

COMPUTATIONAL LEARNING THEORY

(Lecture Notes)

$$\frac{1}{\epsilon}$$

size(c)

n

$$\frac{1}{\delta}$$

Varun Kanade

Contents

| | |
|--|------------|
| Contents | i |
| Preface | iii |
| 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 | 8 |
| 1.4 Hardness of Learning 3-term DNF | 12 |
| 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 |
| 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 | 43 |
| 5 Cryptographic Hardness of Learning | 45 |
| 5.1 The Discrete Cube Root Problem | 45 |
| 5.2 A learning problem based on DCRA | 46 |
| 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 | 53 |
| 6.3 | Learning DFA | 55 |
| 6.4 | Exercises | 59 |
| A | Inequalities from Probability Theory | 61 |
| A.1 | The Union Bound | 61 |
| A.2 | Hoeffding's Inequality | 61 |
| A.3 | Chernoff Bound | 61 |
| B | Elementary Inequalities | 63 |
| B.1 | Convexity of \exp | 63 |
| B.2 | Auxilliary Lemmas | 63 |
| C | Notation | 65 |
| C.1 | Basic Mathematical Notation | 65 |
| C.2 | The PAC Learning Framework | 65 |
| | Bibliography | 67 |
| | Index | 69 |

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 running 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.

Chapter 1

Probably Approximately Correct Learning

Our goal in this chapter is to gradually build up the probably approximately correct (PAC) learning framework while emphasizing 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 labeled 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 $c_R : \mathbb{R}^2 \rightarrow \{+, -\}$, where $c_R(\mathbf{x}) = +$, if \mathbf{x} is inside the rectangle R and $c_R(\mathbf{x}) = -$, otherwise.¹

¹We 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).

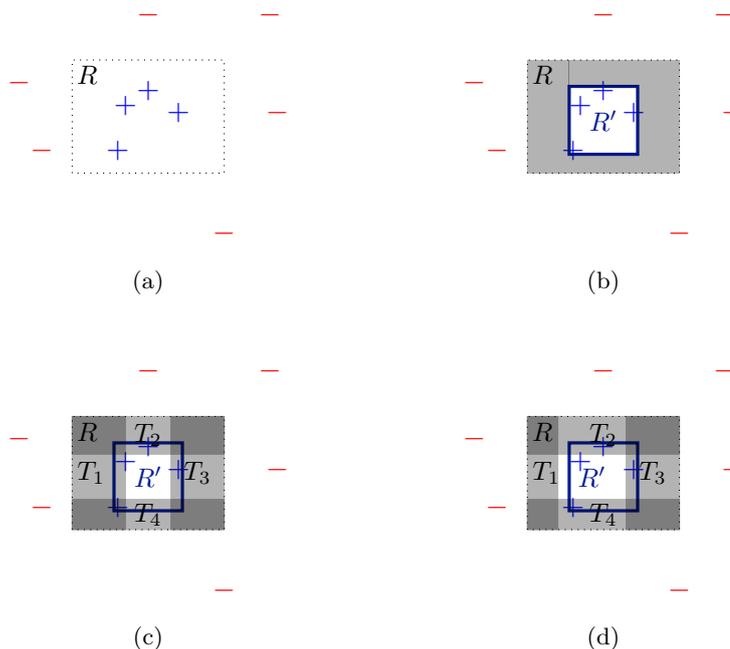


Figure 1.1: (a) Data received for the rectangle learning game. The rectangle R used to generate labels is hidden from the learning algorithm. (b) The *tightest fit* algorithm produces a rectangle R' . (c) & (d) The regions T_1, T_2, T_3 and T_4 contain $\epsilon/4$ mass each under D (for two different 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* $h_{R'} : \mathbb{R}^2 \rightarrow \{+, -\}$ 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

²For 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:

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

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

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 $\text{err}(h_{R'}; c_R, D) \leq \epsilon$. 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, $\text{err}(h_{R'}; c_R, D) \leq \epsilon$. 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

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

⁴Assuming 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 $\text{err}(h_{R'}; c_R, D) \leq \epsilon$. 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 $(1 - \frac{\epsilon}{4})^m$. By a similar argument, $\mathbb{P}[A_i] = (1 - \frac{\epsilon}{4})^m$ for $i = 1, \dots, 4$. 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 $1 - \delta$, $\text{err}(h_{R'}; c_R, D) \leq \epsilon$.

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

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:

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.

⁵So 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.

⁶We 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.

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 $c : X \rightarrow \{0, 1\}$. (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 $\text{EX}(c, D)$. We assume that a learning algorithm can query the oracle $\text{EX}(c, D)$ at unit cost and each query yields an *independent* training example.

1.2.1 PAC Learning: Take I

Let $h : X \rightarrow \{0, 1\}$ 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:

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

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

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 $\text{EX}(c, D)$ and inputs ϵ and δ , L outputs a hypothesis $h \in C$ that with probability at least $1 - \delta$ satisfies $\text{err}(h) \leq \epsilon$. The probability is over the random examples drawn from $\text{EX}(c, D)$ as well as any internal randomisation of L . The number of calls made to $\text{EX}(c, D)$ (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.

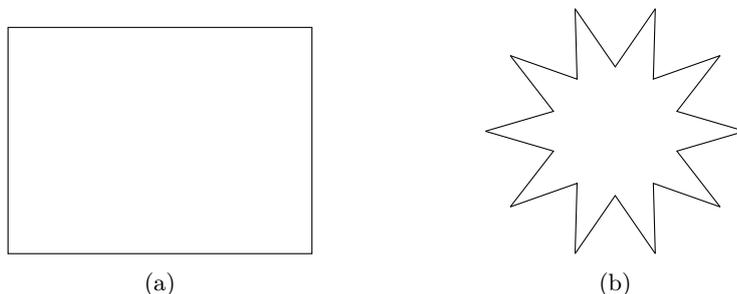
Based on our analysis of the rectangle learning game in Section 1.1, we have essentially already proved the following theorem.

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

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

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

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

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 $f : X \rightarrow \{0, 1\}$; 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. of 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.

