldots, h_{\lfloor\frac{\eta_0}{\Delta}\rfloor}$, and we know that with high probability at least one of theses hypotheses is *good*, in that it has error at most ϵ. In order to identify the best (or good enough) hypothesis from this set, we can simply test each of them on a fresh sample drawn from $\mathsf{EX}^{\eta}(c, D)$. Simply using the hypothesis h that has the smallest *empirical* error does the trick. This is because for any h, we have,

$$\begin{aligned}
\mathbb{P}_{(\mathbf{x},y)\sim\mathsf{EX}^{\eta}(c,D)}\left[h(\mathbf{x}) \neq y\right] &= (1-\eta)\cdot\mathbb{P}_{\mathbf{x}\sim D}\left[h(\mathbf{x}) \neq c(\mathbf{x})\right] + \eta\cdot\mathbb{P}_{\mathbf{x}\sim D}\left[h(\mathbf{x}) = c(\mathbf{x})\right]. \\
&= (1-\eta)\cdot\mathrm{err}(h; c, D) + \eta\cdot(1-\mathrm{err}(h;, c, D) \\
&= \eta + (1-2\eta)\mathrm{err}(h; c, D).
\end{aligned}$$

So even if we evaluate empirical error estimates of $h_1, \ldots, h_{\lfloor \frac{\eta_0}{\Delta} \rfloor}$ using $\mathsf{EX}^{\eta}(c, D)$, provided we pick a large enough sample, using Hoeffding's inequality (A.2), it can be shown that picking h_i that has the least empirical error estimate suffices.

We've described all the main ingredients of the proof of the following theorem. Writing the proof in full detail is left as an exercise to the interested reader.

Theorem 7.5. *If C is efficiently SQ-learnable using H, then C is efficiently PAC-learnable with random classification noise using H.*

7.3 A hard-to-learn concept class

In Section 7.1.1, we designed an algorithm for learning conjunctions using only statistical queries. In Exercise 7.1, you are asked to design an SQ algorithm for learning axis-aligned rectangles in the plane. In fact, most of the efficient PAC learning algorithms we've seen so for, those for learning decision lists, 3-CNF formulae, etc. have analogues that use only statistical queries. Given this one may wonder, if in fact, every concept class that is PAC-learnable is also SQ-learnable. We answer the question in the negative. We show that the class PARITIES is not efficiently learnable using statistical queries. Formally, we will prove the following theorem in this section.

Theorem 7.6. *Any algorithm for learning* PARITIES *using statistical queries with $\epsilon = \frac{1}{10}$, when learning a target from* $\mathsf{PARITIES}_n$*, and which makes queries of the form (χ, τ), where $\tau \geq \tau_0$ for each query, must make $\Omega(\tau_0^2 \cdot 2^n)$ queries.*

Before we sketch a proof of the result, let us look at the statement of the theorem in greater detail. Efficient SQ learnability requires that the tolerance parameter for any query not be too small. In particular for efficient learning parities to accuracy $\frac{1}{10}$, we require that $\frac{1}{\tau_0}$ be bounded by a polynomial in n (in the case of parities $\text{size}(c) = O(n)$). The statement of the theorem implies that if the inverse tolerance for all queries is bounded by some polynomial in n, the algorithm must make $2^{\Omega(n)}$ queries! In particular, this rules out a polynomial time algorithm for learning PARITIES in the statistical query model. Furthermore, observe that unlike the result where we showed that *proper* learning 3-TERM-DNF is hard unless RP = NP, there are no unproven conjectures required to prove the hardness of learning PARITIES in the statistical query framework. The reason for this is that the proof is purely information-theoretic. In particular, even an algorithm that uses unbounded computation and uses query functions χ that are not evaluatable in polynomial time (or indeed even uncomputable ones!), requires a superpolynomial number of queries to the $\mathsf{STAT}(c, D)$ oracle.

Let us now sketch a proof of the result. To make our notation simpler, we will assume that the instance space is $\{-1, 1\}^n$, rather than $\{0, 1\}^n$. We will also assume that the output of boolean functions is in the set $\{-1, 1\}$. Then for a subset $S \subseteq [n]$, the parity function on bits in S, is defined as:

$$f_S(\mathbf{x}) = \prod_{i \in S} x_i$$

If we interpret a bit $x_i = -1$ as being on and $x_i = +1$ as being off, then $f_S(\mathbf{x}) = -1$ if and only if an odd number of bits in S are on, i.e. f_S computes the parity over bits in S. We will consider $\mathcal{U}$, the uniform distribution over $\{-1,1\}^n$, as the target distribution. We will assume that the distribution is known to the algorithm, and as a result, without loss of generality, we may assume that the algorithm only makes correlational queries to the oracle $\mathsf{STAT}(c, D)$.

Let us observe some basic facts about PARITIES with respect to the uniform distribution $\mathcal{U}$. For any non-empty set S, $\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}}\left[f_S(\mathbf{x})\right] = 0$. This is because the uniform distribution $\mathcal{U}$ over $\{-1,1\}^n$ can be viewed as a product distribution over the individual bits, i.e. all bits are independent. The distribution over each bit is also uniform, so that $\mathbb{E}\left[x_i\right] = 0$ for each i. Then, we also have:

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}}\left[f_S(\mathbf{x})f_T(\mathbf{x})\right] &= \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}}\left[\prod_{i\in S} x_i \prod_{i\in T} x_i\right] \\ &= \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}}\left[\prod_{i\in S\Delta T} x_i\right].\end{aligned}$$

Above, $S\Delta T$ denotes the symmetric difference of the sets S and T. This establishes that if $S \neq T$, then $\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}}\left[f_S(\mathbf{x})f_T(\mathbf{x})\right] = 0$. Clearly, it is also the case that $\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}}\left[(f_S(\mathbf{x}))^2\right] = 1$, since $f_S(\mathbf{x}) \in \{-1,1\}$. Since there are 2^n such parity functions and there are exactly 2^n points in $\{-1,1\}^n$, the set of parity functions form an orthonormal basis for the vector space of real-valued functions defined over $\{-1,1\}^n$, with inner product $\langle\varphi,\psi\rangle = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}}\left[\varphi(\mathbf{x})\psi(\mathbf{x})\right]$. Formally, any $\varphi : \{-1,1\}^n \to \mathbb{R}$ can be expressed as:

$$\varphi(\mathbf{x}) = \sum_{S\subseteq[n]} \widehat{\varphi}(S) f_S(\mathbf{x})$$

Furthermore, Parseval's identity establishes that $\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}}\left[(\varphi(\mathbf{x}))^2\right] = \sum_{S\subseteq[n]}(\widehat{\varphi}(S))^2$. In particular, if $\varphi : \{-1,1\}^n \to \{-1,1\}$, then $\sum_{S\subseteq[n]}(\widehat{\varphi}(S))^2 = 1$; the coefficient $\widehat{\varphi}(S) = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{U}}\left[\varphi(\mathbf{x})f_S(\mathbf{x})\right]$.

Suppose that there is an algorithm, A, that learns PARITIES using at most q queries, each of which has a tolerance parameter larger than τ_0. We show that it must be the case that $q/\tau_0^2 > 2^n - 2$. For the sake of contradiction, suppose that this is not the case. We show that in fact there must be a target parity on which the algorithm fails. The definition of SQ learning requires that a *single* algorithm work for all target functions. We will show that this cannot be the case. We run the algorithm A and every time a query is made, we return the answer 0. Let h be the target hypothesis when A terminates. We will show that when $q/\tau_0^2 \leq 2^n - 2$, there must be at least two parity functions, represented by subsets S and S', such that for all the queries made by the algorithm 0 was a valid answer. Let's first see that this finishes the proof. Clearly, both f_S and $f_{S'}$ could be a valid target; thus it must be the case that $\mathrm{err}(h; f_S, \mathcal{U}) \leq \frac{1}{10}$ and $\mathrm{err}(h; f_{S'}, \mathcal{U}) \leq \frac{1}{10}$. However, $\mathbb{P}_{\mathbf{x}\sim\mathcal{U}}\left[f_S(\mathbf{x}) \neq f_{S'}(\mathbf{x})\right] = \frac{1}{2}$; thus h cannot be a $\frac{1}{10}$ accurate hypothesis for both f_S and $f_{S'}$.

Let $\varphi_1, \ldots, \varphi_q$ denote the queries made by the algorithm. There is a subtle point to be mentioned here; in principle the queries made by the algorithm

could be adaptive, i.e. the query depends on past answers from the oracle $\mathsf{STAT}(c, D)$. However, since we always supply 0 as the answer, we may as well assume that the queries $\varphi_1, \ldots, \varphi_q$ are known ahead of time. We claim that for any $\varphi : \{-1, 1\} \to \{-1, 1\}$, there are at most $\frac{1}{\tau_0^2}$ parity functions, such $\left| \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{U}} \left[\varphi(\mathbf{x}) f_S(\mathbf{x}) \right] \right| \geq \tau_0$. This follows from that fact that $\sum_{S \subseteq [n]} (\widehat{\varphi}(S))^2 = 1$ and that $\widehat{\varphi}(S) = \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{U}} \left[\varphi(\mathbf{x}) f_S(\mathbf{x}) \right]$. Thus, a single query rules out at most $\frac{1}{\tau^2}$ parities as the possible target, if we supply a response of 0. Consequently, q queries rule out at most q/τ_0^2 parities as the possible target. Provided, $q/\tau_0^2 \leq 2^n - 2$, there must be at least two different parity functions that are consistent with all the answers supplied to the algorithm, as a simulation of the oracle $\mathsf{STAT}(c, D)$. This completes the proof.

There is another subtle observation worth making. The above proof sketch can be made completely rigorous for deterministic algorithms. However, one may wonder whether randomised algorithms may help avoid the difficulty of learning PARITIES. The answer to this is negative, i.e. randomised algorithms do not help in this case. This can be shown formally by picking a target parity function uniformly at random and computing the probability that the simulation becomes invalid after each query response. The proof is not difficult, but the details are somewhat technical, and it is left as an exercise to the interested reader to fill those in.

7.4 Exercises

7.1 We will consider an extension of the statistical query model, where in addition to making queries of the form (χ, τ) to the oracle $\mathsf{STAT}(c, D)$, the learning algorithm is allowed access to *unlabelled* examples from D, i.e. it may get points $\mathbf{x} \in X$ drawn according to D, but not the labels $c(\mathbf{x})$.

a) Briefly argue why any concept that is (efficiently) learnable with access to $\mathsf{STAT}(c, D)$ and unlabelled examples, is also (efficiently) learnable with access to the noisy example oracle, $\mathsf{EX}^{\eta}(c, D)$.

b) Give an efficient algorithm for learning axis-aligned rectangles in the plane using $\mathsf{STAT}(c, D)$ and unlabelled examples.

7.5 Bibliographic Notes

The variant of PAC learning with random classification noise was introduced in the work of Angluin and Laird [5]. Our presentation of the statistical query model differs a bit from that in the textbook by Kearns and Vazirani [34, Chap. 5]. Bshouty and Feldman [16] first used the idea of separating a general statistical query into target-independent and correlational queries. The proof that PARITIES cannot be learnt in the statistical query model appeared in the work of Kearns [31]. The proof provided in the lecture follows along the lines of that introduced by Blum et al. [11] who also show lower bounds for general concept classes in terms of what is called the *statistical query dimension*.

Of course, the fact that PARITIES is not SQ-learnable does not rule out an efficient algorithm for learning PARITIES in the presence of random classification

noise. Blum et al. [12] provide an algorithm whose complexity (both statistical and computational) is $2^{O(n/\log n)}$. Improving this dependence in terms of computational complexity is a long standing open problem. It is widely believed that learning parities with noise is hard and cryptographic systems based on the hardness of this and related problems have been designed. Blum et al. [12] also demonstrate the existence of a concept class that is not polynomial-time learnable in the SQ model, but can be learnt in polynomial time in the PAC model with random classification noise provided the noise rate is a constant bounded away from $1/2$ (their algorithm does not have polynomial dependence on $1/(1-2\eta)$).

Chapter 8

Learning Real-valued Functions

So far our focus has been on learning boolean functions. Boolean functions are suitable for modelling binary classification problems; in fact, even multi-class classification can be viewed as a sequence of binary classification problems. Many commonly used approaches for multi-class classification, such as one-vs-rest or one-vs-one, solve several binary classification problems as a means to perform multi-class classification. However, sometimes we may need to learn functions whose output is real-valued (or vector-valued). In this chapter, we will study linear models and generalised linear models. In order to give bounds on the generalisation error, we'll need to introduce some new concepts that play a role analogous to the VC dimension. We will also study some basic convex optimisation techniques.

8.1 Learning Real-Valued Functions

Let us start with the general question of learning real-valued functions. Suppose our instance space is $X_n = \mathbb{R}^n$, though we can easily generalise to other forms of instance spaces. Let $\mathcal{F}$ denote a class of real-valued functions from $\mathbb{R}^n \to \mathbb{R}$. Let $f^\star \in \mathcal{F}$, where $f^\star : \mathbb{R}^n \to \mathbb{R}$ is the target function.

At this point, it is worth considering models for data. We will let D denote some joint distribution over $X_n \times \mathbb{R}$. So all observations sampled from D are of the form $(\mathbf{x}, y)$, where $\mathbf{x} \in \mathbb{R}^n$, and y gives us some information about the target function. It will be convenient to also denote the marginal distribution of D over X_n by μ. If we generalise the notion of PAC learning directly, we will restrict the distribution D to have support $\{(\mathbf{x}, f^\star(\mathbf{x})) \mid \mathbf{x} \in \mathbb{R}^n\}$, whenever $f^\star$ is the target function. Thus, an example drawn from D would be the same as an example obtained from the *example oracle*, $\mathsf{EX}(\mu, f^\star)$. When learning real-valued functions, it is completely unrealistic to assume that we would observe the target value, $f^\star(\mathbf{x})$, exactly.

Before we elaborate further on the data model, we need to discuss the measure of *performance*. When considering boolean functions, the indicator, $\mathbb{1}(h(\mathbf{x}) \neq c(\mathbf{x}))$, is in some sense the *only* meaningful measure of error at a given point $\mathbf{x}$. When considering real-valued functions, how far our hypothesis' prediction $h(\mathbf{x})$ is from $f^\star(\mathbf{x})$ may matter. We will define this in terms of *loss functions*. We denote the *loss* function by $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$; here $\mathbb{R}_{\geq 0}$ denotes

the non-negative real numbers.[1] For the most part, we will concern ourselves in the setting where $\ell(y', y) = (y' - y)^2$. Another notion that is important is that of the *risk functional*, which depends on the loss function ℓ and the distribution D over $X_n \times \mathbb{R}$. For some function $h : X_n \to \mathbb{R}$, define the risk of h, denoted by $R(h) = R(h; \ell, D)$ (we will use the shorthand $R(h)$ when ℓ and D are clearly from the context), as,

$$R(h) = R(h; \ell, D) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D} \left[\ell(h(\mathbf{x}), y\right] .$$

It is worth noting that without some restrictions on the function h and D, there is no reason to expect that the above expectation exists at all. Throughout this chapter, we will assume that we are in the setting where expectations (and when needed higher moments) exist. We will not elaborate particularly on this point further. Let us now return to our discussion of data models.

8.1.1 Realisable Setting

In this section, we will restrict ourselves to the case when ℓ is the squared loss. Suppose that $\mathcal{F}$ is some class of functions. In the *realisable* setting, any data generating distribution, D, must satisfy the condition that there exists some $f^\star \in \mathcal{F}$, such that $\mathbb{E}[y|\mathbf{x}] = f^\star(\mathbf{x})$ for every $\mathbf{x} \in \mathbb{R}^n$.[2] We can view this setting as a generalisation of the random classification noise setting. Let $f^\star \in \mathcal{F}$ be the target function. For $(\mathbf{x}, y) \sim D$, denote $\xi(\mathbf{x}, y) = y - f^\star(\mathbf{x})$. Then $\xi(\mathbf{x}, y)$ is a random variable that is *independent* of $\mathbf{x}$, though its law may be a function of $\mathbf{x}$, and it satisfies $\mathbb{E}\left[\xi(\mathbf{x}, y)\right] = 0$. Alternatively, we can view the data generating process as follows:

- Draw $\mathbf{x} \sim \mu$.
- Let $\xi \sim \nu_{\mathbf{x}}$, where ξ is a real-valued random variable distributed according to some law $\nu_{\mathbf{x}}$ and satisfies $\mathbb{E}[\xi] = 0$.
- Return $(\mathbf{x}, \mathbf{x} + \xi)$.

Thus, in this context, *realisability* refers to the fact that the noise in the observations is *zero-mean*. What would be a suitable goal for a learning algorithm in this context? For some function $h : X_n \to \mathbb{R}$, we define the following quantity, denoted by $\varepsilon(h) = \varepsilon(h; f^\star, \mu)$ (again we will omit $f^\star$ and μ when clear from context), as,

$$\varepsilon(h) = \varepsilon(h; f^\star, \mu) = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mu} \left[\ell(h(\mathbf{x}), f^\star(\mathbf{x}))\right] = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mu} \left[(h(\mathbf{x}) - f^\star(\mathbf{x}))^2\right] . \tag{8.1}$$

[1] The non-negativity constraint on the output of ℓ is not strictly necessary. However, when using loss functions that can potentially be negative, we need to be more careful to ensure that we are not achieving trivial results. We will indeed use a loss function that may take negative values in Section 8.5.

[2] Again, we will avoid getting drawn into technicalities about what happens when the conditional expectation is not well-defined on some subset of X_n. We shall assume that all expectations we use are well-defined, though it is possible to remove some of these restrictions.