Representation Scheme

Abstractly, a representation scheme for a concept class C is an onto function $R : \Sigma^* \rightarrow C$, 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, $\text{size} : \Sigma^* \rightarrow \mathbb{N}$, 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 $\text{size}(c)$ 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

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

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

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 $\text{EX}(c, D)$ and inputs n , $\text{size}(c)$, 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 $\text{EX}(c, D)$ as well as any internal randomization of L . The number of calls made to $\text{EX}(c, D)$ (sample complexity) must be bounded by a polynomial in n , $\text{size}(c)$, $\frac{1}{\epsilon}$ 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 , $\text{size}(c)$, $1/\epsilon$ and $1/\delta$.

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, \dots, z_n ; let $X = \bigcup_{n \geq 1} X_n$. Let CONJUNCTIONS_n denote the concept class of conjunctions over the n boolean variables z_1, \dots, 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

$$\varphi = z_1 \wedge \bar{z}_3 \wedge \bar{z}_8 \wedge z_9. \quad (1.3)$$

Formally, a conjunction over z_1, \dots, 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

¹⁰Assuming 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. This field of research is known as *sketching*.

When learning a target concept $c \in C_n$, in general, allowing the learning algorithm resources that increase with $\text{size}(c)$ will be necessary. Mostly, we will consider concept classes C_n over X_n for which every $\text{size}(c)$ 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.

as

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

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

$$\begin{aligned} \text{CONJUNCTIONS}_n &= \{\varphi_{P,N} \mid P, N \subset [n]\}, \\ \text{CONJUNCTIONS} &= \bigcup_{n \geq 1} \text{CONJUNCTIONS}_n. \end{aligned}$$

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 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 $\text{EX}(c, D)$ 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 $i = 1, \dots, n$. In other words, $y = 1$ if \mathbf{x} is a satisfying assignment of the conjunction c , and 0 otherwise.

Algorithm 1.1 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

Algorithm 1.1: CONJUNCTIONS Learner

```

1 Input:  $n, m$ , access to  $\text{EX}(c, D)$ 
2 // initialize hypothesis conjunction with all literals
3 Set  $h = z_1 \wedge \bar{z}_1 \wedge z_2 \wedge \bar{z}_2 \wedge \cdots \wedge z_n \wedge \bar{z}_n$ 
4 for  $i = 1, \dots, m$  do
5   draw  $(\mathbf{x}_i, y_i)$  from  $\text{EX}(c, D)$ 
6   if  $y_i == 1$  then // ignore negative examples
7     for  $j = 1, \dots, n$  do
8       if  $x_{i,j} = 0$  then //  $j^{\text{th}}$  bit of  $i^{\text{th}}$  instance is 0
9         Drop  $z_i$  from  $h$ 
10      else //  $j^{\text{th}}$  bit of  $i^{\text{th}}$  instance is 1
11        Drop  $\bar{z}_i$  from  $h$ 
12 Output:  $h$ 

```

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 $\text{EX}(c, D)$; 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 $p(\ell) = \mathbb{P}_{\mathbf{x} \sim D} [c(\mathbf{x}) = 1 \wedge \ell(\mathbf{x}) = 0]$; 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 $\text{EX}(c, D)$, 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 $(1 - \frac{\epsilon}{2n})^m$. 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.

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

This completes the proof. \square

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 $\text{EX}(c, D)$; 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 generalize 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 $\text{clauses}_{n,k}$ denote the set of all clauses of length exactly k on the n boolean variables z_1, \dots, z_n . We define the class of boolean functions that can be written in conjunctive normal form using clauses of length exactly k as:

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

A representation of boolean function as in the class $k\text{-CNF}$ is called a $k\text{-CNF}$ formula. There are at most $(2n)^k$ possible clauses of length k on n boolean variables and so each $k\text{-CNF}$ 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 $k\text{-CNF}$ 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, \dots, z_n ; we create new boolean variables $(z'_{\ell_1, \dots, \ell_k})$ where each ℓ_i is either some z_j or \bar{z}_j . When placed in parentheses, $(z'_{\ell_1, \dots, \ell_k})$ denotes the set of all possible $(2n)^k$ boolean variables; whereas by itself $z'_{\ell_1, \dots, \ell_k}$ denotes the specific variable corresponding to the tuple of literals (ℓ_1, \dots, ℓ_k) . The boolean variable $z'_{\ell_1, \dots, \ell_k}$ is meant to represent the clause $\ell_1 \vee \dots \vee \ell_k$. Given an assignment to the boolean variables z_1, \dots, z_n denoted by some bit-vector $\mathbf{x} \in \{0, 1\}^n$, an assignment to $(z'_{\ell_1, \dots, \ell_k})$ can be uniquely determined, by assigning the variable $z'_{\ell_1, \dots, \ell_k}$ the value 1 if and only if \mathbf{x} is a satisfying assignment of the clause $\ell_1 \vee \dots \vee \ell_k$. This yields a bit vector in $\{0, 1\}^{(2n)^k}$ that represents the assignment to all $(z'_{\ell_1, \dots, \ell_k})$. 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, \dots, z_n and *monotone conjunctions* over $(z'_{\ell_1, \dots, \ell_k})$: given a k -CNF formula φ , the literal $z'_{\ell_1, \dots, \ell_k}$ appears in the monotone conjunction $f(\varphi)$ if and only if φ contains the clause $\ell_1 \vee \dots \vee \ell_k$.¹¹ (A conjunction is *monotone* if it does not contain any *negated* literals; Algorithm 1.1 modified to start with $h = z_1 \wedge \dots \wedge z_n$ 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

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

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 $\text{EX}(c, D)$, the hypothetical example oracle $\text{EX}(g(c), f(D))$ 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.*

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

¹¹We 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.

as DNF formulae with exactly 3 terms. A term is simply a conjunction over n boolean variables z_1, \dots, z_n . Formally, the class is defined as

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

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 in fact is unlikely to exist. Formally, we'll prove the following theorem.

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

Let us first discuss the condition “unless $\text{RP} = \text{NP}$ ”. We will briefly define the class RP here, but those unfamiliar with (randomized) 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 $\sigma \in L$.

Let us see how an 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 $\text{EX}(\varphi, D)$, 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 $h \in \text{3-TERM-DNF}$, such that $\text{err}(h) \leq \frac{1}{2|S|}$. 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

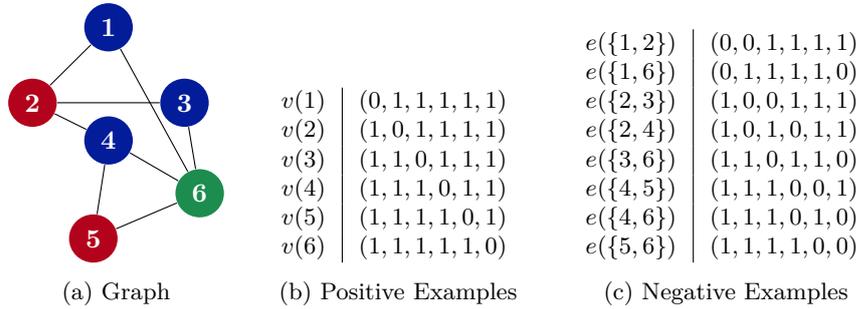


Figure 1.3: (a) A graph G along with a valid three colouring. (b) Positive examples of the sample generated using G . (c) Negative examples of the sample generated using G .

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 $h \in 3\text{-TERM-DNF}$. 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 a 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 $\text{RP} = \text{NP}$, 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. Let z_1, \dots, 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 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 any 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 $\text{RP} \neq \text{NP}$, the class 3-TERM-DNF is not *efficiently* PAC (take II) learnable. Let us recall the distributive law of boolean operations

$$(a \wedge b) \vee (c \wedge d) \equiv (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d). \quad (1.5)$$

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

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

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

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

What this suggests is that if our goal is simply to *predict* as well as the target concept $\varphi \in \text{3-TERM-DNF}$, 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 of 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 consider Exercise 2.4.) The requirement we add is that the hypothesis class H be *polynomially evaluable*.

Definition 1.7 – Polynomially Evaluable Hypothesis Class. *A hypothesis class H is polynomially evaluable 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 $\text{size}(h)$.*

In words, the requirement that H be polynomially evaluable demands that given the description of the “program” encoding the prediction rule, h , and an instance, \mathbf{x} , we should be able to evaluate $h(\mathbf{x})$ in a reasonable amount of time. Here reasonable means polynomial in the input, i.e. $\text{size}(h)$ 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*

¹²The 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. This will be discussed in greater detail in Chapter 2.

is PAC learnable using 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 $\text{EX}(c, D)$ 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 $\text{EX}(c, D)$ as well as any internal randomization of L . The number of calls made to $\text{EX}(c, D)$ (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 learnable using H , if the running time of L is polynomial in n , $\text{size}(c)$, $1/\epsilon$ and $1/\delta$.

Some comments regarding the definition of PAC Learning

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

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

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 is 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 $f : \{0,1\}^n \rightarrow \{0,1\}$ be the parity function on the n bits, i.e. $f = z_1 \oplus z_2 \oplus \dots \oplus z_n$. 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 a directed acyclic graph with exactly n source nodes and 1 sink node. The source nodes contain the inputs z_1, \dots, 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 $\text{EX}(c, D)$ and inputs ϵ and $\text{size}(c)$, runs in time polynomial in n , $\text{size}(c)$ and $1/\epsilon$, and outputs a polynomially evaluable hypothesis $h \in H_n$, that with probability at least $3/4$ satisfies $\text{err}(h) \leq \epsilon$. 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 $w \in \{0,1\}^n$ and $k \in \mathbb{N}$, $f_{w,k} : X_n \rightarrow \{0,1\}$ is a boolean threshold function defined as follows:

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

Define the concept class of threshold functions as

$$\begin{aligned} \text{THRESHOLDS}_n &= \{f_{w,k} \mid w \in \{0,1\}^n, 0 \leq k \leq n\}, \\ \text{THRESHOLDS} &= \bigcup_{n \geq 1} \text{THRESHOLDS}_n. \end{aligned}$$

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

- 1.5. Let $X_n = \{0, 1\}^n$ be the instance space. A parity function, χ_S , over X_n is defined by some subset $S \subseteq \{1, \dots, n\}$, and takes the value 1 if and odd number of the input literals in the set $\{x_i \mid i \in S\}$ are 1 and 0 otherwise. For example, if $S = \{1, 3, 4\}$, then the function $\chi_S = x_1 \oplus x_3 \oplus x_4$ computes the parity on the subset $\{x_1, x_3, x_4\}$. 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.

1.8 Chapter Notes

Material in this lecture is almost entirely adopted from Kearns and Vazirani [21, Chap. 1]. The original PAC learning framework was introduced in a seminal paper by Valiant [24].

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 $RP \neq NP$ conjecture and other conjectures from cryptography. The classic text by Papadimitriou [22], and the more recent book by Arora and Barak [5], 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. data drawn from the target distribution D . Here we consider the following question: Given $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)$, where $\mathbf{x}_i \in X_n$ and $y_i \in \{0, 1\}$, can we find some hypothesis, $h : X_n \rightarrow \{0, 1\}$ that is consistent with the observed data, i.e. for all i , $h(x_i) = y_i$.¹

If there is no restriction on the output hypothesis, then this can be simply achieved by *memorizing* 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

¹In 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 $y_i \neq y_j$.

²This 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 [11, 18, 16].

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, $(\mathbf{x}_1, c(\mathbf{x}_1)), (\mathbf{x}_2, c(\mathbf{x}_2)), \dots, (\mathbf{x}_m, c(\mathbf{x}_m))$, where each $\mathbf{x}_i \in X_n$, L outputs $h \in H_n$ such that for $i = 1, \dots, m$, $h(\mathbf{x}_i) = c(\mathbf{x}_i)$. We say that L is an efficient consistent learner if the running time of L is polynomial in n , $\text{size}(c)$ 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, where 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 .*

³Starting 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 $h : X \rightarrow Y'$, where typically $Y \subseteq Y'$. There is a loss function, $\ell : Y' \times Y \rightarrow \mathbb{R}^+$ 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

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

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

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

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 Minimization (ERM) principle suggests that a learning algorithm should pick a hypothesis $h \in H$ that minimizes 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 $\text{EX}(c, D)$, such that,