We can establish a relationship between $R(h)$, $R(f^\star)$ and $\varepsilon(h)$, as follows:

$$\begin{aligned}
R(h) &= \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D}\left[(h(\mathbf{x})-y)^2\right] \\
&= \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D}\left[(h(\mathbf{x})-f^\star(\mathbf{x})+f^\star(\mathbf{x})-y)^2\right] \\
&= \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D}\left[(h(\mathbf{x})-f^\star(\mathbf{x}))^2\right] + \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D}\left[(f^\star(\mathbf{x})-y)^2\right] \\
&\quad - 2\cdot \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D}\left[(h(\mathbf{x})-f^\star(\mathbf{x}))(f^\star(\mathbf{x})-y)\right] \\
&= \varepsilon(h) + R(f^\star) - 2\cdot \mathop{\mathbb{E}}_{\mathbf{x}\sim\mu}\left[(h(\mathbf{x})-f^\star(\mathbf{x}))\cdot \mathop{\mathbb{E}}_{\mathbf{y}}\left[f^\star(\mathbf{x})-y|\mathbf{x}\right]\right].
\end{aligned}$$

Since $\mathbb{E}[y|\mathbf{x}] = f^\star(\mathbf{x})$, the last term above is 0, and we get,

$$R(h) = R(f^\star) + \varepsilon(h). \tag{8.2}$$

Note that $\varepsilon(h) \geq 0$ for all h. Thus, Eq. (8.2) establishes that $f^\star$ is indeed the minimiser of the *risk*. Although, it may seem intuitively that this should be the case, it is something that requires proof. In this case, it holds because we are using the squared loss, and fixing $\mathbf{x}$, the value v that minimises $\mathbb{E}[(y-v)^2|\mathbf{x}]$ is exactly given by $v = \mathbb{E}[y|\mathbf{x}] = f^\star(\mathbf{x})$. Thus, the *zero mean* assumption for *realisability* is tied to using *squared loss* as the notion of error.[3]

Equation (8.2) also is important in the sense that it suggests an algorithmic approach. As our algorithm can only have access to a sample S drawn from D, it is not clear what the empirical counterpart of $\epsilon(h)$ would be. However, the empirical counterpart of $R(h)$ can easily be defined, indeed it is a fundamental notion called *empirical risk*. Let $S = \big((\mathbf{x}_i, y_i)\big)_{i=1}^m \sim D^m$ be a sample drawn from D. For some function $h : X_n \to \mathbb{R}$, define the empirical risk of h, denoted by $\widehat{R}_S(h)$, as,

$$\widehat{R}_S(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i). \tag{8.3}$$

Provided we can find a way to relate the empirical risk to the population risk, this suggests an algorithmic paradigm where we try to find a function h that minimises the empirical risk. This principle is known as the *empirical risk minimisation* (ERM) principle, and is one of the fundamental concepts in statistical learning theory.

8.1.2 Non-Realisable Setting

In a way, we don't need to restrict the data distribution, D, as we did in the previous section. Let $\mathcal{F}$ be some class of functions. Consider the quantity defined below,

$$R(\mathcal{F}; D) := \inf_{f\in\mathcal{F}} R(f; D).$$

[3] Indeed, if we were using the absolute loss, we would want to pick the conditional median and so on.

Thus $R(\mathcal{F}; D)$ represents how good the class $\mathcal{F}$ may be to explain the data (though it may be scale-dependent, so the absolute value of the above term may not mean very much). We can still however use any such class $\mathcal{F}$ as a comparator class. Thus, we will phrase our learning goal as finding a hypothesis function $h : X_n \to \mathbb{R}$, such that,

$$R(h; D) \leq \inf_{f \in \mathcal{F}} R(f; D) + \epsilon. \tag{8.4}$$

Clearly, since $\mathcal{F}_1 \subseteq \mathcal{F}_2$ implies that $R(\mathcal{F}_2; D) \leq R(\mathcal{F}_1; D)$, the larger the class $\mathcal{F}$ for which we can provide a guarantee of the form (8.4) the better the result. The flip side of this is that the larger the class $\mathcal{F}$ we use, the larger our sample size, $|S|$, may need to be, to ensure that the empirical risk and the true risk are close to each other. This is the common *underfitting* vs *overfitting* tradeoff in machine learning.

Although we have not studied such a setting, where we make no assumptions on the joint distribution over the instances and their labels, in the case of boolean functions, one can ask a similar question there. This is commonly known as the *agnostic setting* in that context.

8.1.3 A General Algorithmic Paradigm

We can now develop a general algorithmic paradigm for learning real-valued functions (or indeed even more broadly than that). Let $\mathcal{F}$ be a class of functions from $X_n \to \mathbb{R}$ that will serve as our comparator class. Let $\mathcal{G}$ be a class of functions from $X_n \to \mathbb{R}$ such that $\mathcal{G} \supseteq \mathcal{F}$. Let $S \sim D^m$ be a sample such that the following holds for all $g \in G$,

$$|\widehat{R}_S(g) - R(g)| \leq \frac{\epsilon}{3}. \tag{8.5}$$

Furthermore suppose we can find some $\widehat{g} \in \mathcal{G}$, such that,

$$\widehat{R}_S(\widehat{g}) \leq \inf_{g \in \mathcal{G}} \widehat{R}_S(g) + \frac{\epsilon}{3}. \tag{8.6}$$

Then for any $f^\star \in \mathcal{F}$, we have,

$$R(\widehat{g}) - R(f^\star) = R(\widehat{g}) - \widehat{R}_S(\widehat{g}) + \widehat{R}_S(\widehat{g}) - \widehat{R}_S(f^\star) + \widehat{R}_S(f^\star) - R(f^\star)$$

Since, $f^\star \in \mathcal{F} \subseteq \mathcal{G}$, using Eqs. (8.5) and (8.6), we have,

$$R(\widehat{g}) - R(f^\star) \leq \epsilon.$$

Since $f^\star \in \mathcal{F}$ was chosen to be arbitrary, we have,

$$R(\widehat{g}) \leq \inf_{f \in \mathcal{F}} R(f) + \epsilon.$$

Thus, this gives us an algorithm in the non-realizable setting. It also works in the *realizable* setting, as we then know that $f^\star(\mathbf{x}) = \mathbb{E}\left[y|\mathbf{x}\right]$ is the minimiser of $R(f)$ and also that $\varepsilon(\widehat{g}) = R(\widehat{g}) - R(f^\star)$.

There are two main outstanding issues. The first is to understand when a guarantee as strong as Eq. (8.5) may hold for a class of functions $\mathcal{G}$. Intuitively,

the "*smaller*" $\mathcal{G}$ is, the better it is from the point of view or relating the empirical risk to the true risk. This question will be discussed in detail in Section 8.3. The second is to design an algorithm to find $\widehat{g}$ that satisfies Eq. (8.6). We will resort to generic convex optimisation techniques for this problem. We discuss convex optimisation briefly in Section 8.2. Intuitively, the "*larger*" $\mathcal{G}$ is, the easier the optimisation problem. Thus, there is a tradeoff between the *statistical* complexity of this problem and the *computational* one, not unlike the one we've seen previously in the context of learning boolean functions. In order to obtain good statistical and optimisation guarantees, we will need some restrictions on the range of functions in $\mathcal{G}$ as well as on the support of the distribution D. Together with some concrete applications, these will be discussed in Sections 8.4 and 8.5.

8.2 Projected Gradient Descent for Lipschitz functions

This section can be read independently of the rest of the chapter (or indeed the lecture notes) and concerns with one specific convex optimisation technique and its analysis. We will briefly describe an algorithm for minimising Lipschitz convex functions over closed and bounded convex sets. Our treatment of convex optimisation is at best cursory and for further details the student may refer to any of the following references [17, 15, 38].

Consider a function $f : \mathbb{R}^n \to \mathbb{R}$; f is convex if $f(\lambda \mathbf{x} + (1-\lambda)\mathbf{x}') \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{x}')$ for all $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$ and for all $\lambda \in [0,1]$. Let $K \subseteq \mathbb{R}^n$ be a closed, bounded, convex set. We are interested in solving the following constrained optimisation problem: minimise $f(\mathbf{x})$ subject to $\mathbf{x} \in K$. Furthermore, we will assume that f is differentiable over K and that there exists some L, such that $\left\|\nabla f(\mathbf{x})\right\|_2 \leq L$ for all $\mathbf{x} \in K$. Also let $B = \max_{\mathbf{x},\mathbf{x}' \in K} \left\|\mathbf{x} - \mathbf{x}'\right\|_2$ denote the *diameter* of K.[4]

We will see a proof that the *average iterate* of Projected Gradient Descent (Algorithm 8.1) is an approximate minimiser of f over K. The algorithm requires a projection operation: for some point $\mathbf{x} \in \mathbb{R}^n$, this operation gives the point in K that is closest to $\mathbf{x}$. More formally, define the projection operator,

$$\Pi_K(\mathbf{x}) := \operatorname*{argmin}_{\mathbf{x}' \in K} \left\|\mathbf{x} - \mathbf{x}'\right\|_2 .$$

Note that such point exists as the set K is closed, and is unique because of the convexity of K.

Algorithmically, one may wonder how to perform the projection operation. In general, projection is itself a convex optimisation problem. There are other convex optimisation algorithms that could be used to solve the projection problem. However, for many applications in machine learning, typically projection is only required onto fairly simple convex sets, such as ℓ_1- or ℓ_2-balls. In these cases, this operation is very easy to perform.

We will first prove a useful property about the projection operator.

[4] Convergence of variants of gradient descent, at different rates, can be established in much more general convex optimisation settings. The settings described above are sufficient for the purpose of the applications in this chapter; however, an interested reader should refer to the books [17, 15, 38] and beyond.

Lemma 8.1. *For any $\mathbf{z} \in K$, if $\mathbf{x}' \in \mathbb{R}^n$ and $\mathbf{x} = \Pi_K(\mathbf{x}')$, then,*

$$\|\mathbf{z} - \mathbf{x}\|_2 \leq \left\|\mathbf{z} - \mathbf{x}'\right\|_2.$$

Proof. Consider the following:

$$\begin{aligned}\left\|\mathbf{z} - \mathbf{x}'\right\|_2^2 &= \left\|\mathbf{z} - \mathbf{x} + \mathbf{x} - \mathbf{x}'\right\|_2^2 \\ &= \|\mathbf{z} - \mathbf{x}\|_2^2 + \left\|\mathbf{x} - \mathbf{x}'\right\|_2^2 - 2(\mathbf{z} - \mathbf{x}) \cdot (\mathbf{x}' - \mathbf{x})\end{aligned}$$

Now if, $(\mathbf{z}-\mathbf{x}) \cdot (\mathbf{x}' - \mathbf{x}) \leq 0$ we are done. Suppose for the sake of contradiction that $(\mathbf{z} - \mathbf{x}) \cdot (\mathbf{x}' - \mathbf{x}) > 0$. We will establish that $\mathbf{x}$ cannot be the projection of $\mathbf{x}'$ onto K. Consider the following:

$$\left\|\lambda\mathbf{z} + (1-\lambda)\mathbf{x} - \mathbf{x}'\right\|_2^2 = \left\|\mathbf{x} - \mathbf{x}'\right\|_2^2 + \lambda^2 \|\mathbf{z} - \mathbf{x}\|_2^2 - 2\lambda(\mathbf{z} - \mathbf{x}) \cdot (\mathbf{x}' - \mathbf{x})$$

This implies that for $\lambda \in \left(0, \min\left\{1, \frac{(\mathbf{z}-\mathbf{x})\cdot(\mathbf{x}'-\mathbf{x})}{\|\mathbf{z}-\mathbf{x}\|_2^2}\right\}\right)$,

$$\left\|\lambda\mathbf{z} + (1-\lambda)\mathbf{x} - \mathbf{x}'\right\|_2 < \left\|\mathbf{x} - \mathbf{x}'\right\|_2.$$

As K is a convex set and $\mathbf{x}, \mathbf{z} \in K$, $\lambda\mathbf{z} + (1-\lambda)\mathbf{x} \in K$, contradicting the claim that $\Pi_K(\mathbf{x}') = \mathbf{x}$. Thus, it must be the case that $\left\|\mathbf{z} - \mathbf{x}'\right\|_2 \geq \|\mathbf{z} - \mathbf{x}\|_2$. □

We can now proceed to analyse the convergence of Iterative Projected Gradient Descent which is presented as Alg. 8.1 below.

Algorithm 8.1: Projected Gradient Descent

Input: η, T
Pick $\mathbf{x}_1 \in K$
for $t = 1, \ldots, T$ **do**
    $\mathbf{x}'_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$
    $\mathbf{x}_{t+1} = \Pi_K(\mathbf{x}'_{t+1})$
Output: $\frac{1}{T}\sum_{t=1}^{T} \mathbf{x}_t$

Theorem 8.2. *Suppose $K \subseteq \mathbb{R}^n$ is a convex, closed and bounded, $\max_{\mathbf{x},\mathbf{x}' \in K} \left\|\mathbf{x} - \mathbf{x}'\right\|_2 \leq B$, and that $\sup_{\mathbf{x} \in K} \left\|\nabla f(\mathbf{x})\right\|_2 \leq L$. Then Alg. 8.1 run with $\eta = \frac{B}{L\sqrt{T}}$, outputs $\bar{\mathbf{x}}$, such that,*

$$f(\bar{\mathbf{x}}) \leq \min_{\mathbf{x} \in K} f(\mathbf{x}) + \frac{RL}{\sqrt{T}}.$$

Proof. Let $\mathbf{x}_t$ denote the point at the t^{th} iteration of the gradient descent procedure. Let $\mathbf{x}^\star \in K$ be any point. Then by the convexity of f, we have the following:

$$\begin{aligned}f(\mathbf{x}_t) - f(\mathbf{x}^\star) &\leq \nabla f(\mathbf{x}_t) \cdot (\mathbf{x}_t - \mathbf{x}^\star) \\ &= \frac{1}{\eta}(\mathbf{x}_t - \mathbf{x}'_{t+1}) \cdot (\mathbf{x}_t - \mathbf{x}^\star) \\ &= \frac{1}{2\eta}\left(\|\mathbf{x}_t - \mathbf{x}^\star\|_2^2 + \left\|\mathbf{x}_t - \mathbf{x}'_{t+1}\right\|_2^2 - \left\|\mathbf{x}'_{t+1} - \mathbf{x}_*\right\|_2^2\right) \\ &= \frac{1}{2\eta}\left(\|\mathbf{x}_t - \mathbf{x}^\star\|_2^2 - \left\|\mathbf{x}'_{t+1} - \mathbf{x}^\star\right\|_2^2\right) + \frac{\eta}{2}\left\|\nabla f(\mathbf{x}_t)\right\|_2^2\end{aligned}$$

We use the bound $\left\|\nabla f(\mathbf{x}_t)\right\|_2 \leq L$ and the fact that $\left\|\mathbf{x}'_{t+1} - \mathbf{x}^\star\right\|_2 \geq \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|_2$, to obtain,

$$f(\mathbf{x}_t) - f(\mathbf{x}^\star) \leq \frac{1}{2\eta}\left(\|\mathbf{x}_t - \mathbf{x}^\star\|_2^2 - \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|_2^2\right) + \frac{\eta L^2}{2}. \tag{8.7}$$

By convexity of f, it follows that $f\left(\frac{1}{T}\sum_{t=1}^T \mathbf{x}_t\right) \leq \frac{1}{T}\sum_{t=1}^T f(\mathbf{x}_t)$. We can average Eq. (8.7) over $t = 1, \ldots, T$ to obtain,

$$\begin{aligned} f\left(\frac{1}{T}\sum_{t=1}^T \mathbf{x}_t\right) - f(\mathbf{x}^\star) &\leq \frac{1}{T}\sum_{t=1}^T (f(\mathbf{x}_t) - f(\mathbf{x}^\star)) \\ &\leq \frac{1}{2T\eta}\left(\|\mathbf{x}_1 - \mathbf{x}^\star\|_2^2 - \|\mathbf{x}_{T+1} - \mathbf{x}^\star\|_2^2\right) + \frac{\eta}{2}L^2 \\ &\leq \frac{\|\mathbf{x}_1 - \mathbf{x}^\star\|_2^2}{2T\eta} + \frac{\eta L^2}{2} \\ &\leq \frac{B^2}{2T\eta} + \frac{\eta L^2}{2} \end{aligned}$$

Setting $\eta = \frac{B}{L\sqrt{T}}$ and choosing $\mathbf{x}^\star$ to be the minimiser of f in K completes the proof. □

8.3 Rademacher Complexity

Let us now address the question of bounding the generalisation error. Let $\mathcal{G}$ be a class of functions from some space X to the bounded interval $[a, b]$.[5] Note that typically, we will not be thinking of $\mathcal{G}$ as being the class of target functions (or hypothesis functions), but functions obtained by composing the *loss* function with the class of target functions. Let D be some distribution over X and let $S = (\mathbf{z}_1, \ldots, \mathbf{z}_m) \sim D^m$ be an i.i.d. sample drawn from D. Let us define the following random variable, $\Phi(S)$, as,

$$\Phi(S) = \Phi(\mathbf{z}_1, \ldots, \mathbf{z}_m) = \sup_{g \in \mathcal{G}} \left\{ \mathop{\mathbb{E}}_{\mathbf{z} \sim D}\left[g(\mathbf{z})\right] - \frac{1}{m}\sum_{i=1}^m g(\mathbf{z}_i) \right\}.$$

Before we proceed to understand the above object, it is worth thinking about why we want to bound a random variable defined in terms of the supremum over $\mathcal{G}$. The output $\widehat{h}$ of a learning algorithm will be a function of the data S, thus $\widehat{h}$ (which is a random variable) is not independent of S. Thus, we can't use standard results to relate the empirical risk $\widehat{R}_S(\widehat{h})$ to the true risk $R(\widehat{h})$. The class $\mathcal{G}$ we will consider will be functions composing the loss ℓ with functions in the range of the algorithm. If we can show that $\Phi(S)$ is bounded, both in expectation and in probability, then we can relate $\widehat{R}_S(\widehat{h})$ to $R(\widehat{h})$ even without stochastic independence between $\widehat{h}$ and S.

[5] While some results can be generalised to the case when the range of functions is not bounded, for simplicity, we will only consider bounded functions.

8.3.1 (Empirical) Rademacher Complexity

We now define a concept called the *Empirical Rademacher Complexity* for a family of functions.

Definition 8.3 – **Empirical Rademacher Complexity.** *Let $\mathcal{G}$ be a family of functions mapping some space $X \to [a,b]$ and let $S = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_m) \in X^m$ be a fixed sample of size m. Then the* empirical Rademacher complexity *of $\mathcal{G}$ with respect to the sample S is defined as,*

$$\widehat{\mathsf{RAD}}_S(G) = \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i g(\mathbf{z}_i)\right],$$

where $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ are i.i.d. Rademacher random variables, i.e. σ_i takes value in $\{-1, 1\}$ uniformly at random.

The formal reason for defining the Empirical Rademacher Complexity in this way will become clear when we consider the proof of Lemma 8.5. It is worth getting some intuitive understanding of this concept as well. The empirical Rademacher complexity measures how well functions from a class correlate with random noise, where the notion of noise refers to Rademacher random variables. This corresponds to our notion that the more complex the class $\mathcal{G}$, the more easily it can fit noise. In particular, let us suppose that $\mathcal{G}$ is a class of boolean functions (with range $\{-1, 1\}$) with $\mathsf{VCD}(\mathcal{G}) \geq m$ and let S be a set that is shattered by $\mathcal{G}$, then $\widehat{\mathsf{RAD}}_S(G) = 1$. However, Rademacher complexity can be defined for any class of real-valued functions.[6]

Let us now define Rademacher complexity, which is defined as the expected empirical Rademacher complexity of sequences of length m drawn from a distribution D over X.

Definition 8.4 – **Rademacher Complexity.** *Let D be a distribution over X and let $\mathcal{G}$ be a family of functions mapping $X \to [a,b]$. For any integer, $m \geq 1$, the Rademacher complexity of $\mathcal{G}$, is the expectation of the empirical Rademacher complexity of $\mathcal{G}$ over samples of size m drawn independently from D, i.e.*

$$\mathsf{RAD}_m(\mathcal{G}) = \mathop{\mathbb{E}}_{S\sim D^m}\left[\widehat{\mathsf{RAD}}_S(\mathcal{G})\right].$$

To be more precise, we should write $\mathsf{RAD}_m(\mathcal{G}; D)$ as the Rademacher Complexity depends on the distribution. When the distribution is clear from context, we will ignore this to simplify notation.

8.3.2 Uniform Convergence Results

Our main results in this section, concern understanding the behaviour of $\Phi(S)$ in expectation and probability. It will be helpful to introduce some shorthand notation. We will use $\mathop{\mathbb{E}}_{D}[g]$ to mean $\mathop{\mathbb{E}}_{\mathbf{z}\sim D}\left[g(\mathbf{z})\right]$ and we will use,

$$\widehat{\mathbb{E}}_S[g] = \frac{1}{m}\sum_{i=1}^{m} g(\mathbf{z}_i),$$

[6] We will ignore issues of measurability; this will not be a matter of concern for the function classes we study in this course.

where $S = (\mathbf{z}_1, \ldots, \mathbf{z}_m)$ is a sample.

We will now prove the following result.

***Lemma 8.5* – Symmetrization.** *Let $\mathcal{G}$ be a class of functions from X to $[a, b]$ and let D be a distribution over X. Let $S = (\mathbf{z}_1, \ldots, \mathbf{z}_m) \sim D^m$, then for,*

$$\Phi(S) = \Phi(\mathbf{z}_1, \ldots, \mathbf{z}_m) = \sup_{g \in \mathcal{G}} \left\{ \mathop{\mathbb{E}}_{\mathbf{z} \sim D} \big[g(\mathbf{z}) \big] - \frac{1}{m} \sum_{i=1}^{m} g(\mathbf{z}_i) \right\},$$

we have,

$$\mathop{\mathbb{E}}_{S} \big[\Phi(S) \big] \leq 2\mathsf{RAD}_m(\mathcal{G}).$$

Proof. We have

$$\mathop{\mathbb{E}}_{S} \big[\Phi(S) \big] = \mathop{\mathbb{E}}_{S} \left[\sup_{g \in \mathcal{G}} \left\{ \mathop{\mathbb{E}}_{D} [g] - \widehat{\mathbb{E}}_S[g] \right\} \right].$$

We can rewrite $\mathop{\mathbb{E}}_{D}[g] = \mathop{\mathbb{E}}_{S'} \left[\widehat{\mathbb{E}}_{S'}[g] \right]$ where $S' \sim D^m$ is independent of S. Thence,

$$\begin{aligned}
\mathop{\mathbb{E}}_{S} \big[\Phi(S) \big] &= \mathop{\mathbb{E}}_{S} \left[\sup_{g \in \mathcal{G}} \left\{ \mathop{\mathbb{E}}_{S'} \left[\widehat{\mathbb{E}}_{S'}[g] \right] - \widehat{\mathbb{E}}_S[g] \right\} \right] \\
&\leq \mathop{\mathbb{E}}_{S,S'} \left[\sup_{g \in \mathcal{G}} \left\{ \widehat{\mathbb{E}}_{S'}[g] - \widehat{\mathbb{E}}_S[g] \right\} \right] \qquad (8.8) \\
&= \mathop{\mathbb{E}}_{S,S'} \left[\sup_{g \in \mathcal{G}} \left\{ \frac{1}{m} \sum_{i=1}^{m} \big(g(\mathbf{z}'_i) - g(\mathbf{z}_i) \big) \right\} \right].
\end{aligned}$$

In (8.8), we obtain the inequality by pushing the sup inside the expectation. We can now use the fact that S and S' are independent and identically redistributed to rewrite the RHS above. In particular, we can thinking of obtaining

$$(\mathbf{z}_1, \ldots, \mathbf{z}_m, \mathbf{z}'_1, \ldots, \mathbf{z}'_m) \sim D^{2m},$$

but then deciding to put either $\mathbf{z}_i$ or $\mathbf{z}'_i$ in S and the other in S' uniformly at random. Clearly this doesn't change the expectation. So we can introduce the Rademacher random variables $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)$ to indicate this choices. We have,

$$\begin{aligned}
\mathop{\mathbb{E}}_{S} \big[\Phi(S) \big] &\leq \mathop{\mathbb{E}}_{S,S',\boldsymbol{\sigma}} \left[\sup_{g \in \mathcal{G}} \left\{ \frac{1}{m} \sum_{i=1}^{m} \sigma_i (g(\mathbf{z}'_i) - g(\mathbf{z}_i)) \right\} \right] \\
&\leq \mathop{\mathbb{E}}_{S',\boldsymbol{\sigma}} \left[\sum_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i g(\mathbf{z}'_i) \right] + \mathop{\mathbb{E}}_{S,\boldsymbol{\sigma}} \left[\sum_{g \in \mathcal{G}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i g(\mathbf{z}_i) \right] \\
&= 2\mathsf{RAD}_m(\mathcal{G}).
\end{aligned}$$

Above, we used the fact that $\boldsymbol{\sigma}$ and $-\boldsymbol{\sigma}$ are identically distributed, that S and S' are also identically distributed, and the definition of Rademacher complexity. □

We can see how the symmetrisation using Rademacher random variables in the above proof leads to terms that motivate the definition of Rademacher complexity. Thus, the formal reason for defining the Rademacher complexity the way we have is that it gives us an upper bound on $\mathop{\mathbb{E}}_{S}\left[\Phi(S)\right]$.

In order to bound $\Phi(S)$ in probability, we will use McDiarmid's inequality, which we state below without proof. A proof can be found in in [37, Theorem 13.7].

Theorem 8.6 – **McDiarmid's Inequality.** *Let X be some set and let $f : X^m \to \mathbb{R}$ be a function such that for all i, there exists $c_i > 0$, such that for all $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_m, \mathbf{z}'_i$, the following holds:*

$$\left|f(\mathbf{z}_1, \ldots, \mathbf{z}_{i-1}, \mathbf{z}_i, \mathbf{z}_{i+1}, \ldots, \mathbf{z}_m) - f(\mathbf{z}_1, \ldots, \mathbf{z}_{i-1}, \mathbf{z}'_i, \mathbf{z}_{i+1}, \ldots, \mathbf{z}_m)\right| \leq c_i$$

Let $Z_1, Z_2, \ldots, Z_m$ be independent random variables taking values in $\mathcal{X}$. Then, for every $\epsilon > 0$, the following holds:

$$\mathbb{P}\left[f(Z_1, \ldots, Z_m) \geq \mathbb{E}\left[f(Z_1, \ldots, Z_m)\right] + \epsilon\right] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^m c_i^2}\right).$$

McDiarmid's inequality is a generalisation of Hoeffding's inequality. For instance, if $X = [a, b]$, using $f(z_1, \ldots, z_m) = \frac{1}{m}\sum_{i=1}^m z_i$ and $c_i = (b-a)/m$, we obtain Hoeffding's inequality. McDiarmid's inequality shows that as long as no single variable has significant influence over the function f, then the random variable $f(Z_1, \ldots, Z_m)$ is concentrated around its expectation.

We can now prove the following lemma.

Lemma 8.7. *Consider the same setting as Lemma 8.5. Let $\mathcal{G}$ be a class of functions from X to $[a, b]$ and let D be a distribution over X. Let $S = (\mathbf{z}_1, \ldots, \mathbf{z}_m) \sim D^m$, then for*

$$\Phi(S) = \Phi(\mathbf{z}_1, \ldots, \mathbf{z}_m) = \sup_{g \in \mathcal{G}} \left\{ \mathop{\mathbb{E}}_{\mathbf{z} \sim D}\left[g(\mathbf{z})\right] - \frac{1}{m}\sum_{i=1}^m g(\mathbf{z}_i) \right\},$$

the following holds with probability at least $1 - \delta$,

$$\Phi(S) \leq 2\mathsf{RAD}_m(\mathcal{G}) + (b-a) \cdot \sqrt{\frac{\log\frac{1}{\delta}}{2m}}.$$

Proof. For some $i \in [m]$, let S' be obtained by replacing $\mathbf{z}_i$ by $\mathbf{z}'_i$ for some $\mathbf{z}'_i \in X$. We have,

$$\begin{aligned}\Phi(S) - \Phi(S') &= \sup_{g \in \mathcal{G}} \left(\mathop{\mathbb{E}}_{D}[g] - \widehat{\mathbb{E}}_S[g] \right) - \sup_{g \in \mathcal{G}} \left(\mathop{\mathbb{E}}_{D}[g] - \widehat{\mathbb{E}}_{S'}[g] \right) \\ &\leq \frac{1}{m} \sup_{g \in G}(g(\mathbf{z}_i) - g(\mathbf{z}'_i)) \leq \frac{b-a}{m}\end{aligned}$$

Above, we used the fact that the difference of suprema can be upper bounded by the suprema of the difference, and that since both $g(\mathbf{z}_i), g(\mathbf{z}') \in [a,b]$, their difference is bounded by $(b-a)$. Since S and S' are symmetric, we can get the same bound for $\Phi(S') - \Phi(S)$. Thus, we can apply McDiarmid's inequality (Theorem 8.6) with $c_i = (b-a)/m$ for all i. Thus, we have,

$$\mathbb{P}\left[\Phi(S) \geq 2\mathsf{RAD}_m(\mathcal{G}) + \epsilon\right] \leq \mathbb{P}\left[\Phi(S) \geq \mathbb{E}\left[\Phi(S)\right] + \epsilon\right] \leq \exp(-2m\epsilon^2/(b-a)^2).$$

The result then follows by setting $\epsilon = (b-a) \cdot \sqrt{\frac{\log \frac{1}{\delta}}{2m}}$. □

We can of course define a function Ψ, which is like Φ, but with the quantity inside the supremum negated, i.e.

$$\Psi(S) = \sup_{g \in \mathcal{G}} \left\{ \frac{1}{m} \sum_{i=1}^{m} g(\mathbf{z}_i) - \mathop{\mathbb{E}}_{z \sim D}\left[g(\mathbf{z})\right] \right\},$$

and prove the equivalents of Lemmas 8.5 and 8.7. Thus, combining all this we can prove the following theorem.

Theorem 8.8. *Let $\mathcal{G}$ be a family of functions mapping $X \to [a,b]$ and let D be a distribution over X. Let $S = (\mathbf{z}_1, \ldots, \mathbf{z}_m) \sim D^m$ be a sample of size m drawn according to D. Then for any $\delta > 0$, with probability at least $1-\delta$, the following holds for all $g \in \mathcal{G}$,*

$$\left| \mathop{\mathbb{E}}_{\mathbf{z} \sim D}\left[g(\mathbf{z})\right] - \frac{1}{m} \sum_{i=1}^{m} g(\mathbf{z}_i) \right| \leq 2\mathsf{RAD}_m(\mathcal{G}) + (b-a) \cdot \sqrt{\frac{\log \frac{2}{\delta}}{2m}}.$$

The above theorem combined with the algorithm design paradigm in Section 8.1.3 shows that provided we find a minimiser of the empirical risk from a suitable class of functions, then we are guaranteed to have low true risk. Actually, if the algorithm outputs some function h from some class H of functions, and the loss function is ℓ, then we consider the class,

$$\mathcal{G} = \{(\mathbf{x}, y) \mapsto \ell(h(\mathbf{x}), y) \mid h \in H\},$$

and it is $\mathsf{RAD}_m(\mathcal{G})$ that we are interested in bounding. We will see how to apply Theorem 8.8 in the remaining sections of this chapter. However, first we will see some composition results concerning Rademacher Complexity that will be useful in deriving bounds on the Rademacher complexity of function classes of interest.

8.3.3 Composition Properties for Rademacher Complexity

We note (and prove) some results that indicate how the Rademacher complexity is affected by simple transformations to function classes.

Lemma 8.9. *Let X be some set and $S = (\mathbf{z}_1, \ldots, \mathbf{z}_m) \in X^m$. Let $\mathcal{F}, \mathcal{G}$ be classes of functions from X to some bounded interval $[a,b]$. Let $c, v \in \mathbb{R}$. The following hold:*

- *Define the class* $\mathcal{F}+\mathcal{G}$ *as,*

$$\mathcal{F}+\mathcal{G}=\{f+g \mid f\in\mathcal{F}, g\in\mathcal{G}\},$$

and note that the range of functions in $\mathcal{F}+\mathcal{G}$ *is contained in* $[2a,2b]$*. Then,*

$$\widehat{\mathsf{RAD}}_S(\mathcal{F}+\mathcal{G})=\widehat{\mathsf{RAD}}_S(\mathcal{F})+\widehat{\mathsf{RAD}}_S(\mathcal{G}).$$

- *Define the class* $c\mathcal{F}+v$ *as,*

$$c\mathcal{F}+v=\{cf+v \mid f\in\mathcal{F}\},$$

and note that the range of functions in $c\mathcal{F}+v$ *is contained in* $[-u+v,u+v]$ *where* $u=|c|\max\{|a|,|b|\}$*. Then,*

$$\widehat{\mathsf{RAD}}_S(c\mathcal{F}+v)=|c|\widehat{\mathsf{RAD}}_S(\mathcal{F}).$$

The proof of the above lemma is straightforward and is left as an exercise. Let us now see a result that will be quite useful. The following lemma is usually referred to as Talagrand's (Contraction) Lemma in the learning theory literature, as it follows from a much more general theory developed by Talagrand.

Lemma 8.10 **– Talagrand's Lemma.** *Let* $\mathcal{G}$ *be a family of functions from* $X\to[a,b]$ *and let* $S=(\mathbf{z}_1,\ldots,\mathbf{z}_m)\in X^m$*. Let* $\phi_1,\ldots,\phi_m$ *be univariate real-valued functions, for each* i*,* $\phi_i:[a,b]\to\mathbb{R}$ *being* l*-Lipschitz. Then,*

$$\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{g\in\mathcal{G}}\frac{1}{m}\sum_{i=1}^{m}\sigma_i\phi_i(g(\mathbf{z}_i))\right]\leq l\cdot\widehat{\mathsf{RAD}}_S(\mathcal{G}).$$

We will provide a proof of this lemma, though readers may well skip this proof without hurting their understanding the applications of this lemma. Before we do that, let us observe the following immediate corollary, which immediately follows from setting each ϕ_i to be the same in the above lemma.

Corollary 8.11. *Let* $\mathcal{G}$ *be a family of functions from* $X\to[a,b]$*, let* $\phi:[a,b]\to\mathbb{R}$ *be* l*-Lipschitz. Then define the class,*

$$\phi\circ\mathcal{G}=\{\phi\circ g \mid g\in\mathcal{G}\}.$$

Then, for any $S\in X^m$*,* $\widehat{\mathsf{RAD}}_S(\phi\circ\mathcal{G})\leq\widehat{\mathsf{RAD}}_S(\mathcal{G})$*.*

Proof of Lemma 8.10. We have the following:

$$\frac{1}{m}\cdot\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{g\in\mathcal{G}}\sum_{i=1}^{m}\sigma_i\phi_i g(\mathbf{z}_i)\right]=\frac{1}{m}\cdot\mathop{\mathbb{E}}_{\sigma_1,\ldots,\sigma_{m-1}}\mathop{\mathbb{E}}_{\sigma_m}\left[\sup_{g\in\mathcal{G}}u_{m-1}(g)+\sigma_m\phi_m(g(\mathbf{z}_i))\right],$$

where $u_{m-1}(g)=\sum_{i=1}^{m-1}\sigma_i\phi_i(g(\mathbf{z}_i))$. Let us concentrate on just the inner expectation:

$$\mathop{\mathbb{E}}_{\sigma_m}\left[\sup_{g\in\mathcal{G}}u_{m-1}(g)+\sigma_m\phi_m g(\mathbf{z}_m))\right].$$

Note that by definition of supremum, we have the existence of $g_1, g_2 \in \mathcal{G}$ satisfying the following for every $\epsilon > 0$:

$$u_{m-1}(g_1) + \phi_m(g_1(\mathbf{z}_m)) \geq \sup_{g \in \mathcal{G}} \left(u_{m-1}(g) + \phi_m(g(\mathbf{z}_m))\right) - \epsilon \tag{8.9}$$

$$u_{m-1}(g_2) - \phi_m(g_2(\mathbf{z}_m)) \geq \sup_{g \in \mathcal{G}} \left(u_{m-1}(g) - \phi_m(g(\mathbf{z}_m))\right) - \epsilon \tag{8.10}$$

Then, we have the following for every $\epsilon > 0$:

$$\mathop{\mathbb{E}}_{\sigma_m}\left[\sup_{g \in \mathcal{G}} u_{m-1}(g) + \sigma_m \phi_m(g(\mathbf{z}_m))\right] - \epsilon \leq \frac{1}{2}\left[u_{m-1}(g_1) + \phi_m(g_1(\mathbf{z}_m)) + u_{m-1}(g_2) - \phi_m(g_2(\mathbf{z}_m))\right]$$

As ϕ_m is l-Lipschitz, we have $|\phi_m(g_1(\mathbf{z}_m)) - \phi_m(g_2(\mathbf{z}_m))| \leq l \cdot |g_1(\mathbf{z}_m) - g_2(\mathbf{z}_m)| = ls(g_1(\mathbf{z}_m) - g_2(\mathbf{z}_m))$, where $s = \operatorname{sign}(g_1(\mathbf{z}_m) - g_2(\mathbf{z}_m))$. Thus, we have

$$\begin{aligned}\mathop{\mathbb{E}}_{\sigma_m}\left[\sup_{g \in \mathcal{G}} u_{m-1}(g) + \sigma_m \phi_m(g(\mathbf{z}_m))\right] - \epsilon &\leq \frac{1}{2}\left[u_{m-1}(g_1) + u_{m-1}(g_2) + ls(g_1(\mathbf{z}_m) - g_2(\mathbf{z}_m))\right] \\ &= \frac{1}{2}\left[u_{m-1}(g_1) + lsg_1(\mathbf{z}_m) + u_{m-1}(g_2) - lsg_2(\mathbf{z}_m)\right]\end{aligned}$$

As $\{s, -s\} = \{-1, 1\}$, we can rewrite the above as:

$$\mathop{\mathbb{E}}_{\sigma_m}\left[\sup_{g \in \mathcal{G}} u_{m-1}(g) + \sigma_m \phi_m(g(\mathbf{z}_m))\right] - \epsilon \leq \mathop{\mathbb{E}}_{\sigma_m}\left[\sup_{g \in \mathcal{G}} u_{m-1}(g) + \sigma_m l g(\mathbf{z}_m)\right]$$

As this inequality holds for every $\epsilon > 0$, we can in fact write,

$$\mathop{\mathbb{E}}_{\sigma_m}\left[\sup_{g \in \mathcal{G}} u_{m-1}(g) + \sigma_m \phi_m(g(\mathbf{z}_m))\right] \leq \mathop{\mathbb{E}}_{\sigma_m}\left[\sup_{g \in \mathcal{G}} u_{m-1}(g) + \sigma_m l g(\mathbf{z}_m)\right]$$

We can repeat the above for $i = m-1, m-2, \ldots, 1$, to show that,

$$\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{g \in \mathcal{G}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i \phi_i(g(\mathbf{z}_i))\right] \leq l \cdot \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{g \in \mathcal{G}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i g(\mathbf{z}_i)\right] = l \cdot \widehat{\mathsf{RAD}}_S(\mathcal{G})$$

□

In the following sections, we will see some examples of how these composition lemmas together with Theorem 8.8 can be used to give guarantees on either $\varepsilon(h)$ or $R(h) - \inf_{f \in \mathcal{F}} R(f)$ for learning algorithms derived using the paradigm outlined in Section 8.1.3.