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

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

If furthermore, L is an efficient consistent learner, $\log |H_n|$ is polynomial in n and $\text{size}(c)$, 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 $\text{err}(h) \geq \epsilon$. Let A_h be the event that

m independent examples drawn from $\text{EX}(c, D)$ are all consistent with h , i.e. $h(\mathbf{x}_i) = c(\mathbf{x}_i)$, for $i = 1, \dots, m$. 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 $\text{EX}(c, D)$. However, any hypothesis that is not “bad”, satisfies $\text{err}(h) \leq \epsilon$ as required. \square

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 $\text{poly}(n, \text{size}(c)) \cdot m^\beta$ 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 proofs for this version appears in the book by Kearns and Vazirani [21, 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 $\text{EX}(c, D)$ was at least $\frac{2n}{\epsilon} (\log(2n) + \log \frac{1}{\delta})$, the output hypothesis with high probability has error at most ϵ . Theorem 2.2 shows that in fact even a sample of size $\frac{1}{\epsilon} (n \log 3 + \log \frac{1}{\delta})$ 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 $|\text{3-TERM-DNF}|$ and $|\text{3-CNF}|$. 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, $|\text{3-TERM-DNF}| \leq 2^{6n}$.

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, $|\text{3-CNF}| \leq 2^{8n^3}$. It is also not hard to show that $|\text{3-CNF}| \geq 2^{\kappa n^3}$ for some universal constant $\kappa > 0$. Thus, it is the case that $\log |\text{3-CNF}| = \Omega(n^3)$. 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 $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)$, 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 , $\text{size}(c)$, $\frac{1}{\delta}$ 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 $S \subseteq \{1, \dots, n\}$, and takes the value 1 if and odd number of the input literals in the set $\{x_i \mid i \in S\}$ are 1 and 0 otherwise. For example, if $S = \{1, 3, 4\}$, then the function $\chi_S = x_1 \oplus x_3 \oplus x_4$ computes the parity on the subset $\{x_1, x_3, x_4\}$. 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

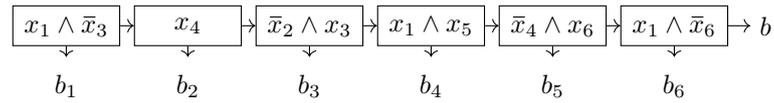
⁴At the very least, this is the lower bound we get if we apply Theorem 2.2. We will see shortly that in fact this is a lower bound on sample complexity for learning 3-CNF, no matter what algorithm is used.

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 x_1, \dots, x_n , is defined by an ordered list

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

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 $a \in \{0, 1\}^n$ the value $L(a)$ is defined to be b_j , where j is the smallest index satisfying $t_j(a) = 1$ and $L(a) = b$ 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.

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 $c : X \rightarrow \{0, 1\}$, we can consider the restriction of c to S , $c|_S : S \rightarrow \{0, 1\}$, where $c|_S(x) = c(x)$ for $x \in S$. We define the following:

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

The set $\Pi_C(S)$ is the class of distinct restrictions of concepts in C defined by the set S . Alternatively, if $S = \{x_1, \dots, x_m\}$, we can associate each element of $\Pi_C(S)$ with the function values at each of the m points,

$$\Pi_C(S) = \{(c(x_1), \dots, c(x_m)) \mid c \in C\} \quad (3.2)$$

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

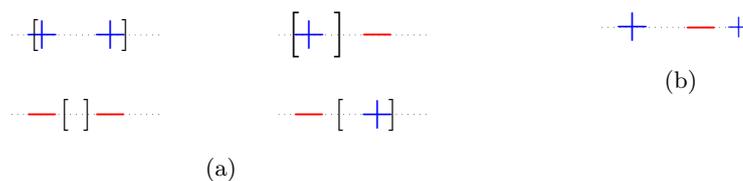


Figure 3.1: (a) All possible dichotomies on 2 points can be realised using intervals. (b) A dichotomy on three points that cannot be realised by intervals.

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 $\text{VCD}(C)$ is the cardinality d of the largest finite set S shattered by C . If C shatters arbitrarily large finite sets, then $\text{VCD}(C) = \infty$.

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 $c_{a,b} : \mathbb{R} \rightarrow \{0, 1\}$ is defined as $c_{a,b}(x) = 1$ if $x \in [a, b]$ and 0 otherwise. What is $\text{VCD}(C)$? It is easy to see that any subset $S \subset \mathbb{R}$ 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, $\text{VCD}(C) = 2$.

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

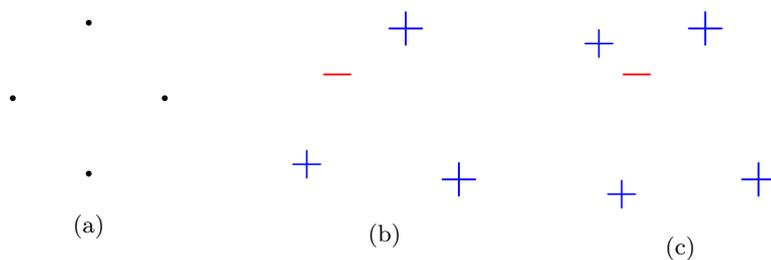


Figure 3.2: (a) A set of 4 points on which all dichotomies can be realised using rectangles. (b) A set of 4 points with a dichotomy that cannot be realised by rectangles. (c) Any set of 5 points always has a dichotomy that cannot be realised using rectangles.

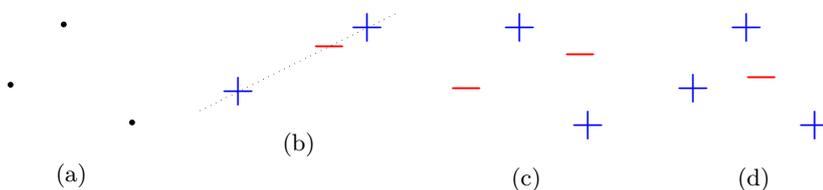


Figure 3.3: (a) A set of 3 points shattered by linear threshold functions. (b) A dichotomy on a set of 3 points that cannot be realised by linear threshold functions. (c) & (d) No set of 4 points can be shattered by linear threshold functions.

of a certain size that is shattered. It is possible to show that no set of size 5 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 ?? 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 realized 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 $m \leq \text{VCD}(C)$. We are interested in understanding the maximum possible growth of the number of dichotomies for $m \geq \text{VCD}(C)$. We will show that this growth can be bounded by a polynomial in m of degree $\text{VCD}(d)$, 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 Sauer’s Lemma.