8.4 Linear Regression

Let us look one of the most well-studied problem in statistics and machine learning—linear regression. In linear regression, we assume that the target

function, $f^\star$, is of the form $f^\star(x) = \mathbf{w}^\star \cdot \mathbf{x}$ for $\mathbf{w}^\star \in \mathbb{R}^n$. The goal is to estimate, $\widehat{f}$, represented by parameters $\widehat{w}$, such that,

$$\varepsilon(\widehat{f}) = \mathop{\mathbb{E}}_{\mathbf{z}\sim\mu}\left[(\mathbf{w}^\star \cdot \mathbf{x} - \widehat{\mathbf{w}^\star} \cdot \mathbf{x})^2\right] \leq \epsilon.$$

In order to apply Theorem 8.2 and Theorem 8.8, we will require some boundedness assumptions. We shall assume that there exists some W, such that $\|\mathbf{w}^\star\|_2 \leq W$. For some $R \geq 0$, let $\mathbb{B}_n(\mathbf{0}, R) = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\|_2 \leq R\}$. We will assume that the support of D is contained in $\mathbb{B}_n(\mathbf{0}, B) \times [-M, M]$ for some $B, M > 0$.

For any $\mathbf{w} \in \mathbb{R}^n$, let $f_\mathbf{w} : \mathbb{B}_n(\mathbf{0}, B) \to \mathbb{R}$ be the linear function defined as $f_\mathbf{w}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$. Consider the class of linear functions from $\mathbb{B}_n(\mathbf{0}, B) \to [-BW, BW]$ defined as,

$$\mathcal{F}_{W,B} = \{f_\mathbf{w} \mid \|\mathbf{w}\|_2 \leq W\}.$$

Note that using the Cauchy-Schwarz inequality, $|\mathbf{x} \cdot \mathbf{w}| \leq \|\mathbf{x}\|_2 \|\mathbf{w}\|_2 \leq BW$, so the range of these functions is indeed contained in $[-BW, BW]$ as stated.

8.4.1 The Least Squares Approach

Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ denote the observed training data sample. Let $\ell(\widehat{y}, y) = (\widehat{y} - y)^2$. The ERM approach suggests that we should obtain $\widehat{w}$ by minimising the *empirical risk*, $\widehat{R}_S(f)$ over $f \in \mathcal{F}_{W,B}$.

By slight abuse of notation, we can view the empirical risk directly as a function of the parameters $\mathbf{w}$,

$$\widehat{R}_S(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{w} \cdot \mathbf{x}_i - y_i)^2.$$

Then, if we define $K = \{\mathbf{w} \in \mathbb{R}^n \mid \|\mathbf{w}\|_2 \leq W\}$, we are solving the following optimisation problem,

$$\min_{\mathbf{w}\in K} \widehat{R}_S(\mathbf{w}).$$

For $\mathbf{x}, y$, define the function, $r : \mathbb{R}^n \to \mathbb{R}$, $r(\mathbf{w}; \mathbf{x}, y) = (\mathbf{w} \cdot \mathbf{x} - y)^2$. Thus, $\widehat{R}_S(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} r(\mathbf{w}; \mathbf{x}_i, y_i)$. We note that, $\nabla_\mathbf{w} r(\mathbf{w}; \mathbf{x}, y) = 2(\mathbf{w} \cdot \mathbf{x} - y)\mathbf{x}$ and the Hessian $H(r) = 2\mathbf{x}\mathbf{x}^\top \succeq 0$ is positive semi-definite. Thus, the function $\widehat{R}_S$ is a convex function of $\mathbf{w}$. Furthermore, since

$$\left\|\nabla_\mathbf{w} r(\mathbf{w}; \mathbf{x}, y)\right\|_2 \leq |2(\mathbf{w} \cdot \mathbf{x} - y)| \|\mathbf{x}\|_2,$$

under the boundedness conditions above, and using the Cauchy Schwarz inequality, we have $\left\|\nabla_\mathbf{w} \widehat{R}_S(\mathbf{w})\right\|_2 \leq 2(BW + M)B$. Clearly, for any $\mathbf{w}, \mathbf{w}' \in K$, we have $\left\|\mathbf{w} - \mathbf{w}'\right\|_2 \leq 2W$. Thus, we can apply Theorem 8.2 and by choosing T to be $\Theta\left(\frac{(BW+M)BW}{\epsilon^2}\right)$, we can obtain $\widehat{\mathbf{w}} \in K$ such that,

$$\widehat{R}_S(\widehat{\mathbf{w}}) \leq \min_{\mathbf{w}\in K} \widehat{R}_S(\mathbf{w}) + \epsilon.$$

It is worth pointing out that without constraining the solution to lie in K, there is a simple closed form solution to obtain a minimiser of $\widehat{R}_S(\mathbf{w})$.[7]

[7] The constrained problem is closely related to ridge regression which also has a closed form solution.

However, in certain cases gradient-based approaches might be preferable in any case, as the closed-form solution requires inverting matrices which is computationally expensive.

8.4.2 Rademacher Complexity of Linear Functions

Let us now compute the empirical Rademacher complexity for the class $\mathcal{F}_{W,B}$ defined above. For this purpose, let $X = \mathbb{B}_n(\mathbf{0}, B)$ and let $S = (\mathbf{x}_1, \ldots, \mathbf{x}_m) \in X^m$. We have,

$$\begin{aligned}\widehat{\mathsf{RAD}}_S(\mathcal{F}_{W,B}) &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\mathbf{w}\in\mathbb{B}_n(\mathbf{0},W)} \frac{1}{m}\sum_{i=1}^{m}\sigma_i(\mathbf{w}\cdot\mathbf{x}_i)\right] \\ &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{\mathbf{w}\in\mathbb{B}_n(0,W)} \left(\mathbf{w}\cdot\frac{1}{m}\sum_{i=1}^{m}\sigma_i\mathbf{x}_i\right)\right].\end{aligned}$$

Using the Cauchy-Schwartz Inequality (the equality case),

$$\widehat{\mathsf{RAD}}_S(\mathcal{F}_{W,B}) = W\cdot\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\left\|\frac{1}{m}\sum_{i=1}^{m}\sigma_i\mathbf{x}_i\right\|_2\right]$$

Using Jensen's Inequality,

$$\begin{aligned}\widehat{\mathsf{RAD}}_S(\mathcal{F}_{W,B}) &\le W\cdot\left(\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\left\|\frac{1}{m}\sum_{i=1}^{m}\sigma_i\mathbf{x}_i\right\|_2^2\right]\right)^{\frac{1}{2}} \\ &= W\cdot\left(\mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\frac{1}{m^2}\sum_{i=1}^{m}\|\mathbf{x}_i\|_2^2 + \frac{2}{m^2}\sum_{i<j}\sigma_i\sigma_j(\mathbf{x}_i\cdot\mathbf{x}_j)\right]\right)^{\frac{1}{2}}.\end{aligned}$$

As σ_i are i.i.d. and have mean 0, the cross terms are all 0. Using the bound $\|\mathbf{x}_i\|_2 \le B$, we get,

$$\widehat{\mathsf{RAD}}_S(\mathcal{F}_{W,B}) \le \frac{W\cdot B}{\sqrt{m}}.$$

Let us now see how we can apply the composition results from Section 8.3.3 to bound the Rademacher Complexity of classes of functions that also incorporate the loss. Let $\mathcal{F}$ be a family of functions from X_n to $[a, b]$ and furthermore suppose that for every $\mathbf{x} \in X_n$, $\|\mathbf{x}\|_2 \le B$. Let $X = X_n \times [-M, M]$ and let $S = (\mathbf{z}_1, \ldots, \mathbf{z}_m) \in X^n$, where each $\mathbf{z}_i = (\mathbf{x}_i, y_i)$. Let $\overline{S} = (\mathbf{x}_1, \ldots, \mathbf{x}_m) \in X_n^m$ denote the sequence obtained from S by ignoring the observations y_i.

Define the class of functions that map X to $[a - M, b + M]$,

$$\mathcal{F}^1 = \{(\mathbf{x}, y) \mapsto f(\mathbf{x}) - y \mid f \in \mathcal{F}\}.$$

An easy calculation shows that the Rademacher complexity, $\widehat{\mathsf{RAD}}_S(\mathcal{F}^1) = \widehat{\mathsf{RAD}}_{\overline{S}}(\mathcal{F})$. Alternatively, we may appeal to the first part of Lemma 8.9 by using the class $\mathcal{G}$ that contains only 1 function, $(\mathbf{x}, y) \mapsto y$. Clearly $\widehat{\mathsf{RAD}}_S(\mathcal{G}) = 0$.

Now consider the class of functions that map X to $[0, \alpha^2]$, where $\alpha = \max\{|a - M|, |b + M|\}$, defined as,

$$\mathcal{F}^\ell = \{(\mathbf{x}, y) \mapsto (f(\mathbf{x}) - y)^2 \mid f \in \mathcal{F}\}.$$

Alternatively, we can view $\mathcal{F}^\ell$ as,

$$\mathcal{F}^\ell = \{(\mathbf{x}, y) \mapsto \phi \circ h(\mathbf{x}, y) \mid h \in \mathcal{F}^1\},$$

where $\phi : [-\alpha, \alpha] \to \mathbb{R}$ is the square function, $\phi(z) = z^2$. We note that as $\phi'(z) = 2z$, ϕ is 2α-Lipschitz on the interval $[-\alpha, \alpha]$. Thus, using Lemma 8.10, we have,

$$\widehat{\mathsf{RAD}}_S(\mathcal{F}^\ell) \leq 2\alpha\widehat{\mathsf{RAD}}_S(\mathcal{F}^1) = 2\alpha\widehat{\mathsf{RAD}}_{\overline{S}}(\mathcal{F}).$$

In the case of linear functions, let $\mathcal{F}^\ell_{W,B}$ denote the corresponding class obtained when starting from the class $\mathcal{F}_{W,B}$. Note that if $f \in \mathcal{F}_{W,B}$ corresponds to some function, $h \in \mathcal{F}^\ell_{W,B}$, i.e. $h(\mathbf{x}, y) = (f(\mathbf{x}) - y)^2$, then,

$$\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D}\left[h(\mathbf{x}, y)\right] = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D}\left[(f(\mathbf{x}) - y)^2\right] = R(f).$$

In the case of linear functions, we have that,

$$\widehat{\mathsf{RAD}}_S(\mathcal{F}^\ell_{W,B}) \leq 2(BW + M) \cdot \frac{BW}{\sqrt{m}}.$$

Thus, applying Theorem 8.8, we get that provided $m = \Theta\left(\frac{BW(BW+M)+\log\frac{1}{\delta}}{\epsilon^2}\right)$, then with probability at least $1-\delta$, $\widehat{\mathbf{w}}$ obtained from the optimisation problem above would guarantee,

$$R(\widehat{\mathbf{w}}) \leq \min_{\mathbf{w}\in B_n(\mathbf{0},W)} R(\mathbf{w}) + O(\epsilon).$$

This completes the analysis in the case of Linear Regression.

8.5 Generalised Linear Models

Let us now look at a more expressive class of functions. In the statistics literature, these are referred to as generalised linear models. These are models of the form, $g(\mathbf{x}) = u(\mathbf{w} \cdot \mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{w} \in \mathbb{R}^n$ and $u : \mathbb{R} \to \mathbb{R}$. It is common to assume that u is monotone and Lipschitz, however these models can be defined more broadly. Suppose that u is strictly monotone, so that u^{-1} is well-defined. Then although g is no longer a linear function of $\mathbf{x}$, $u^{-1}(g(\mathbf{x}))$ is linear. The function u^{-1} is referred to as the link function.

Generalised linear models are widely used in machine learning and also form the basis of a single unit in most neural networks. Note that units with a sigmoid, hyperbolic tangent, or rectifier activation functions, are all generalised linear models.

In what follows we'll assume that u is monotonically increasing (not necessarily strict) and 1-Lipschitz, i.e. $|u(z) - u(z')| \leq |z - z'|$ for all $z, z' \in \mathbb{R}$. We will also assume that u is known to the learning algorithm. We will assume that the instance space is $X_n = \mathbb{B}_n(\mathbf{0}, B)$ for $B > 0$, and for $W > 0$ consider the class of generalised linear models, of functions from $X_n \to \mathbb{R}$:

$$\mathcal{G}_{W,B,u} = \{\mathbf{x} \mapsto u(\mathbf{w} \cdot \mathbf{x}) \mid \mathbf{w} \in \mathbb{B}_n(\mathbf{0}, W)\}$$

Note that as we are allowing W to be an arbitrary parameter, the requirement u is 1-Lipschitz is not an extra restrictions beyond just Lipschitzness. For example, if u were l-Lipschitz, we can use some $\widetilde{u}$, where $\widetilde{u}(z) = u(z/l)$ is 1-Lipschitz, and instead allow $\|\mathbf{w}\|_2$ to be as large as Wl.

For simplicity we'll assume that $u(0) = 0$ (although, this is not the case for some functions, we can easily centre u for the purpose of the algorithm and then undo the centring at the time of prediction). As in the previous section, suppose that the distribution D has support contained in $X = X_n \times [-M, M]$ for some $M \geq BW$. Then, observe that as $|\mathbf{w} \cdot \mathbf{x}| \leq BW$ for all $\mathbf{w} \in \mathbb{B}_n(\mathbf{0}, W)$ and $\mathbf{x} \in X_n$, and since u is 1-Lipschitz, it is possible for the distribution to satisfy the condition that $\mathbb{E}\left[y|\mathbf{x}\right] = u(\mathbf{w}^\star \cdot \mathbf{x})$ for some $\mathbf{w}^\star \in \mathbb{B}_n(\mathbf{0}, W)$. For the rest of the analysis in the section, we shall assume that this is the case, i.e. we are in the realizable setting.

8.5.1 Empirical Risk Minimisation

As in the case of linear regression, we can attempt to find a minimiser of the empirical risk, defined, with slight abuse of notation, on a sample $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, as,

$$\widehat{R}_S(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} (u(\mathbf{w} \cdot \mathbf{x}_i) - y_i)^2.$$

The trouble is that unlike in the case of linear regression, the empirical risk is no longer a convex function of $\mathbf{w}$. In fact, it has been shown by Auer et al. [8] that for even relatively simple inverse link functions, such as the sigmoid, $u(z) = \frac{1}{1+e^{-z}}$, the empirical risk may have exponentially many local minima as a function of the dimension.

Surrogate Loss Function

In order to avoid optimising a non-convex function (for which there are no general purpose algorithms), we'll use a strategy often employed in machine learning—using a surrogate convex loss function. Remarkably, in the case of generalised linear models, there exists a surrogate loss function, such that, in the *realizable setting*, the minimiser of the risk functional arising from this surrogate loss function and that from the squared loss is exactly the same! In addition, we can also show that an approximate minimiser of the risk for the surrogate loss function is also an approximate minimiser of the risk for squared loss.

Technically, the function we define is not *strictly* a loss function as we defined in this chapter. However, if u is strictly monotone, then we can by translation make it satisfy the conditions required for a loss function. Instead,

we will directly define the function, $r(\mathbf{w}; \mathbf{x}, y)$ which will define a penalty term for using the vector $\mathbf{w}$ on the datapoint $(\mathbf{x}, y)$. Formally, for $\mathbf{x} \in \mathbb{B}_n(\mathbf{0}, B), y \in [-M, M]$, and for $\mathbf{w} \in \mathbb{B}_n(\mathbf{0}, W)$, define

$$r(\mathbf{w}; \mathbf{x}, y) = \int_0^{\mathbf{w}\cdot\mathbf{x}} (u(z) - y)dz.$$

We will define the empirical risk of the surrogate loss as,

$$\widehat{R}_S^r(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} r(\mathbf{w}; \mathbf{x}_i, y_i)$$

It is useful to compute the gradient and Hessian of $r(\mathbf{w}; \mathbf{x}, y)$. Note that we have,

$$\begin{aligned}\nabla_{\mathbf{w}} r(\mathbf{w}; \mathbf{x}, y) &= (u(\mathbf{w}\cdot\mathbf{x}) - y)\mathbf{x},\\ H(r) &= u'(\mathbf{w}\cdot\mathbf{x})\mathbf{x}\mathbf{x}^\top.\end{aligned}$$

Since u is monotonically increasing, $u'(z) \geq 0$ for all z. What this shows is that $r(\mathbf{w}; \mathbf{x}, y)$ is a convex function for every $\mathbf{x}, y$. Since $\widehat{R}_S^r$ is an average of $r(\mathbf{w}; \mathbf{x}_i, y_i)$, $\widehat{R}_S^r$ is also convex and so we can apply convex optimisation algorithms to find the approximate minimiser of $\widehat{R}_S^r$.

Let us write down the gradient of $\widehat{R}_S^r(\mathbf{w})$,

$$\nabla_{\mathbf{w}}\widehat{R}_S^r(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}\nabla_{\mathbf{w}} r(\mathbf{w}; \mathbf{x}_i, y_i) = \frac{1}{m}\sum_{i=1}^{m}(u(\mathbf{w}\cdot x) - y_i)\mathbf{x}_i.$$

For comparison, let us also write the gradient of the empirical risk for the squared loss, $\widehat{R}_S(\mathbf{w})$,

$$\nabla_{\mathbf{w}}\widehat{R}_S(\mathbf{w}) = \frac{2}{m}\sum_{i=1}^{m}(u(\mathbf{w}\cdot\mathbf{x}_i) - y_i)u'(\mathbf{w}\cdot\mathbf{x}_i)\mathbf{x}_i.$$

Notice that apart from the factor 2, the main difference is that the i^{th} example has $u'(\mathbf{w}\cdot\mathbf{x}_i)$ as a multiplicative factor in the gradient. Although, $u' \geq 0$ as u is monotonically increasing, it may at times be very small.[8]

Next, let us also show that $\mathbf{w}^\star$, a minimiser of $R(\mathbf{w})$, is also a minimiser of $R^r(\mathbf{w})$, which is defined as,

$$R^r(\mathbf{w}) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim D}\left[r(\mathbf{w}; \mathbf{x}, y)\right].$$

Let $\mathbf{w}^\star \in \mathbb{R}^n$ be used to define the target function, i.e. it is the case that $\mathbb{E}\left[y|\mathbf{x}\right] = u(\mathbf{w}^\star\cdot\mathbf{x})$. Eq. (8.2) shows that $\mathbf{w}^\star$ is a minimiser of $R(\mathbf{w})$. Consider

[8] For instance, when u is the sigmoid function $u'(z) \approx 0$ when $|z|$ is somewhat large. This is also the reason why using the cross-entropy loss is better than using the squared loss to avoid the vanishing gradient problem.

the following for any $\mathbf{x} \in \mathbb{B}_n(\mathbf{0}, B)$:

$$\begin{aligned}\mathbb{E}\left[r(\mathbf{w};\mathbf{x},y)|\mathbf{x}\right] - \mathbb{E}\left[r(\mathbf{w}^\star;\mathbf{x},y)|\mathbf{x}\right] &= \mathbb{E}\left[\int_{\mathbf{w}^\star\cdot\mathbf{x}}^{\mathbf{w}\cdot\mathbf{x}} (u(z)-y)dz|\mathbf{x}\right] \\ &= \int_{\mathbf{w}^\star\cdot\mathbf{x}}^{\mathbf{w}\cdot\mathbf{x}} \Big(u(z) - \mathbb{E}\left[y|\mathbf{x}\right]\Big)\, dz \\ &= \int_{\mathbf{w}^\star\cdot\mathbf{x}}^{\mathbf{w}\cdot\mathbf{x}} (u(z) - u(\mathbf{w}^\star\cdot\mathbf{x}))dz \\ &\geq \frac{1}{2}(u(\mathbf{w}\cdot\mathbf{x}) - u(\mathbf{w}^\star\cdot\mathbf{x}))^2\end{aligned}$$

The last inequality in the calculation above follows from the fact that u is monotonically increasing and 1-Lipschitz. We can take expectations with respect to $\mathbf{x}$ to get,

$$R^r(\mathbf{w}) - R^r(\mathbf{w}^\star) \geq \frac{1}{2}\cdot \mathop{\mathbb{E}}_{\mathbf{x}\sim\mu}\left[(u(\mathbf{w}\cdot\mathbf{x}) - u(\mathbf{w}^\star\cdot\mathbf{x})^2\right] = \frac{\varepsilon(\mathbf{w})}{2}.$$

This shows that $R^r(\mathbf{w}) \geq R^r(\mathbf{w}^\star)$, i.e. $\mathbf{w}^\star$ is a minimiser of $R^r(\mathbf{w})$.

This shows that it is sufficient to identify a $\widehat{\mathbf{w}}$, for which $R^r(\mathbf{w})$ is at most $\frac{\epsilon}{2}$ larger than $R^r(\mathbf{w}^\star)$. In order to find a good enough minimiser of the empirical quantity, $\widehat{R}^r_S(\mathbf{w})$, it is sufficient to perform roughly $\Theta\left(\frac{WBM}{\epsilon^2}\right)$ projected gradient steps. We will see below how to use Rademacher complexity bounds to give bounds on $|R^r(\mathbf{w}) - \widehat{R}^r_S(\mathbf{w})|$ for all $\mathbf{w} \in \mathbb{B}_n(\mathbf{0}, W)$ with high probability.

Bounding the Generalisation Error for learning GLMs

Let us now consider the surrogate loss function, $r(\mathbf{w};\mathbf{x},y)$, used for learning GLMs. We can write $r(\mathbf{w};\mathbf{x},y)$ as follows:

$$r(\mathbf{w};\mathbf{x},y) = \int_0^{\mathbf{w}\cdot\mathbf{x}} (u(z)-y)dz = \left(\int_0^{\mathbf{w}\cdot\mathbf{x}} u(z)dz\right) - y(\mathbf{w}\cdot\mathbf{x})$$

Thus, we can write $r(\mathbf{w};\mathbf{x},y) = \psi(\mathbf{w}\cdot\mathbf{x}) - y(w\cdot x))$, where ψ is a BW-Lipschitz function.

Recall that $X_n = \mathbb{B}_n(\mathbf{0}, B)$ and consider the following classes of functions from $X_n \times [-M, M] \to \mathbb{R}$:

$$\begin{aligned}\mathcal{G}_{r,W} &= \{(\mathbf{x},y) \mapsto r(\mathbf{w};\mathbf{x},y) \mid \mathbf{w} \in \mathbb{B}(\mathbf{0}, W)\} \\ \mathcal{G}^1_{r,W} &= \{(\mathbf{x},y) \mapsto \psi(\mathbf{w}\cdot\mathbf{x}) \mid \mathbf{w} \in \mathbb{B}_n(\mathbf{0}, W)\} \\ \mathcal{G}^2_{r,W} &= \{(\mathbf{x},y) \mapsto -y(\mathbf{w}\cdot\mathbf{x}) \mid \mathbf{w} \in \mathbb{B}_n(\mathbf{0}, W)\}\end{aligned}$$

Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\} \subseteq X_n \times [-M, M]$. and let $\overline{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$. First, observer that

$$\mathcal{G}_{r,W} \subseteq \mathcal{G}^1_{r,W} + \mathcal{G}^2_{r,W}.$$

Thus, it suffices to bound $\widehat{\mathsf{RAD}}_S(\mathcal{G}^i_{r,W})$ for $i = 1, 2$ and use Lemma 8.9.

We know that if we consider the class of functions over $X_n \times [-M, M]$ of linear functions,

$$\mathcal{G}_W = \{(\mathbf{x},y) \mapsto \mathbf{w}\cdot\mathbf{x} \mid \mathbf{w} \in \mathbb{B}_n(\mathbf{0}, W)\},$$

then $\widehat{\mathsf{RAD}}_S(\mathcal{G}_W) \leq \frac{BW}{\sqrt{m}}$. Then, by applying Corollary 8.11, and using the fact that ψ is BW Lipschitz, we get

$$\widehat{\mathsf{RAD}}_S(\mathcal{G}^1_{r,W}) \leq \frac{(BW)^2}{\sqrt{m}}.$$

For, $\mathcal{G}^2_{r,W}$, we let $\psi_i(z) = -y_i z$ and observe that when $|y_i| \leq M$, each ψ_i is M-Lipschitz. We can then apply Lemma 8.10 to obtain that

$$\widehat{\mathsf{RAD}}_S(\mathcal{G}^2_{r,W}) \leq \frac{BWM}{\sqrt{m}}.$$

Putting everything together, we get that,

$$\widehat{\mathsf{RAD}}_S(\mathcal{G}_{r,W}) \leq \frac{BW(BW+M)}{\sqrt{m}}.$$

Using Theorem 8.8 and Theorem 8.2, this shows that the class of generalised linear models $\mathcal{G}_{W,u}$ can be learnt with running time polynomial in W, B, M, n, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ and with sample complexity polynomial in B, W, M, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Notice that there is no direct dependence on n in the sample complexity; it may appear implicitly through B and W. An advantage of this lack of dependence on n is that, provided B and W can be suitably bounded these models and algorithms can be kernelised.

8.5.2 Application to ℓ_p loss functions

Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\} \subseteq X \times [-M, M]$. Let H be a family of functions mapping $X \to [-W, W]$. Let $\overline{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$. Let $\phi(z) = |z|^p$ for $p \geq 1$; $|\phi'(z)| = p|z|^{p-1}$ (we can also consider $p = 1$, though it is not differentiable at 0). Thus, ϕ is pa^{p-1}-Lipschitz on the interval $[-a, a]$. Let $\mathcal{G} = \{(\mathbf{x}, y) \mapsto |h(\mathbf{x}) - y|^p \mid h \in H\}$ be a family of functions from $X \times [-M, M] \to [-a, a]$, where $a \leq (M+W)^p$. Suppose, $\widetilde{H} = \{h(\mathbf{x}) - y \mid h \in H\}$ be a family of functions from $X \times [-M, M] \to [-(M+W), (M+W)]$. Let us observe that,

$$\begin{aligned}
\widehat{\mathsf{RAD}}_S(\widetilde{H}) &= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h \in H} \frac{1}{m}\sum_{i=1}^{m} \sigma_i(h(x_i) - y_i)\right] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h \in H} \frac{1}{m}\sum_{i=1}^{m} \sigma_i h(x_i)\right] + \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\frac{1}{m}\sum_{i=1}^{m} \sigma_i y_i\right] = \widehat{\mathsf{RAD}}_{\overline{S}}(H)
\end{aligned}$$

Then, using Talagrand's lemma, we have that:

$$\widehat{\mathsf{RAD}}_S(\phi \circ \widetilde{H}) \leq p \cdot (W+M)^{p-1} \cdot \widehat{\mathsf{RAD}}_S(\widetilde{H}) = p \cdot (W+M)^{p-1} \cdot \widehat{\mathsf{RAD}}_{\overline{S}}(H)$$

For instance, when using the squared loss, we use $\phi(z) = |z|^2$, and thus, we get $\widehat{\mathsf{RAD}}_S(\phi \circ \widetilde{H}) \leq 2(W+M)\widehat{\mathsf{RAD}}_{\overline{S}}(H)$.

Chapter 9

Mistake-Bounded Learning

Thus far we've mainly looked at settings where there is an underlying distribution over the data and we are given access to an oracle that provides random labelled examples from this distribution. While this framework provides a useful way to analyse the behaviour of learning algorithms, it is not always the case that one may get independent training examples in practice. In reality, the distribution from which the data is generated may change over time. In this chapter, we will look at a specific learning framework that removes the requirement that data comes from a fixed distribution as (stochastically) independent examples.

9.1 Online Prediction Framework

We consider the setting where the learning algorithm is interacting with an *environment* and has to make predictions at discrete time-steps. Let X be an instance space and C a class of concepts.[1] The setup is as follows:

(a) At time t, the learning algorithm is presented an instance $\mathbf{x}_t \in X$.

(b) The learning algorithm makes a prediction $\widehat{y}_t \in \{0, 1\}$.

(c) The true label y_t is revealed and the learning algorithm is said to have made a mistake if $\widehat{y}_t \neq y_t$.

The process defined above repeats indefinitely for $t = 1, 2, \ldots$. How might one measure the performance of such a learning algorithm? For a learning algorithm L and infinite sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$, for time $t \in \mathbb{N}$, define,

$$\textsf{MISTAKES}\left(t; L, ((\mathbf{x}_i, y_i))_{i=1}^{\infty}\right) = \sum_{s=1}^{t} \mathbb{1}(\widehat{y}_s \neq y_s).$$

In the process defined above, the access L gets to the data $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ is sequential, where it has to make a prediction $\widehat{y}_t$ before seeing y_t. We will only consider sequences $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ for which there exists $c \in C$, such that

[1] We will forgo the slightly cumbersome notational overhead of writing $X = \bigcup_{n \geq 1} X_n$ and $C = \bigcup_{n \geq 1} C_n$ and implicitly assume that there is a parameter n that captures the size of instances. Likewise, we assume that there is a function size(c) that gives the representation size of concepts.

Algorithm 9.1: Mistake-bounded algorithm for learning conjunctions

Input: Sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ provided online.
// initialise hypothesis conjunction with all literals
Set $h_1 = z_1 \wedge \bar{z}_1 \wedge z_2 \wedge \bar{z}_2 \wedge \cdots \wedge z_n \wedge \bar{z}_n$
for $t = 1, 2, 3, \ldots$ **do**
    Receive $\mathbf{x}_t$
    $\widehat{y}_t = h_t(\mathbf{x}_t)$
    Receive y_t
    if $y_t \neq \widehat{y}_t$ **then** // mistake made
        for $j = 1, \ldots n$ **do**
            if $x_{t,j} = 0$ **then** // j^{th} bit of $\mathbf{x}_t$ is 0
                Drop z_j from h_t if it exists
            else // j^{th} bit of $\mathbf{x}_t$ is 1
                Drop $\bar{z}_j$ from h_t if it exists
    Call the resulting hypothesis h_{t+1}

$y_i = c(\mathbf{x}_i)$ for all i. We will say that C is learnable with a finite mistake bound B, if there exists an online learning algorithm L, that for every sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ that satisfies for some $c \in C$ that for all i, $y_i = c(\mathbf{x}_i)$, satisfies for all $t \in \mathbb{N}$, $\mathsf{MISTAKES}(t; L, ((\mathbf{x}_i, y_i))_{i=1}^{\infty}) \leq B$.

It is worth making a couple of observations at this point. If X is finite and there is no restriction on the computational resources available to L, one can always trivially get a mistake bound of $|X|$. We will typically be interested in algorithms that are efficient in their use of space and time. We will define the notion of efficiency later, but first let us consider an example. We will design an algorithm for online learning CONJUNCTIONS in the mistake-bounded setting. The Algorithm is described in Alg. 9.1.

Theorem 9.1. CONJUNCTIONS *can be learnt online with mistake-bound* $n+1$. *Furthermore, the running time of the algorithm is polynomial in* n *at each time-step* t.

Proof. First observe that because we start with all literals in the hypothesis conjunction and only drop literals when we are sure that the literal can't be part of the target conjunction, the only mistakes we make are of the form $y_t = 1$ and $\widehat{y}_t = 0$.

To begin, h_1 has $2n$ literals. When the first mistake occurs, exactly n literals are removed: for each i, exactly one of z_i or $\bar{z}_i$ is dropped. At every subsequent mistake at least one literal is dropped. Thus, the number of mistakes cannot exceed $n + 1$.

Finally, note that the algorithm is only maintaining a hypothesis conjunction and using it to make the prediction $\widehat{y}_t$. So the running time of the algorithm at each time-step is $O(n)$. □

Exercise: Show that the above bound is tight for this particular algorithm. What can you say about a general mistake bound for any algorithm for online learning CONJUNCTIONS?

9.1.1 Resource Constraints on Online Algorithms

In the most generous setting, we can allow the algorithm L to predict $\widehat{y}_t$ using any computable function of $(\mathbf{x}_1, y_1, \mathbf{x}_2, y_2, \ldots, \mathbf{x}_{t-1}, y_{t-1}, \mathbf{x}_t)$. For computationally efficient algorithms, we may require that this function be computable in time polynomial in n, size(c) and t. However, as we observed in the algorithm for learning CONJUNCTIONS, the algorithm did not need to store the entire history of observations, but the current hypothesis h_t was sufficient as a *sketch* of the history up to that point.

We consider space-bounded algorithms, where at time t, the algorithm maintains a state S_t, such that for each t, $|S_t| \leq \text{poly}(n, \text{size}(c))$. For an efficient algorithm, we will require that there are two polynomial time computable functions f and g, such that $\widehat{y}_t = f(S_t, \mathbf{x}_t)$ and $S_{t+1} = g(S_t, \mathbf{x}_t, y_t)$. Thus, the function f is used to make a prediction $\widehat{y}_t$ at time t, and g is used to update the state.

Note that the we can define $h_t : X \to \{0, 1\}$ as $h_t(\mathbf{x}) = f(S_t, \mathbf{x})$, thus essentially this is equivalent to the algorithm maintaining a hypothesis at each time t. Furthermore, because of the requirement that $|S_t| \leq \text{poly}(n, \text{size}(c))$, the total number of possible hypothesis is at most $2^{\text{poly}(n, \text{size}(c))}$, thus the algorithm in this case is making predictions using a hypothesis that comes from a fairly restricted class of hypotheses. We will refer to such algorithms as *efficient* online algorithms.

9.1.2 Conservative Online Learning

Definition 9.2 – **Conservative Online Learner.** *We say that an online learning algorithm is conservative, if it only changes its prediction rule after making a mistake. Equivalently it only updates its state if it makes a mistake.*

Proposition 9.3. *If C is learnable with a mistake bound B using an online learning algorithm A, then C is learnable with mistake bound B using a conservative online learning algorithm. The conservative online learning algorithm is efficient if A is efficient.*

Proof. The proof of this result is relatively straightforward. We design an algorithm A' as follows. A' initialises itself exactly the same way as A does. Let S'_t be the state of the A' at time t and let $m(t)$ denote the number of mistakes made by A' up to (but no including) time t. We will simulate A on a subsequence of examples on which A' makes mistakes. We will maintain the invariant that $S'_t = S_{m(t)+1}$. Note that by definition $S'_1 = S_1$.

A' behaves as follows. If there is no mistake at time t, then $S'_{t+1} = S'_t$. If on the other hand a mistake is made, then we pass the example $\mathbf{x}_t$ to the simulation of A and set $S'_{t+1} = S_{m(t+1)+1}$. Clearly A' is conservative by definition. However, the prediction rule used by A' at time t is the same as the one used by A at time $m(t) + 1$; as a result every time A' makes a mistake so does A. Since A has a mistake bound of B, so does A'. □

The requirement that an online learning algorithm be conservative is a natural one and the above result shows that it is not a restrictive one. This result will be useful to establish that efficient mistake-bounded online learning implies PAC learning.

9.2 Relationships to Other Models of Learning

In the PAC learning framework, we have access to an example oracle $\mathsf{EX}(c, D)$ that when queried returns an example $(\mathbf{x}, c(\mathbf{x}))$ where $\mathbf{x} \sim D$. Earlier in the course, we also considered two other oracles, a membership query oracle, $\mathsf{MQ}(c)$, which when queried with $\mathbf{x}$, returns $c(\mathbf{x})$, and an equivalence query oracle, $\mathsf{EQ}(c)$, which when queried with a hypothesis h, either returns that $c \equiv h$ or returns a counterexample $\mathbf{x}$, such that $h(\mathbf{x}) \neq c(\mathbf{x})$. We will now relate mistake-bounded learning to learning using some of these other oracles.

9.2.1 Relationship to PAC Learning

Theorem 9.4. *If C is efficiently learnable with a mistake bound B using an online learning algorithm A, where $B \leq \text{poly}(n, \text{size}(c))$, then C is efficiently PAC learnable.*

Proof. Without loss of generality, let A be a conservative online learning algorithm. We generate each example $(\mathbf{x}_t, y_t)$ at time t by querying the example oracle $\mathsf{EX}(c, D)$. Let h_t denote the hypothesis used by A at time t. Since A is conservative and has a mistake-bound of B, we need to consider no more than $B + 1$ distinct hypotheses.

We either stop the simulation of A after B mistakes have been made and output the hypotheses that is then guaranteed to be equal to the target c, or we stop the simulation if we simulate $s = \frac{1}{\epsilon}\log\frac{B}{\delta}$ steps without making a mistake. The probability that a hypothesis, h, with $\text{err}(h) \geq \epsilon$ will go for s steps with examples drawn i.i.d from $\mathsf{EX}(c, D)$, without making a mistake is at most $(1-\epsilon)^s \leq e^{-\epsilon s} \leq \delta/B$. Thus, a simple union bound suffices to show the correctness.