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

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

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

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

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:

$$\Phi(m, d) = \Phi(m-1, d) + \Phi(m-1, d-1), \quad (3.4)$$

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

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 $\binom{m}{r} = \binom{m-1}{r} + \binom{m-1}{r-1}$,

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

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 $m \leq d$, $\Phi(m, d) = 2^m$.) \square

Finally Lemma 3.7 together with Lemma 3.6 completes the proof of Sauer's Lemma (Lemma 3.4).

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

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,

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

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

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

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

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

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 suffices to show that $\text{VCD}(C') \leq d-1$, 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. \square

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 $\text{VCD}(H) = d$, 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 $\text{EX}(c, D)$, 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 $h \in H$, $\text{err}(h; c, D) = \mathbb{P}_{x \sim D} [(c \oplus h)(x) = 1]$.

Let $H \oplus c = \{h \oplus c \mid h \in H\}$. It is easy to show that $\text{VCD}(H \oplus c) = \text{VCD}(H)$ (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 $\mathbb{P}_{x \sim D} [(h \oplus c)(x) = 1] \geq \epsilon$, there exists some $x \in S$, such that $(h \oplus c)(x) = 1$.

We observe that a hypothesis, h , for which $\mathbb{P}_{x \sim D} [(h \oplus c)(x) = 1] \geq \epsilon$ is problematic, as $\text{err}(h; c, D) \geq \epsilon$. 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 $\text{EX}(c, D)$ 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 symmetrizing. 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 $\text{EX}(c, D)$. 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 $\tilde{h} \in H$ such that $(\tilde{h} \oplus c)(x) = 0$ for all $x \in S_1$ and $\mathbb{P}_{x \sim D} [(\tilde{h} \oplus c)(x) = 1] \geq \epsilon$. Fix such a $\tilde{h} \in H$ and draw a second sample S_2 of size m . Now, let us obtain a *lower*

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

bound on the number of elements x in S_2 that satisfy $(\tilde{h} \oplus c)(x) = 1$. Let X_i denote the random variable that takes value 1 if the i^{th} element of S_2 satisfies $(\tilde{h} \oplus c)(x) = 1$ 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

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

Provided $\epsilon m \geq 16$ (which our final bound will ensure), the probability that $|\{x \in S_2 \mid (\tilde{h} \oplus c)(x) = 1\}| \geq \epsilon m/2$ 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 $\text{EX}(c, D)$, there exists a $\tilde{h} \oplus c \in \Pi_{H \oplus c}(S)$, such that $|\{x \in S \mid (\tilde{h} \oplus c)(x) = 1\}| \geq \epsilon m/2$ and $(\tilde{h} \oplus c)(x) = 0$ for all $x \in S_1$. We have slightly abused notation and used $\tilde{h} \oplus c$ 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 $(\tilde{h} \oplus c) \in \Pi_{H \oplus c}(S)$ such that $|\{x \in S_1 \mid \tilde{h} \oplus c(x) = 1\}| = 0$ and $|\{x \in S_2 \mid \tilde{c}(x) = 1\}| \geq \epsilon m/2$ 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

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

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 $\text{EX}(c, D)$. For any fixed $h \oplus c \in \Pi_{H \oplus c}^\epsilon(S)$, let $B_{h \oplus c}|S$ denote the event (conditioned on S) that $|\{x \in S_1 \mid (h \oplus c)(x) = 1\}| = 0$. 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. Thus, we have We can then bound the probability of the

event B as follows:

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

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. \square

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 $\text{VCD}(C) \geq d$, 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 randomized, however, we know that there is an upper bound $m = \max\{\frac{d-1}{32\epsilon}, \frac{1}{4\epsilon} \log \frac{1}{\delta}\}$ on the number of examples it uses. Thus, if the learning algorithm outputs 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 $T = \{x_1, x_2, \dots, x_d\}$, and let D be a distribution defined as follows: $D(x_1) = 1 - 8\epsilon$, and $D(x_j) = 8\epsilon/(d-1)$ for $j = 2, \dots, d$. 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 $T \setminus \{x_1\} = \{x_2, \dots, x_d\}$. 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 $|S| = Z + 1 < \frac{d+1}{2}$. 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

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

²The 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,

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

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

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

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 $\text{VCD}(C) \geq 2$. Now define a distribution D over T such that $D(x_1) = 1 - 4\epsilon$ and $D(x_2) = 4\epsilon$. 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, the same argument as above shows that with probability $> 1/4$ the error of h output by L will be $> \epsilon$. That completes the proof. \square

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

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

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 $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$, 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, \dots, w_n . The objective function is constant, so we are in fact only looking for a feasible point. The constraints are given by:

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

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

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

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

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 $\text{VCD}(C) = d$, 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 $c_S : X \rightarrow \{0, 1\}$, 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 $\text{VCD}(H \oplus c) = \text{VCD}(H)$.

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.

¹What 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.

²If 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, $\text{size}(c)$.

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 $\text{EX}(c, D)$ and inputs $\text{size}(c)$, δ and γ , outputs $h \in H_n$ that with probability at least $1 - \delta$, satisfies, $\text{err}(h) \leq \frac{1}{2} - \gamma(n, \text{size}(c))$.