We remark that the condition that $B \leq \text{poly}(n, \text{size}(c))$ together with the efficiency of A suffices to conclude that the resulting PAC learning algorithm is efficient. Note that the sample complexity of the resulting algorithm is $O\left(\frac{B}{\epsilon}\log\frac{B}{\delta}\right)$. □

Remark 9.5. *If we wanted to allow non-efficient algorithms, but insist on polynomial sample complexity and polynomial-time evaluatability of the hypothesis class, we would still require $B \leq \text{poly}(n, \text{size}(c))$ and would require that the online learning algorithm A had a polynomial time prediction rule, even if the state update could potentially take more than polynomial time.*

9.2.2 Relationship to Learning Using Equivalence Queries

Proposition 9.6. *If C is learnable with a mistake bound B using an online algorithm A, then C can be learnt using $\mathsf{EQ}(c)$ only with at most $B+1$ equivalence queries. Furthermore, if A is efficient so is the algorithm that learns using $\mathsf{EQ}(c)$.*

Proof. Let h_t be the hypothesis used by A at time t for $t \geq 1$. We will query $\mathsf{EQ}(c)$ with h_t: if we get that $c \equiv h_t$, then we are done, otherwise we get a counterexample $\mathbf{x}_t$ which forces A to make a mistake at time t. In fact, this forces A to make mistakes at every single time-step in the simulation. Thus,

Algorithm 9.2: Halving algorithm for online learning C

Input: Sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ provided online.
Let $C_1 = C$
for $t = 1, 2, 3, \ldots$ **do**
    Receive $\mathbf{x}_t$
    $\widehat{y}_t = \text{majority}\{c(\mathbf{x}_t) | c \in C_t\}$
    Receive y_t
    if $y_t \neq \widehat{y}_t$ **then** // mistake made
        $C_{t+1} = \{c \in C_t \mid c(\mathbf{x}_t) = y_t\}$
    else
        $C_{t+1} = C_t$

after B mistakes h_{B+1} will be identical to c. We can verify this by an additional query to $\mathsf{EQ}(c)$. □

Proposition 9.7. *If C is learnable using only $\mathsf{EQ}(c)$ and makes at most Q queries to $\mathsf{EQ}(c)$, then C is learnable with a mistake bound of Q using an online algorithm. Furthermore, if the algorithm that learns using $\mathsf{EQ}(c)$ is efficient, then so is the online algorithm.*

Proof. Let L be the learning algorithm that only uses $\mathsf{EQ}(c)$. At any point in its simulation when it is about to make an equivalence query, it has a hypothesis h, we will use h to make predictions in the online setting. If we make a mistake, we have successfully simulated the oracle $\mathsf{EQ}(c)$ to get a counterexample. After Q such mistakes L has a hypothesis h that is equivalent to c and will make no further mistakes. Thus, the mistake bound is Q. □

Theorem 9.4 also follows using Proposition 9.6 and an exercise previously seen that simulates an equivalence oracle using $\mathsf{EX}(c, D)$. The results in this section show that online learning with a finite mistake bound is at least as hard as PAC learning. In fact it can be shown that it is strictly harder in that for the class of linear threshold functions in general we cannot get a finite mistake bound. We will discuss this point further in Section 9.4. We will study the Perceptron algorithm in Section 9.4 that gives a finite mistake bound for learning linear threshold functions with a margin. In Section 9.5, we will study an algorithm that learns sparse disjunctions with a significantly improved mistake-bound than can be achieved directly using the Perceptron algorithm.

9.3 The Halving Algorithm and Some Examples

In this section, we will see some *information-theoretic* bounds for mistake-bounded online learning algorithms. We will not be concerned with computational efficiency but see how these compare to other notions such as the VC dimension.

Theorem 9.8. *For any finite concept class C over an instance space X, the Halving algorithm (Alg. 9.2) has a mistake bound of $\log |C|$.*

Proof. The proof is immediate. If there is a mistake at time t, then $|C_{t+1}| \leq |C_t|/2$. Thus, after $\log_2 |C|$ mistakes, we can have at most one concept left, which must be the target concept. □

It is also straightforward to see that $\mathsf{VCD}(C)$ is a lower bound for any achievable mistake bound for deterministic algorithms, as we can give any learning algorithm points from a shattered set and force it to make a mistake on every one of them. We do know that for finite C, $\mathsf{VCD}(C) \leq \log |C|$. We will now see some examples where the gap between $\mathsf{VCD}(C)$ and $\log |C|$ is large and show that it is possible for the mistake bound to lie at either end of this interval.

Dictators

Let $X = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ be the instance space where $\mathbf{e}_i$ is the i^{th} basis vector which has 1 in the i^{th} co-ordinate and 0 everywhere else. Let C be the class of dictator functions $C = \{c_1, \ldots, c_n\}$, where $c_i(\mathbf{x}) = x_i$, i.e. the output of c_i is simply the i^{th} bit of $\mathbf{x}$ regardless of the remaining bits.[2] In this case, the mistake bound of the Halving Algorithm is 1, as is the $\mathsf{VCD}(C)$; obviously in this case $\log |C| = \log n$. As an exercise, the reader is invited to show that if $X = \{0,1\}^n$, then the mistake bound is indeed $\log n$ rather than 1.

Binary Search

Let $X = \{1, 2, \ldots, 2^n\} \subseteq \mathbb{N}$. Let C be the class of half intervals defined as $C = \{c_i | 0 \leq i \leq 2^n\}$, where,

$$c_i(x) = \begin{cases} 1 & \text{if } x \geq i \\ 0 & \text{otherwise} \end{cases}$$

In this case it is easy to see that $\mathsf{VCD}(C) = 1$, however the mistake-bound for any algorithm must be $\Theta(\log n) = \Theta(\log |C|)$.

9.4 Perceptron

The Perceptron algorithm is perhaps the most famous online learning algorithm. It was designed by Rosenblatt [41] and the first proofs of its convergence were given by Block [10] and Novikoff [39].

For the purpose of this section it will be convenient to treat Boolean functions as taking values in $\{-1, 1\}$ and also letting $\text{sign}(0) = 1$. We consider homogeneous halfspaces, or linear threshold functions passing through the origin, characterised by $\mathbf{w} \in \mathbb{R}^n$. This defines a threshold function $f_{\mathbf{w}} : \mathbb{R}^n \to \{-1, 1\}$ as,

$$f_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

The Perceptron algorithm is given as Alg. 9.3.

Before we formally analyse the algorithm, it is worth understanding geometrically what the algorithm is doing. Let $\mathbf{w}^\star$ be the true vector that defines the labels

[2] This terminology comes from social choice theory. The bits 0 and 1 can represent binary preferences of a group of n individuals and a dictator rule essentially uses the preference of the i^{th} individual, ignoring the rest. Obviously, the *majority* rule also lies in this framework of social choice theory.

Algorithm 9.3: The Perceptron Algorithm

Input: Sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ provided online.
// initialise with the $\mathbf{0}$ vector
Set $\mathbf{w}_1 = \mathbf{0} \in \mathbb{R}^n$
for $t = 1, 2, 3, \ldots$ **do**
    Receive $\mathbf{x}_t$
    $\widehat{y}_t = \text{sign}(\mathbf{x}_t \cdot \mathbf{w}_t)$
    Receive y_t
    if $y_t \neq \widehat{y}_t$ **then** // mistake made
        $\mathbf{w}_{t+1} = \mathbf{w}_t + y_t \mathbf{x}_t$
    else
        $\mathbf{w}_{t+1} = \mathbf{w}_t$

y_t. If the algorithm makes a mistake at time t it must be the case that $\mathbf{w}^\star \cdot \mathbf{x}_t$ and $\mathbf{w}_t \cdot \mathbf{x}_t$ have opposite signs. Thus adding $y_t \mathbf{x}_t$ to $\mathbf{w}_t$ has the effect of *rotating* $\mathbf{w}_t$ in the direction of $\mathbf{w}^\star$. This is reasonably correct intuition, but it is not perfectly accurate, it only does this on *average* as will be established by Lemmas 9.10 and 9.11. Rather than analyse the angles between $\mathbf{w}_t$ and $\mathbf{w}^\star$, it will be easier to show that the inner product between $\mathbf{w}_t$ and $\mathbf{w}^\star$ increases every time a mistake is made and that the length of $\mathbf{w}_t$ doesn't increase too much. This means that the increase in the inner product is at least in part caused by the decrease in the angle and not merely by the increase in the length. We state and prove the following theorem.

Theorem 9.9. *Suppose $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ is a sequence such that for every t, $\|\mathbf{x}_t\|_2 \leq D$ and there exists $\mathbf{w}^\star \in \mathbb{R}^n$ with $\|\mathbf{w}^\star\|_2 = 1$ and $\gamma > 0$, such that for every t, $y_t(\mathbf{w}^\star \cdot \mathbf{x}_t) \geq \gamma$. Then the total number of mistakes made by the Perceptron Algorithm (Alg. 9.3) with input sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ is bounded by D^2/γ^2.*

Proof. Let m_t denote the number of mistakes made by the Perceptron algorithm up to but not including time t. Using Lemmas 9.10 and 9.11, we have the following,

$$\begin{aligned} m_t \gamma &\leq \mathbf{w}^\star \cdot \mathbf{w}_t && \text{Lemma 9.10} \\ &\leq \|\mathbf{w}^\star\|_2 \|\mathbf{w}_t\|_2 && \text{By the Cauchy-Schwarz Inequality} \\ &\leq \sqrt{m_t} D. && \text{Lemma 9.11} \end{aligned}$$

This gives that $m_t \leq D^2/\gamma^2$ for every t.

□

Lemma 9.10. *Let m_t denote the number of mistakes made by the Perceptron algorithm (Alg. 9.3) up to, but not including, time t on a sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ satisfying the conditions of Theorem 9.9. Then,*

$$\mathbf{w}_t \cdot \mathbf{w}^\star \geq \gamma m_t.$$

Proof. We prove this by induction. When $t = 1$, $\mathbf{w}_t = \mathbf{0}$ and $m_t = 0$, so this is clearly true.

Now suppose this is true for time t. If no mistake occurs at time t, then $m_{t+1} = m_t$ and $\mathbf{w}_{t+1} = \mathbf{w}_t$, so the inequality continues to hold. On the other hand, if there is a mistake, then $m_{t+1} = m_t + 1$, and we have,

$$\begin{aligned}\mathbf{w}_{t+1} \cdot \mathbf{w}^\star &= (\mathbf{w}_t + y_t \mathbf{x}_t) \cdot \mathbf{w}^\star \\ &\geq m_t \gamma + y_t (\mathbf{x}_t \cdot \mathbf{w}^\star) \\ &\geq (m_t + 1)\gamma = m_{t+1}\gamma.\end{aligned}$$

This completes the proof. □

Lemma 9.11. *Let m_t denote the number of mistakes made by the Perceptron algorithm (Alg. 9.3) up to, but not including, time t on a sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ satisfying the conditions of Theorem 9.9. Then,*

$$\|\mathbf{w}_t\|_2^2 \leq D^2 m_t.$$

Proof. We prove this by induction. When $t = 1$, $\mathbf{w}_t = \mathbf{0}$ and $m_t = 0$, so this is clearly true.

Now suppose this is true for time t. If no mistake occurs at time t, then $m_{t+1} = m_t$ and $\mathbf{w}_{t+1} = \mathbf{w}_t$, so the inequality continues to hold. On the other hand, if there is a mistake, then $m_{t+1} = m_t + 1$, and we have,

$$\begin{aligned}\|\mathbf{w}_{t+1}\|_2^2 &= \|\mathbf{w}_t + y_t \mathbf{x}_t\|_2^2 \\ &\leq \|\mathbf{w}_t\|_2^2 + \|\mathbf{x}_t\|_2^2 + 2 y_t \mathbf{w}_t \cdot \mathbf{x}_t \\ &\leq m_t D^2 + D^2 \\ &\leq (m_t + 1) D^2 = m_{t+1} D^2.\end{aligned}$$

Above, in the second last step, we used the fact that $\|\mathbf{x}_t\|_2 \leq D$ and that $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) < 0$ as the Perceptron algorithm made a mistake at time t. This completes the proof. □

Some Comments

Notice that for convergence, we require a *margin condition*: $y(\mathbf{w}^\star \cdot \mathbf{x}) \geq \gamma > 0$ for all points $(\mathbf{x}, y)$ seen by the algorithm. Clearly, the condition implies that y has the same sign as $\mathbf{w}^\star \cdot \mathbf{x}$, but it further says that the projection of $\mathbf{x}$ in the direction of $\mathbf{w}^\star$ must have length at least γ (as $\mathbf{w}^\star$ is a unit vector). This means that there is a region of width 2γ around the hyperplane $\{\mathbf{x} \mid \mathbf{w}^\star \cdot \mathbf{x} = 0\}$ in which we do not observe any data. This is what it means to say that the linear threshold function has a *margin* γ under the distribution. On Problem 3 of Sheet 6, you will essentially show a matching lower bound on the mistakes made by the Perceptron algorithm. It is in fact possible to show that no online algorithm can achieve a finite mistake bound without a margin condition for learning threshold functions. This can already be seen in the one-dimensional case by generalising the binary search example from Section 9.3. It is worth recalling that in the PAC setting we do have efficient algorithms for learning linear threshold functions. Thus, this also yields a separation between PAC learning and mistake-bounded online learning.

It is also worth comparing the updates of the Perceptron algorithm to the algorithm we developed for learning GLMs. Note that we can consider sign :

$\mathbb{R} \to \mathbb{R}$ as a function that is monotone; it is however not Lipschitz. If we used the same surrogate loss approach we would get update rules that are the same as the one used by Perceptron (up to constant factors); this corresponds to an online gradient descent approach. The margin condition allows us to use a Lipschitz approximation of the sign function as there is no data in the region around the boundary.

9.5 The Winnow Algorithm

We will now look at an online learning problem where the target function depends on a relatively small number of features, but the total number of available features is very large. The particular example we shall look at is the class of monotone disjunctions. Let $X = \{0,1\}^n$ be the instance space, for any subset $S \subseteq [n]$, then define the monotone disjunction,

$$f_S(\mathbf{x}) = \bigvee_{i \in S} x_i.$$

The class of monotone disjunctions is the set of all such f_S, $S \subseteq [n]$. For any monotone disjunction, f_S, it can be expressed as a linear threshold function,

$$f_S(\mathbf{x}) = \text{sign}\left(\sum_{i \in S} x_i - \frac{1}{2}\right),$$

where we map $\{0,1\}$ to $\{-1,1\}$ in order to ensure the equivalence. This suggests that one can use the Perceptron algorithm for online learning monotone disjunctions. There are a couple of tweaks required to make this work. First, the way we have defined the Perceptron algorithm it only works for homogeneous halfspaces. This is relatively easy to handle. We instead consider the instance space to be X_{n+1} and let the last bit of all examples always be 1. Then, any threshold function of the form $\text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$ where $\mathbf{w} \in \mathbb{R}^n$, $\mathbf{x} \in X_n$ and $b \in \mathbb{R}$ can be written as $\text{sign}((\mathbf{w}, b) \cdot (\mathbf{x}, 1))$, where $(\mathbf{w}, b) \in \mathbb{R}^{n+1}$ and $(\mathbf{x}, 1) \in X_{n+1}$.

Second, we need to look at target vectors having norm 1 in order to apply Theorem 9.9. For $S \subseteq [n]$, let $\mathbf{w}_S^\star \in \mathbb{R}^{n+1}$ be such that $\mathbf{w}_i^\star = c$ if $i \in S$ and $\mathbf{w}_i^\star = 0$ for $i \notin S$ and $i \leq n$, and let $\mathbf{w}_{n+1}^\star = -c/2$. By setting $c^2 = 1/(|S| + \frac{1}{4})$ we can ensure that $\|\mathbf{w}^\star\|_2 = 1$. Note that $f_S(\mathbf{x}) = \text{sign}(\mathbf{w}_S^\star \cdot (\mathbf{x}, 1))$ for every $\mathbf{x} \in X_n$. Clearly, we have that $\left\|(\mathbf{x}, 1)\right\|_2^2 \leq n+1$. And $y(\mathbf{w}_S^\star \cdot (\mathbf{x}, 1)) \geq c/2$. Thus, we can use $\gamma = c/2$ and $D = \sqrt{n+1}$, to get a mistake bound of $4(n+1)(|S| + \frac{1}{4})$. If we knew that $|S| \leq k$, applying Theorem 9.4 this would give us a PAC learning algorithm with a sample complexity $O((nk/\epsilon)\log(nk/\epsilon))$. However, using Occam's Razor, we would expect a sample complexity of $O((k \log n)/\epsilon + \log(1/\delta)/\epsilon)$.

We will study an online algorithm called *Winnow* which achieves a mistake bound of $O(k \log n)$ for this problem. This will essentially give us a near optimal sample complexity after applying Theorem 9.4. The online algorithm itself is of independent interest as it uses multiplicative weight updates which we shall study in greater detail later. This algorithm and its analysis appeared in the work of Littlestone [35]. The algorithm in the paper can be generalised to certain types of linear threshold functions which we won't consider here.

Algorithm 9.4: The Winnow Algorithm

Input: Sequence $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ provided online.
// initialise with a vector of 1s
Set $\mathbf{w}_1 = (1, 1, \ldots, 1)$ of length n
for $t = 1, 2, 3, \ldots$ **do**
    Receive $\mathbf{x}_t \in \{0, 1\}^n$
    $\widehat{y}_t = \mathbb{1}\left(\mathbf{w}_t \cdot \mathbf{x}_t \geq \frac{n}{2}\right)$
    Receive y_t
    if $\widehat{y}_t = 1$ *and* $y_t = 0$ **then** // mistake made; elimination case
        for $j = 1, \ldots n$ **do**
            if $x_{t,j} = 1$ **then** // j^{th} bit of $\mathbf{x}_t$ is 1
                $w_{t+1,j} = 0$
            else
                $w_{t+1,j} = w_{t,j}$
    else if $\widehat{y}_t = 0$ *and* $y_t = 1$ **then** // mistake made; promotion case
        for $j = 1, \ldots n$ **do**
            if $x_{t,j} = 1$ **then** // j^{th} bit of $\mathbf{x}_t$ is 1
                $w_{t+1,j} = 2w_{t,j}$
            else
                $w_{t+1,j} = w_{t,j}$
    else
        $\mathbf{w}_{t+1} = \mathbf{w}_t$

Winnow starts by assigning a weight of 1 for every input feature. It adjusts these weights every time it makes a mistake by either doubling the weight for some feature or setting it to 0. Once a weight is set to 0 it can never become non-zero again. The predictions are made using a weighted majority rule. The main result we will prove is that for online learning monotone disjunctions, the mistake bound of Winnow is $O(k \log n)$, where k is the number of variables in the target disjunction.