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

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 [19], 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 [13] and Schapire [23] 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 [14] 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

³If 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

```

1 Inputs:
2 Training data  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$  drawn from  $\text{EX}(c, D)$ ,
3  $T$  (#iterations),  $\delta$  (confidence parameter)
4 Weak learning algorithm  $\text{WEAKLEARN}(D, \delta)$ 
5 // uniform initial distribution over training data
6 Set  $D_1(i) = 1/m$ 
7 for  $t = 1, \dots, T$  do
8   // examples drawn from  $D_t$  are passed to  $\text{WEAKLEARN}$ 
9   Obtain  $h_t \leftarrow \text{WEAKLEARN}(D_t, \delta/T)$ 
10  Set  $\epsilon_t = \mathbb{P}_{(\mathbf{x}, y) \sim D_t} [h_t(\mathbf{x}) \neq y]$  //  $\epsilon_t \leq 1/2 - \gamma$ , w.p.  $\geq 1 - \frac{\delta}{T}$ 
11  Set  $\alpha_t = \frac{1}{2} \log \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$ 
12  //  $Z_{t+1}$  is the normalizing constant
13  Update  $D_{t+1}(i) = D_t(i) \cdot \exp(-\alpha_t y_i h_t(\mathbf{x}_i)) / Z_{t+1}$ 
14 Set  $\tilde{h} = \sum_{i=t}^T \alpha_t h_t$ 
15 Output: hypothesis  $\hat{h} : X \rightarrow \{-1, 1\}$ , where  $\hat{h}(\mathbf{x}) = \text{sign}(\tilde{h}(\mathbf{x}))$ 

```

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 $\text{EX}(c, D)$. 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 $\text{sign} : \mathbb{R} \rightarrow \{-1, 1\}$, 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 $\mathbf{1}(\cdot)$ be the indicator of the predicate inside the parantheses, which takes the value 1 if the predicate is true and 0 otherwise. Observe that $\mathbf{1}(\text{sign}(\tilde{h}(\mathbf{x})) \neq y) \leq e^{-y\tilde{h}(\mathbf{x})}$ for $y \in \{-1, 1\}$.

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

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

We introduce some additional notation. Let $\tilde{h}_t = \sum_{s=t}^T \alpha_s h_s$ be the weighted sum of the hypotheses returned in iterations t through T ; and thus, $\tilde{h}_t = \alpha_t h_t + \tilde{h}_{t+1}$. 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:

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

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

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

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

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

We use: in (4.3) $\tilde{h} = \tilde{h}_1$, in (4.4) $\tilde{h}_1 = \alpha_1 h_1 + \tilde{h}_2$, 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) $\tilde{h}_2 = \alpha_2 h_2 + \tilde{h}_3$, 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,

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

And thus,

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

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

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

Now, by the guarantee on the weak learning algorithm, $\gamma_t \geq \gamma$ for $t = 1, \dots, T$. 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 $1/m$). \square

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 normalizing 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 minimize 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 . Exercise 4.1. asks you to further explore some of the behaviour of this algorithm.

4.2.1 Bounding the Generalization Error

One way to bound the generalization 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 $\text{VCD}(H) = d$. Denote by $\text{THRESHOLDS}_k(H)$ the class of functions given by

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

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

The proof of Lemma 4.3 is left as Exercise 4.2.. Together with Theorem 3.8 we can obtain a generalization bound on the error of the hypothesis output by Adaboost.

4.3 Exercises

4.1. Consider the AdaBoost algorithm described in Algorithm 4.1.

- Show that the error of h_t with respect to the distribution D_{t+1} is exactly $1/2$.
- What is the maximum possible value of $D_t(i)$ for some $1 \leq t \leq T$ and $1 \leq i \leq m$?
- Fix some example, say i , let t_i be the first iteration such that $h_{t_i}(x_i) = y_i$. How large can t_i be?

4.2. Prove Lemma 4.3.

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

$$H_n = \{0, 1, x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}.$$

The hypothesis class contains $2n + 2$ functions. The functions “0” and “1” are constant and predict 0 and 1 on all instances in X_n . The function

⁴This 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 generalizes even without such a condition on the weak learning algorithm.

" x_i " evaluates to 1 on any $a \in \{0, 1\}^n$ satisfying $a_i = 1$ and 0 otherwise. Likewise, the function " \bar{x}_i " evaluates to 1 on any $a \in \{0, 1\}^n$ satisfying $a_i = 0$ 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 algorithms in the PAC learning framework for concept classes such as those of 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 *learnability* exactly, 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 NP-hard. In the case of 3-term DNF formulae, we can avoid this NP-hardness by identifying 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 lecture, 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-learned, even in the case of *improper* learning, where the output hypothesis is allowed to come from any polynomially evaluable 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 $f_N : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$ defined as $f_N(y) \equiv y^3 \pmod{N}$. 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,

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

The last equality follows as $y^{\varphi(N)} \equiv 1 \pmod{N}$ for all $y \in \mathbb{Z}_N^*$.

Definition 5.1 – Discrete Cube Root Problem. *Let two n -bit primes p and q 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 $y^3 \equiv x \pmod{N}$.*

Observe that if we can factorise N , the discrete cuberoot problem is easy to solve. We can simply obtain $\varphi(N)$ and then find d such that $3d \equiv 1 \pmod{\varphi(N)}$ 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 randomized) 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 randomly from \mathbb{Z}_N^* , outputs $y \in \mathbb{Z}_N^*$ that with probability at least $1/P(n)$ satisfies $y^3 \equiv x \pmod{N}$. 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 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 cuberoot assumption, this cannot be decrypted, except if one has access to d , such that $3d \equiv 1 \pmod{\varphi(N)}$. The integer d is the private key.

5.2 A learning problem based on DCRA

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

$$S = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\},$$

¹Note 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.

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

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 cuberoot 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 $x_i \equiv y_i^3 \pmod N$ 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 $y_i^3 \equiv x_i \pmod N$, can't help to find $h : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$, that satisfies $\mathbb{P}[(h(x))^3 \not\equiv x \pmod N] \leq \epsilon$, 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 evaluable) hypothesis h , such that $\mathbb{P}_{x \sim D} [f_{N,i}^{-1}(x) \neq h(x)] \leq \frac{\epsilon}{2n}$. 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 $x^d \pmod N$. 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, $x \pmod N, x^2 \pmod N, x^4 \pmod N, x^8 \pmod N, \dots, x^{2^{\lfloor \log \varphi(N) \rfloor}} \pmod N$, i.e. we compute $x^{2^i} \pmod N$ for $i = 0, 1, \dots, \lfloor \log \varphi(N) \rfloor$. To obtain $x^d \pmod N$, we simply take the product of the terms $x^{2^i} \pmod N$ 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 $i = 1, \dots, 2n$. 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 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 weaker 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 $x^{2^i} \bmod N$ for each value of $i = 0, 1, \dots, \lfloor \log \varphi(N) \rfloor$; the reason being that $x^{2^{i+1}} \bmod N$ can only be computed after $x^{2^i} \bmod N$ 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 $(x \bmod N, x^2 \bmod N, x^4 \bmod N, \dots, x^{2^{\lfloor \log \varphi(N) \rfloor}} \bmod N)$, 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, $x^{2^i} \bmod N, i = 1, 2, \dots, \lfloor \log \varphi(N) \rfloor$. It is relatively straightforward to show that there exists a circuit of depth $O((\log n)^2)$ that when given as input $(x \bmod N, x^2 \bmod N, \dots, x^{2^{\lfloor \log \varphi(N) \rfloor}} \bmod N)$ outputs $f_N^{-1}(x)$. To show that there is a circuit of depth $O(\log n)$ requires more work using the techniques of Beame et al. [6]. 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 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 [21, 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 [20]. The proof that the class of polynomial-size circuits cannot be PAC-learned if one-way functions exist, even when membership queries are allowed (cf. Chap. 6.1), first appeared in the work of Goldreich et al. [15]. 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 of 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 [10, 9, 8]. 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. [10] 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 how much computational power is necessary to run a learning algorithm. 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 evaluable hypothesis. For proper learning, we have already established that even relatively simple concept classes such as 3-TERM-DNF are hard to PAC-learn unless $\text{RP} = \text{NP}$. 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 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

¹All 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, MQ(c). *A membership (or value) query oracle for a concept $c : X \rightarrow \{0, 1\}$, 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) $h : X \rightarrow \{0, 1\}$ 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, EQ(c). *An equivalence oracle for a concept $c : X \rightarrow \{0, 1\}$, EQ(c), when queried with a representation of a hypothesis, $h : X \rightarrow \{0, 1\}$, 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 amount of computational time spent. 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.

²The 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 MQ + EQ. We say that a concept class C is efficiently exactly learnable from membership and equivalence queries, if there exists a polynomially evaluable 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 $\text{MQ}(c)$ and $\text{EQ}(c)$, and input $\text{size}(c)$, 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 require that every query made by L to $\text{EQ}(c)$ 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 $\text{MQ}(c)$ oracle;³ it is harder to expect humans to be able to simulate an $\text{EQ}(c)$ oracle. The latter however is primarily defined for mathematical convenience. Exercise 6.2 shows how any algorithm designed in the *exact learning* with MQ + EQ framework allows one to design a PAC learning algorithm provided the algorithm has access to the membership oracle $\text{MQ}(c)$. 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 $\text{EQ}(c)$ is not necessary.

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 recognizable 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.

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, $\text{MQ}(c)$, does not help for learning DNF formulae under certain plausible assumptions used in cryptography.

³Recently this has become easier using various online labelling services that allow interaction with humans, a catchall term for which is “crowdsourcing”. 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.

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”.

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

```

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

```

⁴There have been recent attempts to establish the hardness of learning DNF formulae based on assumptions from average case complexity theory. See the discussion in Section 5.3 for some discussion about this.

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

Theorem 6.4. *The class MONOTONE-DNF is exactly learnable using MQ + EQ.*

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 $\varphi(x) = 1$ at any stage in the algorithm, then $c(x) = 1$. 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, $\varphi(x) = 1$ implies $c(x) = 1$, any counterexample a that establishes that $\varphi \neq c$ must be such that $c(a) = 1$ and $\varphi(a) = 0$. Let $P = \{j \mid T_j(a) = 1\}$ denote the indices of terms in the minimal representation of c that are satisfied by a . As $c(a) = 1$, 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 $T_j(a) = 1$, where a 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 a will never be modified to be such that $c(a) = 0$. 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 x_i is a literal in T_j , but not in $T_{j'}$. Setting $a_i = 0$ 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 a that could have been set to 0 and still have allowed a 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 $\varphi(x) = 1$ implies $c(x) = 1$. 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 queries to EQ(c), and at most $n \cdot s$ queries to MQ(c). Clearly, the running time of the algorithm is polynomial in n and s . \square

6.3 Learning DFA

In this section, we will consider the problem of learning Discrete 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 recognize 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

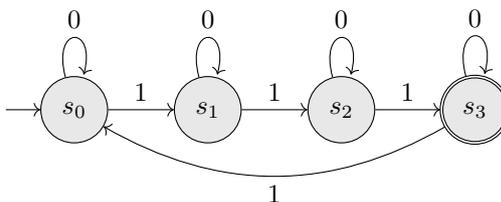


Figure 6.1: A DFA that accepts words in $\{0, 1\}^*$ in which the number of times 1 appears is 3 modulo 4.

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, $\delta : Q \times \Sigma \rightarrow Q$ 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) outdegree 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 recognized by some DFA A . Let $|A|$ denote the number of states of A and suppose that A is the smallest such DFA recognizing 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 $T \subseteq \Sigma^*$.

Definition 6.5. *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 what it means for a pair (Q, T) or access and test words to be separable and closed.

Definition 6.6. 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 6.7. 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 $qa \equiv_T q'$. 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 the Algorithm 6.2.

Lemma 6.8. Suppose that A is an automaton with the minimum number of states recognizing 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. \square

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 6.9. 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. \square

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 6.10. *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 unique, so we use the pair (q_i, i) to distinguish possible multiple occurrences of the same word. 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 $i \in \{0, \dots, n-1\}$, 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. \square

6.3.4 The L^* Algorithm

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 recognized by the automaton in Figure 6.1.

The main result we prove here is the following.