Theorem 9.12. *Let $((\mathbf{x}_i, y_i))_{i=1}^{\infty}$ be an infinite sequence with $y_i = f_S(\mathbf{x}_i)$ for every i where $|S| = k$. Then if Winnow (Alg. 9.4) is given this input in an online fashion, the number of mistakes made by Winnow is at most $2k \log_2 n + 2$.*

Proof. We will prove this theorem through a sequence of claims which are proved separately. Let P be the number of times the algorithm makes mistakes of the form where $y_t = 1$ and $\widehat{y}_t = 0$. These are promotion steps, as the weights increase for some of the features. Let E be the number of times the algorithm makes mistakes of the form of $y_t = 0$ and $\widehat{y}_t = 1$. These are elimination steps, as some weights are set to 0.

The total number of mistakes is $P + N$. Claim 9.15 shows that $P \leq k \log_2 n$ and Claim 9.14 shows that $E \leq P + 2$. Together these yield the required result. □

Claim 9.13. *Under the conditions of Theorem 9.12, if $w_{t,i}$ is the weight of the i^{th} feature at time t for the Winnow Algorithm (Alg. 9.4), we have $w_{i,t} \leq n$.*

Proof. Suppose there is a mistake at time t. Clearly the weights only increase when $y_t = 1$ and $\widehat{y}_t = 0$. Only those indices i for which $x_{t,i} = 1$ can have their weights changed. The fact that $\widehat{y}_t = 0$ means that $w_{t,i} < n/2$ for each i such that $x_{t,i} = 1$. Then, it follows that $w_{t+1,i} \leq n$. □

Claim 9.14. *Under the conditions of Theorem 9.12, if P is the number of mistakes when $\widehat{y}_t = 0$ and E is the number of mistakes when $\widehat{y}_t = 1$, we have $E \leq P + 2$.*

Proof. We consider how the quantity $\sum_i w_{t,i}$ varies with time. Note that this quantity is always non-negative and when $t = 1$, $\sum_i w_{1,i} = n$. Every mistake where $y_t = 1$ and $\widehat{y}_t = 0$ increases the sum of weights by at most $n/2$. This is because the weights are doubled for all i such that $x_{t,i} = 1$; however, the reason $\widehat{y}_t = 0$ was that $\sum_i w_{t,i} x_{t,i} < n/2$. So doubling these weights can't increase the total weight by more than $n/2$.

On the other hand, by essentially the same logic, every mistake where $y_t = 0$ and $\widehat{y}_t = 1$ decreases the total weight by at least $n/2$.

Thus, we have for all t,

$$0 \leq \sum_i w_{t,i} \leq n + P \cdot \frac{n}{2} - E \cdot \frac{n}{2}.$$

The conclusion then follows. □

Claim 9.15. *Under the conditions of Theorem 9.12, if P is the number of mistakes when $\widehat{y}_t = 0$, we have $P \leq k \log_2 n$.*

Proof. We look at each time a mistake of the type where $y_t = 1$ and $\widehat{y}_t = 0$ occurs. There must be some $i \in S$, where S is the set of literals in the target disjunction, such that $x_{t,i} = 1$. Thus $w_{t,i}$ will be doubled. Note that for any $i \in S$, $w_{t,i}$ can never be set to 0 and every mistake that is counted in P doubles the weight of at least one of the relevant variables. The fact that $|S| \leq k$ and Claim 9.13 finishes the proof. □

Chapter 10

Online Learning with Expert Advice

In this chapter, we will move away from the *prediction* learning framework, either binary or real-valued, and consider a more abstract sequential decision making framework. The general framework turns out to be very powerful, and as applications, we will re-derive a boosting algorithm similar to AdaBoost, as well as see a proof of von Neumann's Min-Max theorem.

10.1 Learning with Expert Advice

We consider a sequential decision making problem as a repeated game between a learning algorithm or a *decision maker* (DM) and an environment. The game will be played for T rounds and we assume that T is known to the decision maker.[1] The decision maker has to choose between n options at each time step; the decision maker can pick a probability distribution over n options, i.e. a point from the probability simplex,

$$\Delta_n = \{\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^n, x_i \geq 0, \sum_i x_i = 1\}.$$

These options are sometimes called *experts*; this is because we can view this as the decision maker having advice from n different experts and seeking to combine the advice to come up with their own decision rule. Hence the name learning with *expert advice*. The game proceeds in rounds as follows. For $t = 1, 2, \ldots, T$:

- At time t, DM picks $\mathbf{x}_t \in \Delta_n$.
- The environment picks a loss vector $\mathbf{l}_t \in [0, 1]^n$.
- The loss suffered by DM at time t is $\mathbf{x}_t \cdot \mathbf{l}_t$ and DM observes the entire loss vector, $\mathbf{l}_t$.

First, we consider the choice of the decision maker. The decision maker may choose $\mathbf{x}_t$ at time t based on all the available information, i.e. $\mathbf{x}_1, \mathbf{l}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{l}_{t-1}$. In principle, we would allow $\mathbf{x}_t$ to be an arbitrary computable function of

[1] This is not a serious restriction. For tricks to remove this restriction refer to [19].

the history, though the algorithms we study turn out to be computationally efficient.

Second, we can consider the loss vectors $\mathbf{l}_t$ produced by the environment. We will not constrain the environment beyond the requirement that each entry of $\mathbf{l}_t$ be in the interval $[0, 1]$. In fact, the environment may chose the loss vector $\mathbf{l}_t$ in response to all the available history, including the choice $\mathbf{x}_t$ made by the decision maker at time t.

How do we measure performance of the decision maker in this case? Clearly simply requiring the DM to minimise loss across the T time steps is not sufficient as the environment could simply require all n options to have a loss of 1 at each time step. Instead, we will be interested in a notion of relative performance with respect to a class of strategies. We will look at the simplest set of strategies, i.e. strategies that don't change over time. However, we will find the *best* such strategy in hindsight. We define the notion of *regret* of the decision maker, DM, as,

$$\text{Regret}(\mathsf{DM}) = \sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t - \min_{\mathbf{x} \in \Delta_n} \sum_{t=1}^{T} \mathbf{x} \cdot \mathbf{l}_t.$$

It is worth commenting on the notion of regret before we design algorithms to minimise regret. The decision maker has the advantage of being able to change its decisions every time step but has the disadvantage of not knowing the future. On the other hand, the performance comparison is to a fixed strategy which has the benefit of hindsight. (It is worth observing that regret may be negative in this case.) What this suggests is that if there were a fixed strategy that would have performed "well", then having low regret would suggest that the decision maker performs almost as well. Thus, if the environment was relatively benign with loss vectors not changing very rapidly and the same options doing well over time, we would expect the decision maker to eventually figure out a good distribution over options. On the other hand, if the environment was very adversarial and no fixed strategy could have achieved a good performance then we would not expect the decision maker to have low loss even if it had low regret. Of course, one may take the view that the comparison to fixed strategies is too restrictive and we would like the decision maker to have low regret with more complex strategies. Indeed, such problems can be considered and the interested reader is referred to the excellent book by Cesa-Bianchi and Lugosi [19] for several generalisations and extensions.

A natural interpretation of the above formulation is in terms of betting over n different options which may change in value over time in uncertain ways. We can spread a unit amount of money every day and we would like our long term performance to be not much worse than the single best option (or single distribution) in hindsight. In fact, because the loss function $\mathbf{l} \cdot \mathbf{x}$ is linear in $\mathbf{x}$, the minimum is (also) achieved at a vertex of the simplex Δ_n. So we may in fact simply compare our performance with the best single option and rewrite the definition of regret as,

$$\text{Regret}(\mathsf{DM}) = \sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t - \min_{i \in [n]} \sum_{t=1}^{T} l_{t,i}.$$

10.2 Follow The Leader

We will use the notation $\mathbf{L}_t = \sum_{s=1}^{t-1} \mathbf{l}_s$. A natural strategy is to pick $\mathbf{x}_t$ at time t that minimises the *cumulative* historical loss, i.e. pick $\mathbf{x}_t \in \Delta_n$ that minimises $\mathbf{x}_t \cdot \mathbf{L}_t$. In fact, this $\mathbf{x}_t$ can be chosen to be one of the vertices of the simplex Δ_n, i.e. we may pick $\mathbf{x}_t$ with $x_{t,i} = 1$ for $i \in \operatorname{argmin}_i L_{t,i}$ and $x_{t,j} = 0$ for $j \neq i$. This algorithm is sometimes called *Follow-the-Leader* as it picks the strategy that has historically proved the best.

It turns out however that a strategy that would have performed best on historical observations, may not perform well in the future, and indeed this algorithm does not get a good bound on the regret. The following example with only two options, when T is even, establishes the problem with this strategy.

	$\mathbf{l}_1$	$\mathbf{l}_2$	$\mathbf{l}_3$	$\mathbf{l}_4$	...	$\mathbf{l}_T$
Option 1	0.5	0	1	0	...	0
Option 2	0	1	0	1	...	1

The strategy may pick either option at time $t = 1$, but subsequently it would always pick option 2 in even time steps and option 1 in odd time steps. Thus, it would incur a total loss of at least $T - 1$. However, the first option only has a loss of $T/2 - 1/2$. Thus, the regret incurred is at least $T/2 - 1/2$ which grows linearly in T. If we ignore constant factors, this is a completely trivial regret bound as no algorithm incurs a loss of more than T as the losses are constrained to be in $[0, 1]$ at each time step; as all losses are non-negative the regret can be at most T.

Although the Follow-the-Leader strategy does not work well in this particular case, there are online optimisation problems in which the strategy does give non-trivial (and in fact close to optimal) regret bounds. The problem in this specific setting is that the decision rule is too unstable, i.e. it may change drastically at every time step based on small changes to the loss vectors. In the next section, we will see a small modification to the algorithm that is more stable that does actually give optimal regret bounds.

10.3 The Multiplicative Weight Update Algorithm (MWUA)

Algorithm 10.1 presents the Multiplicative Weight Update Algorithm (MWUA) for the sequential decision making problem. This algorithm, or close variants, go by various names including Weighted Majority, Exponentially-weighted Forecaster, Hedge, etc. The reason for these names and indeed variants is that very similar abstractions of this sequential decision making problem have been studied in various fields including information theory, game theory, learning theory, complexity theory, among others. The excellent survey article by Arora et al. [7] gives several variants of this algorithm and applications to approximation algorithms and optimisation. The book by Cesa-Bianchi and Lugosi [19] considers various problems in learning theory, game theory, information theory and optimisation.

Before we analyse the algorithm, let us see in what sense this algorithm is a tweak of the *Follow-the-leader Algorithm*. Let $\mathbf{L}_t = \sum_{s=1}^{t-1} \mathbf{l}_s$ as defined above. Then the Follow-the-leader strategy simply picks $\mathbf{x}_t$ that minimises $\mathbf{L}_t$, which

Algorithm 10.1: The Multiplicative Weight Update Algorithm (MWUA)

Input: Time Horizon T; parameter η
// initialise $\mathbf{w}_1$ with a vector of 1s
Let $\mathbf{w}_1 = (1, \ldots, 1)^\top \in \mathbb{R}^n$
for $t = 1, 2, \ldots, T$ **do**
    $\mathbf{x}_t = \frac{\mathbf{w}_t}{Z_t}$, $Z_t = \sum_{i=1}^n w_{t,i}$
    Play $\mathbf{x}_t$
    Receive loss $\mathbf{x}_t \cdot \mathbf{l}_t$
    Observe $\mathbf{l}_t$
    for $j = 1, \ldots n$ **do**
        $w_{t+1,j} = w_{t,j} \cdot \exp(-\eta l_{t,i})$

can be represented in vector form as picking from $\operatorname{argmin}_{\mathbf{x}\in\Delta_n} \mathbf{x} \cdot \mathbf{L}_t$. In fact, all vectors in $\operatorname{argmin}_{\mathbf{x}\in\Delta_n} \mathbf{x} \cdot \mathbf{L}_t$ are exactly the subgradients of the function $f : \mathbb{R}^n \to \mathbb{R}$, given by $f(\mathbf{z}) = \min_i z_i$ at the point $\mathbf{L}_t$. Instead, if we define the function,

$$f^\eta(\mathbf{z}) = -\frac{1}{\eta} \ln \left(\sum_{i=1}^n e^{-\eta z_i} \right),$$

then it is easy to see that in the limit as $\eta \to \infty$ $f^\eta(\mathbf{z}) = \min_i z_i$. Thus, f^η is from a family of *soft min* functions, which are differentiable; the gradient of f^η is called the argsoftmin function and indeed the MWUA algorithm sets $\mathbf{x}_t = \nabla f^\eta(\mathbf{L}_t)$. The *softening* of the min function can be viewed as a form of regularisation. Yet another view of the $\mathbf{x}_t$ picked by the algorithm is the following, $\mathbf{x}_t$ is the constrained minimum of the following optimisation problem,

$$\min_{\mathbf{x}\in\Delta_n} \mathbf{L}_t \cdot \mathbf{x} - \frac{1}{\eta} H(\mathbf{x}).$$

where $H(\mathbf{x}) = -\sum_{i=1}^n x_i \ln x_i$ is the entropy function. The *negative* entropy term above acts as a regulariser. Minimising the function $\mathbf{L}_t \cdot \mathbf{x}$ pushes the solution to a corner of the simplex, whereas minimising the negative entropy term pushes it towards the centre of the simplex, i.e. encourages more *hedging*.

Indeed, many analyses of the regret bound of MWUA are possible using the interpretations above, some of which lead to more insightful proofs. We will consider a short potential-based proof which uses Z_t as a potential function.

Theorem 10.1. *For the* MWUA *algorithm run with* $\eta = \sqrt{\frac{\ln n}{T}}$*, we have,*

$$\text{Regret}(\text{MWUA}) \leq 2\sqrt{T \ln n}.$$

Proof. Consider the following,

$$\frac{Z_{t+1}}{Z_t} = \frac{\sum_{i=1}^n w_{t+1,i}}{Z_t} = \frac{\sum_{i=1}^n w_{t,i} \exp(-\eta l_{t,i})}{Z_t}$$

By the convexity of $z \mapsto e^{-\eta z}$, we have $\exp(-\eta z) \leq 1 + (e^{-\eta} - 1)z$ for $z \in [0, 1]$. Thus, we have,

$$\begin{aligned}\frac{Z_{t+1}}{Z_t} &\leq \sum_{i=1}^{t} \frac{w_{t,i}}{Z_t} \cdot (1 + (e^{-\eta} - 1) l_{t,i}) \\ &= 1 + (e^{-\eta} - 1)\mathbf{x}_t \cdot \mathbf{l}_t.\end{aligned}$$

Above, we have used the definition $\mathbf{x}_t = \mathbf{w}_t / Z_t$. Further, using the fact that $1 + x \leq e^x$, we have,

$$\frac{Z_{t+1}}{Z_t} \leq \exp\left((e^{-\eta} - 1)\mathbf{x}_t \cdot \mathbf{l}_t\right).$$

We can multiply the ratios Z_{t+1}/Z_t for $t = 1, \ldots, T$, and use the bound above to obtain,

$$\frac{Z_{T+1}}{Z_1} \leq \exp\left((e^{-\eta} - 1) \cdot \sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t\right). \tag{10.1}$$

On the other hand, we note that $w_{t,i} = \exp(-\eta \sum_{s=1}^{t-1} l_{s,i})$. So we have the following for each $i \in [n]$,

$$\frac{Z_{T+1}}{Z_1} \geq \frac{w_{T+1,i}}{n} = \frac{\exp\left(-\eta \sum_{t=1}^{T} l_{t,i}\right)}{n}. \tag{10.2}$$

Combining Equations (10.1) and (10.2), and taking natural logarithms, we have for each $i \in [n]$,

$$-\eta \sum_{t=1}^{T} l_{t,i} - \ln n \leq (e^{-\eta} - 1) \sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t.$$

Moving $\ln n$ to the RHS and adding $\eta \sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t$ to both sides, we get,

$$\eta \left(\sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t - \sum_{t=1}^{T} l_{t,i} \right) \leq (e^{-\eta} - (1 - \eta)) \sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t + \ln n.$$

Using the fact that $e^{-\eta} \leq 1 - \eta + \eta^2$ for $\eta \in [0, 1]$ and dividing both sides by η, we have for all i,

$$\sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t - \sum_{t=1}^{T} l_{t,i} \leq \eta \sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t + \frac{\ln n}{\eta}.$$

Finally, noticing that $\mathbf{x}_t \cdot \mathbf{l}_t \leq 1$ for each t, setting η as in the statement of the theorem, and observing that because the above holds for each $i \in [n]$, by linearity, it applies to each $\mathbf{x} \in \Delta_n$, we get the result as follows:

$$\sum_{t=1}^{T} \mathbf{x}_t \cdot \mathbf{l}_t - \sum_{t=1}^{T} \sum_{i=1}^{n} x_i l_{t,i} \leq \eta T + \frac{\ln n}{\eta} \leq 2\sqrt{T \ln n}.$$

As the above applies to each $\mathbf{x} \in \Delta_n$, the proof is complete. □

We remark that the choice $\mathbf{x}_t$ made by the algorithm at time t only depends on the loss vectors $\mathbf{l}_1, \ldots, \mathbf{l}_{t-1}$ given by the environment (in fact only on their sum). There is no dependence whatsoever on the past choices made by the algorithm, and the algorithm itself is completely deterministic. Hence, even if the environment completely adversarially picks the loss vectors $\mathbf{l}_t$ the algorithm still retains the guarantee. In fact, there is no need for the environment to wait to see the choice made by the algorithm as it can be computed directly using the previous loss vectors. In this sense, the MWUA algorithm achieves guarantees against adaptive and adversarial environments. This particular aspect will be useful in applications to game theory.

10.3.1 Lower Bound

In this section, we will give a sketch of why the bound achieved in Theorem 10.1 is essentially tight (up to constant factors). We will not give a formal proof, though writing a formal proof is not very difficult. We will also only establish a lower bound of $\Omega(\sqrt{T})$ for deterministic algorithms, when there are only $n = 2$ options. The environment at time t picks the loss vector $(1, 0)$ or $(0, 1)$ with equal probability independently at each time step. Now for any algorithm, its expected loss is exactly $T/2$. On the other hand the expectation of $\min_{i \in \{1,2\}} \sum_{t=1}^{T} l_{t,i}$ is $T/2 - \Omega(\sqrt{T})$. This follows from a simple exercise: if $X \sim \text{Binomial}(T, 1/2)$, what is the $\mathbb{E}\left[\min\{X, T - X\}\right]$? This shows that the $\sqrt{T}$ factor in the regret bound is tight; a slightly more detailed construction shows that $\sqrt{\ln n}$ is also required in the regret bound as a multiplicative factor. The formal proof appears in [19].

10.4 Application: Boosting Algorithm

In this section, we will see an application of the MWUA algorithm to get a boosting algorithm. As in the case of AdaBoost, we will assume that the weak learner returns hypotheses from a hypothesis class, H, of finite VC dimension and focus on finding a (combined) hypothesis that is consistent with the training data. We will not worry about the generalisation error here, as it can be bounded using the bound on the VC dimension of the resulting classifier exactly as in the case of AdaBoost. In fact, there are many similarities between the AdaBoost algorithm and MWUA and indeed in their analyses.

Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ be a training dataset with m points, where $y_i = c(\mathbf{x}_i)$ for some $c \in C$. Here C is the concept class for which we have a weak learning algorithm available.

We are going to treat this as a sequential decision making problem with m options. At each time t, the decision maker, DM, will use MWUA to pick a distribution, which we will call $\mathbf{d}_t \in \Delta_m$ to distinguish it from the data $(\mathbf{x}, y)$.

We will also simulate the environment. For this reason, we will use the weak learning algorithm, WeakLearn. We shall assume for simplicity that the weak learning algorithm succeeds with probability 1 instead of probability $1 - \delta$. At time t, let h_t be the hypothesis returned by the weak learning algorithm with respect to the distribution $\mathbf{d}_t \in \Delta_m$. Note that the distribution $\mathbf{d}_t$, puts probability mass $d_{t,i}$ on $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, m$. In particular, we have $\text{err}(h_t; \mathbf{d}_t) \leq \frac{1}{2} - \gamma$, where γ is the parameter of WeakLearn. We define

the loss vector $\mathbf{l}_t \in [0,1]^m$ as follows, $l_{t,i} = 1$ if $h_t(\mathbf{x}_i) = y_i$ and $l_{t,i} = 0$ otherwise. Based on the construction and the property of the weak learner, we have $\mathbf{d}_t \cdot \mathbf{l}_t = 1 - \text{err}(h_t; \mathbf{d}_t) \geq \frac{1}{2} + \gamma$ for every t.

We can make use of the regret guarantee. Note that $l_{t,i} = \mathbb{1}(h_t(\mathbf{x}_i) = y_i)$. Theorem 10.1 allows us to conclude that provided the DM uses MWUA to generate the decisions $\mathbf{d}_t \in \Delta_m$, we have for each i,

$$\sum_{t=1}^{T} \mathbf{d}_t \cdot \mathbf{l}_t - \sum_{t=1}^{T} \mathbb{1}(h_t(\mathbf{x}_i) = y_i) \leq 2\sqrt{T \ln m}.$$

Using, the fact that $\mathbf{d}_t \cdot \mathbf{l}_t \geq 1/2 + \gamma$ for each t, and rearranging, we get,

$$T \cdot \left(\frac{1}{2} + \gamma\right) - 2\sqrt{T \ln m} \leq \sum_{t=1}^{T} \mathbb{1}(h_t(\mathbf{x}_i) = y_i).$$

Hence,

$$\frac{T}{2} + \sqrt{T} \cdot (\gamma\sqrt{T} - 2\sqrt{\ln m}) \leq \sum_{t=1}^{T} \mathbb{1}(h_t(\mathbf{x}_i) = y_i).$$

Observe that for $T > 4 \ln m/\gamma^2$, the LHS is strictly larger than $T/2$. This means that the majority of the hypotheses h_t classify the i^{th} example $(\mathbf{x}_i, y_i)$ correctly for every i. Thus simply outputting majority$\{h_1(\mathbf{x}), \ldots, h_T(\mathbf{x})\}$ will give a hypothesis that is consistent with the training set. Note that the number of calls made to the weak learning algorithm is of the same order as that made by AdaBoost and as the majority function can be written as thresholds of hypotheses from the hypothesis class used by WeakLearn, the generalisation error can be bounded in exactly the same way as in the analysis of AdaBoost.

10.5 Application: von Neumann's Min-Max Theorem

As another application of the MWUA algorithm, we will give a proof von Neumann's Min-Max theorem. We will be brief and terse in our description of two player zero-sum games. Readers unfamiliar with these notions may wish to read an introductory chapter in a book on game theory.

We will consider finite two player zero-sum games with bounded payoffs. We will assume that all payoffs for one player are bounded in $[0,1]$; so for the other player they are in $[-1,0]$. The game has two players, typically called a row player and a column player. The game is specified as an $n \times m$ matrix, P, with n rows and m columns. The row player has to pick one of n options and the column player has to pick one of m options. If the row player picks i and the column player picks j, then the row player gets a payoff of P_{ij}. (As it is a zero-sum game, the payoff to the column player is $-P_{ij}$.) Clearly, there is an advantage in moving *second* as you can observe the option picked by the other player. Suppose the row player moves first: if they pick i, the column player will certainly pick j that will minimise the payoff of the row player (as it is a zero sum game), resulting in a payoff of $\min_j P_{ij}$; thus to maximise their payoff, the row player will pick i for which this quantity is the largest. In other words, the best achievable payoff for the row player is $\max_i \min_j P_{ij}$.

An entirely identical argument shows that the best payoff achieved by the row player if they go second is $\min_j \max_i P_{ij}$. This is of course assuming that both players are playing rationally. Thus, clearly we have that,

$$\max_i \min_j P_{ij} \leq \min_j \max_i P_{ij}.$$

When players are forced to pick a single option, something which is known as *pure strategies*, the above inequality can indeed be strict. The following simple game in which both players have two options establishes a strict inequality.

	1	2
1	1	0
2	0	1

For this game, $\max_i \min_j P_{ij} = 0$ and $\min_j \max_i P_{ij} = 3$. However, we can also consider *mixed* strategies, where rather than picking a single option, each player picks a distribution over their options. The other player can see the distribution of their competitor, but not the actual random choice. For example, think of playing rock-paper-scissors with someone, where you know their (random) strategy, but can't predict what particular random choice they will make when actually playing the game. Now suppose the row player has a strategy $\mathbf{x} \in \Delta_n$ and a column player has strategy $\mathbf{y} \in \Delta_m$, then the payoff to the row player (in expectation) is,

$$\mathop{\mathbb{E}}_{i \sim \mathbf{x}, j \sim \mathbf{y}} \left[P_{ij} \right] = \mathbf{x}^\top P \mathbf{y}.$$

We can similarly define optimal strategies for the row player depending on whether they are going first or second. When the row player goes first, the optimal value they can achieve is called the maxmin value, denoted by v_{maxmin}, and defined as,

$$v_{\text{maxmin}} = \max_{\mathbf{x} \in \Delta_n} \min_{\mathbf{y} \in \Delta_m} \mathbf{x}^\top P \mathbf{y}.$$

Similarly, the best value achievable by the row player if they go second is the minmax value, denoted by v_{minmax}, and defined as,

$$v_{\text{minmax}} = \min_{\mathbf{y} \in \Delta_m} \max_{\mathbf{x} \in \Delta_n} \mathbf{x}^\top P \mathbf{y}.$$

As in the previous case, we can immediately see that,[2]

$$v_{\text{maxmin}} \leq v_{\text{minmax}}. \tag{10.3}$$

What von Neumann's theorem states is that, for finite two player zero-sum games, $v_{\text{maxmin}} = v_{\text{minmax}}$.

We could of course frame this theorem in terms of losses rather than payoffs, but that would deviate from standard statements of this result. Instead, we make a series of observations. It is without loss of generality to assume that payoffs are in $[0, 1]$. First, we can always make payoffs for the row player to be *positive* because in terms of solution concepts, zero-sum games and constant-sum games are identical. Second, we can always scale payoffs so that they are

[2] Essentially, this part of the argument is weak duality, and von Neumann's theorem states that strong duality holds.

in the interval $[0,1]$. Finally, we can replace losses in the sequential decision making problem by payoffs, or rather consider loss vectors as coming from payoff vectors, $\mathbf{p}_t$ and set $\mathbf{l}_t = \mathbf{1} - \mathbf{p}_t$. Then, we can rephrase the statement of Theorem 10.1 as,

$$\text{Regret}(\mathsf{MWUA}) = \max_{\mathbf{x}\in\Delta_n} \sum_{t=1}^{T} \mathbf{x}\cdot\mathbf{p}_t - \sum_{t=1}^{T} \mathbf{x}_t\cdot\mathbf{p}_t \leq 2\sqrt{T\ln n}. \tag{10.4}$$

We will simply run the algorithm by using $\mathbf{l}_t = \mathbf{1} - \mathbf{p}_t$ as our loss vectors when we are actually given payoff vectors $\mathbf{p}_t$.

With that out of the way, we are ready to give a proof of von Neumann's Min-Max theorem using the MWUA algorithm. The row player will implement the MWUA algorithm (with payoff vectors suitably converted to loss vectors). Let $\mathbf{x}_t \in \Delta_n$ be the strategy picked by MWUA at time t. The column player picks $\mathbf{y}_t$ as,

$$\mathbf{y}_t \in \operatorname*{argmin}_{\mathbf{y}} \mathbf{x}_t^\top P\mathbf{y}.$$

Clearly, we have that,

$$\mathbf{x}_t^\top P\mathbf{y} = \min_{\mathbf{y}\in\Delta_m} \mathbf{x}_t^\top P\mathbf{y} \leq \max_{\mathbf{x}\in\Delta_n} \min_{\mathbf{y}\in\Delta_m} \mathbf{x}^\top P\mathbf{y} = v_{\text{maxmin}}.$$

We set the payoff vector $\mathbf{p}_t = P\mathbf{y}_t \in [0,1]^n$; and note that the above inequalities show that $\mathbf{x}_t\cdot\mathbf{p}_t \leq v_{\text{minmax}}$ for all t. Hence,

$$\sum_{t=1}^{T} \mathbf{x}_t\cdot\mathbf{p}_t \leq T v_{\text{maxmin}}. \tag{10.5}$$

On the other hand, we have,

$$\begin{aligned}\max_{\mathbf{x}\in\Delta^n} \sum_{t=1}^{T} \mathbf{x}\cdot\mathbf{p}_t &= T\cdot \max_{\mathbf{x}\in\Delta_n} \mathbf{x}^\top P\left(\frac{1}{T}\sum_{t=1}^{T}\mathbf{y}_t\right)\\ &\geq T\cdot \min_{\mathbf{y}\in\Delta_m}\max_{\mathbf{x}\in\Delta_n} \mathbf{x}^\top P\mathbf{y} = T v_{\text{minmax}}.\end{aligned} \tag{10.6}$$

Combining Eqns. (10.4), (10.5) and (10.6), we get that,

$$v_{\text{minmax}} - v_{\text{maxmin}} \leq \frac{1}{T}\cdot \text{Regret}(\mathsf{MWUA}) \leq 2\sqrt{\frac{\ln n}{T}}.$$

Since the above holds for all $T \geq 1$, it must be that $v_{\text{minmax}} \leq v_{\text{maxmin}}$, which combined with Eq. (10.3), completes the proof of the theorem.

Appendix A

Inequalities from Probability Theory

It is assumed that the reader has sufficient familiarity with the basics of the theory of probability.

A.1 The Union Bound

This is an elementary inequality, though surprisingly powerful in several applications in learning theory and the analysis of algorithms. If $A_1, A_2, \ldots$ is a (at most countable) collection of events, then

$$\mathbb{P}\left[\bigcup_i A_i\right] \leq \sum_i \mathbb{P}(A_i). \tag{A.1}$$

This inequality is known as the *union bound* (or Boole's inequality) as it shows that the probability of the union of a collection of events can be upper-bounded by the sum of the probabilities of the individual events in the union.

A.2 Hoeffding's Inequality

Let $X_1, \ldots, X_m$ be m independent random variables taking values in the interval $[0, 1]$. Let $\overline{X} = \frac{1}{m}\sum_{i=1}^m X_i$ and let $\mu = \mathbb{E}[\overline{X}]$. Then for every $t \geq 0$,

$$\mathbb{P}\left[\left|\overline{X} - \mu\right| \geq t\right] \leq 2\exp\left(-2mt^2\right). \tag{A.2}$$

This inequality is known as the Hoeffding's inequality [29].

A.3 Chernoff Bound

Let $X_1, \ldots, X_m$ be m independent random variables taking values in the set $\{0, 1\}$. Let $\overline{X} = \sum_{i=1}^m X_i$ and let $\mu = \mathbb{E}[\overline{X}]$. Then for every $0 \leq \delta \leq 1$,

$$\mathbb{P}\left[\overline{X} \leq (1-\delta)\mu\right] \leq \exp\left(-\delta^2\mu/2\right), \tag{A.3}$$

$$\mathbb{P}\left[\overline{X} \geq (1+\delta)\mu\right] \leq \exp\left(-\delta^2\mu/3\right). \tag{A.4}$$

The above pair of inequalities are known as the Chernoff bound. These are not the tightest possible bounds that can be obtained, but will be sufficient for our purposes. The interested reader may refer to more complete works on concentration inqualities, e.g. [24, 14].

Appendix B

Elementary Inequalities

B.1 Convexity of exp

For any $x \in \mathbb{R}$, the following inequality holds,

$$1 + x \leq e^x. \tag{B.1}$$

The proof is immediate using the convexity of the exponential function.

B.2 Auxilliary Lemmas

Lemma B.1. *For any $a \geq e$, $b > 0$, for every $x \geq \max\{8, 2 + 2\log b\} a \log a$, $x \geq a \log(bx)$.*

Proof. Let $f(x) = x - a\log(bx)$. It is easy to check that $f'(x) \geq 0$ for $x \geq a$. Note that if $C = \max\{8, 2 + 2\log b\}$ and as $a \geq e$, we have $Ca \log a \geq a$. As a result, $f(x) \geq f(Ca \log a)$. Thus it suffices to show that $f(Ca \log a) \geq 0$.

This can be verified as follows:

$$\begin{aligned} f(Ca\log a) &= Ca\log a - a\log(Cab\log a) \\ &= Ca\log a - a\log C - a\log a - a\log b - a\log\log a \\ &= (2a\log a - a\log a - a\log\log a) + a((C-2)/2 - \log C) \\ &\quad + a((C-2)/2 - \log b) \\ &\geq 0. \end{aligned}$$

Above we have used that $\log\log a \leq \log a$, $\log a \geq 1$, $C \geq 2 + 2\log b$ and that for $C \geq 8$, $C \geq 2 + 2\log C$.

□

Appendix C

Notation

C.1 Basic Mathematical Notation

$\mathbb{N}$	The set of natural numbers (not including 0)
$\mathbb{Z}$	The set of integers
$\mathbb{Q}$	The set of rational numbers
$\mathbb{R}$	The set of real numbers

C.2 The PAC Learning Framework

$\mathbf{x}$	A datum or the *input* part of an example
x_i	The i^{th} co-ordinate (attribute) of example $\mathbf{x}$

Bibliography

[1] Sarah R Allen, Ryan ODonnell, and David Witmer. How to refute a random csp. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 689–708. IEEE, 2015.

[2] Noga Alon and Joel H Spencer. *The probabilistic method*. John Wiley & Sons, 2004.

[3] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[4] 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. ACM, 1991.

[5] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[6] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[7] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[8] Peter Auer, Mark Herbster, and Manfred K Warmuth. Exponentially many local minima for single neurons. *Advances in neural information processing systems*, pages 316–322, 1996.

[9] Paul W Beame, Stephen A Cook, and H James Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15(4): 994–1003, 1986.

[10] Hans-Dieter Block. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123, 1962.

[11] 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 twenty-sixth annual ACM symposium on Theory of computing*, pages 253–262. ACM, 1994.

[12] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM (JACM)*, 50(4):506–519, 2003.

[13] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Occam's razor. *Information processing letters*, 24(6):377–380, 1987.

[14] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford University Press, 2013.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[16] Nader H Bshouty and Vitaly Feldman. On using extended statistical queries to avoid membership queries. *Journal of Machine Learning Research*, 2(Feb):359–395, 2002.

[17] Sébastien Bubeck. *Convex Optimization: Algorithms and Complexity*. Foundations and Trends in Machine Learning. Now, 2015.

[18] David M Burton. *Number theory*. McGraw-Hill, 2005.

[19] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge University press, 2006.

[20] Amit Daniely. Complexity theoretic limitations on learning halfspaces. *arXiv preprint arXiv:1505.05800*, 2015.

[21] Amit Daniely and Shai Shalev-Shwartz. Complexity theoretic limitations on learning DNFs. *CoRR, abs/1404.3378*, 1(2.1):2–1, 2014.

[22] Amit Daniely, Nati Linial, and Shai Shalev-Shwartz. From average case complexity to improper learning complexity. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 441–448, New York, NY, USA, 2014. ACM.

[23] Ronald de Wolf. Philosophical applications of computational learning theory : Chomskyan innateness and occam's razor. Master's thesis, Erasmus Universiteit Rotterdam, 1997.

[24] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[25] Yoav Freund. Boosting a weak learning algorithm by majority. In *COLT*, volume 90, pages 202–216, 1990.

[26] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[27] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 464–479. IEEE, 1984.

[28] Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

[29] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. ISSN 01621459. URL http://www.jstor.org/stable/2282952.

[30] Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

[31] Michael Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM (JACM)*, 45(6):983–1006, 1998.

[32] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, 1989.

[33] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41 (1):67–95, 1994.

[34] Michael J. Kearns and Umesh K. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994.

[35] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning*, 2(4):285–318, 1988.

[36] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[37] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge University Press, 2017.

[38] A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley Interscience, 1983.

[39] Albert B Novikoff. On convergence proofs for perceptrons. Technical report, Stanford Research Institute, Menlo Park, CA, 1963.

[40] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[41] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[42] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[43] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Index

3-term-DNF, 13

AdaBoost, 40

Boole's inequality, *see* union bound
boolean circuit, 18
boolean hypercube, 7
boosting, 40

Chernoff Bound, 117
clauses, *see* disjunctions
concept class, 5
conjunctions, 9
consistent learning, 22

DCRA, *see* Discrete Cube Root Assumption
decision list, 26
DFA, *see* Discrete Finite Automata
dichotomies, 27
Discrete Cube Root Assumption, 46
Discrete Cube Root Problem, 45
Discrete Finite Automata, 56
disjunctions, 11
disjunctive normal form, *see* DNF
DNF, 7

EQ, *see* equivalence query
equivalence query, 52
error, 6
exact learning, 52
example oracle, 5

growth function, 30

Hoeffding's inequality, 117
hypothesis class, 16

improper learning, 17
instance size, 8
instance space, 5

k-CNF, 11

L* Algorithm, 59
linear threshold functions, 18, 29, 36
LTF, *see* linear threshold functions

membership query, 52
Monontone DNF, 54
MQ, *see* membership query

Noisy Oracle (RCN), 63

Occam's Razor, 21

PAC learning, 17
  Take I, 6
  Take II, 7
PAC+MQ learning, 53
parities, 25
proper learning, 17

Radon's Theorem, 37
Random Classification Noise, 63
randomised polynomial time, 13
RCN, *see* Random Classification Noise
representation scheme, 8
representation size, 8
RP, *see* randomised polynomial time

Sauer-Shelah Lemma, 30
shattering, 28
size, *see* representation size
SQ Learning, *see* Statistical Query Learning
STAT oracle, 66
Statistical Query Learning, 66

union bound, 117

VC dimension, 28

weak learning, 40