Theorem 6.11. *Let $L \subseteq \Sigma^*$ be a regular language over Σ and let A be an automaton with the fewest states recognizing L . Algorithm 6.2 when given access to a membership oracle, MQ, and equivalence oracle, EQ, for L , outputs a hypothesis automaton that recognizes L . The running time of the algorithm is polynomial in $|A|$, $|\Sigma|$ 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 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 6.8

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

```

1 Let  $Q \leftarrow \{\varepsilon\}, T \leftarrow \{\varepsilon\}$ 
2 while true do
3   while  $(Q, T)$  not separable and closed do
4     Use Lemma 6.9 to obtain  $q'$ 
5      $Q \leftarrow Q \cup \{q'\}$ 
6   //  $(Q, T)$  now separable and closed
7   Let  $A_{(Q,T)}$  be hypothesis automaton
8   Let  $\text{ans} \leftarrow \text{EQ}(A_{(Q,T)})$ 
9   if  $\text{ans} = \text{equivalent}$  then
10    break
11  else
12    Let  $w \in \Sigma^*$  be the counterexample
13    Use Lemma 6.10 to obtain  $q', t'$ 
14     $Q \leftarrow Q \cup \{q'\}$ 
15     $T \leftarrow T \cup \{t'\}$ 
16 Output: Hypothesis  $A_{(Q,T)}$ 

```

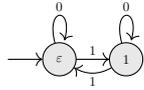
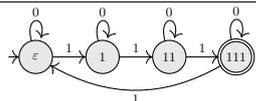
| # | State | Action |
|---|--|--|
| 1 | $Q = \{\varepsilon\}, T = \{\varepsilon\}$ |  |
| 2 | Receive counterexample 111 | Set $Q = Q \cup \{1\}, T = T \cup \{11\}$ |
| 3 | $Q = \{\varepsilon, 1\}, T = \{\varepsilon, 11\}$ |  |
| 4 | Receive counterexample 111 | Set $Q = Q \cup \{11\}, T = T \cup \{1\}$ |
| 5 | $Q = \{\varepsilon, 1, 11\}, T = \{\varepsilon, 1, 11\}$ Make (Q, T) closed | Set $Q = Q \cup \{1111\}, T = T \cup \{1\}$ |
| 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

shows that the size of $|Q| \leq |A|$, as a result the algorithm must terminate correctly with automaton $A_{(Q,T)}$ exactly recognizing the target language L .

The runtime argument follows from Lemma 6.9 and Lemma 6.10. \square

6.4 Exercises

- 6.1 The class $\text{MONOTONE-DNF}_{n,s}$ 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,

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

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 MONOTONE-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, $\text{EX}(c, D)$, 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 $\text{EX}(c, D)$ and the membership oracle for c and with inputs ϵ , δ and $\text{size}(c)$, outputs h that with probability at least $1 - \delta$ satisfies $\text{err}(h) \leq \epsilon$. The running time of L should be polynomial in n , $\text{size}(c)$, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ and the h should be from a hypothesis class H that is polynomially evaluable.

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, \dots is a (at most countable) collection of events, then

$$\mathbb{P} \left[\bigcup_i A_i \right] \leq \sum_i \mathbb{P}(A_i). \quad (\text{A.1})$$

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, \dots, X_m be m independent random variables taking values in the interval $[0, 1]$. Let $\bar{X} = \frac{1}{m} \sum_{i=1}^m X_i$ and let $\mu = \mathbb{E}[\bar{X}]$. Then for every $t \geq 0$,

$$\mathbb{P} \left[\left| \bar{X} - \mu \right| \geq t \right] \leq 2 \exp \left(-2mt^2 \right). \quad (\text{A.2})$$

This inequality is known as the Hoeffding's inequality [17].

A.3 Chernoff Bound

Let X_1, \dots, X_m be m independent random variables taking values in the interval $\{0, 1\}$. Let $\bar{X} = \sum_{i=1}^m X_i$ and let $\mu = \mathbb{E}[\bar{X}]$. Then for every $0 \leq \delta \leq 1$,

$$\mathbb{P} \left[\bar{X} \leq (1 - \delta)\mu \right] \leq \exp \left(-\delta^2 \mu / 2 \right), \quad (\text{A.3})$$

$$\mathbb{P} \left[\bar{X} \geq (1 + \delta)\mu \right] \leq \exp \left(-\delta^2 \mu / 3 \right). \quad (\text{A.4})$$

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 inequalities, e.g. [12, 7].

Appendix B

Elementary Inequalities

B.1 Convexity of exp

For any $x \in \mathbb{R}$, the following inequality holds,

$$1 + x \leq e^x. \quad (\text{B.1})$$

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 O'Donnell, 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] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- [6] 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.
- [7] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford University Press, 2013.
- [8] Amit Daniely. Complexity theoretic limitations on learning halfspaces. *arXiv preprint arXiv:1505.05800*, 2015.
- [9] Amit Daniely and Shai Shalev-Shwartz. Complexity theoretic limitations on learning DNFs. *CoRR*, abs/1404.3378, 1(2.1):2–1, 2014.
- [10] 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.
- [11] Ronald de Wolf. Philosophical applications of computational learning theory : Chomskyan innateness and occam's razor. Master's thesis, Erasmus Universiteit Rotterdam, 1997.
- [12] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

- [13] Yoav Freund. Boosting a weak learning algorithm by majority. In *COLT*, volume 90, pages 202–216, 1990.
- [14] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.
- [15] 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.
- [16] Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.
- [17] 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>.
- [18] Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.
- [19] 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.
- [20] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.
- [21] Michael J. Kearns and Umesh K. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994.
- [22] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.
- [23] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.
- [24] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Index

- DFA, *see* Discrete Finite Automata
- 3-term-DNF, 13

- AdaBoost, 40

- Boole's inequality, *see* union bound
- boolean circuit, 18
- boolean hypercube, 7
- boosting, 40

- Chernoff Bound, 61
- clauses, *see* disjunctions
- concept class, 5
- conjunctions, 8
- consistent learning, 22

- decision list, 26
- dichotomies, 27
- Discrete Finite Automata, 56
- disjunctions, 11
- disjunctive normal form, *see* DNF
- DNF, 7

- error, 6
- example oracle, 5

- growth function, 30

- Hoeffding's inequality, 61
- hypothesis class, 16

- improper learning, 17
- instance size, 8
- instance space, 5

- k-CNF, 11

- linear threshold functions, 18, 29, 36
- LTF, *see* linear threshold functions

- Occam's Razor, 21

- PAC learning, 16
 - Take I, 5
 - Take II, 6
- parities, 19, 25
- proper learning, 17

- Radon's Theorem, 37
- randomized polynomial time, 13
- representation scheme, 7
- representation size, 7
- RP, *see* randomized polynomial time

- Sauer's Lemma, 30
- shattering, 28
- size, *see* representation size

- union bound, 61

- VC dimension, 28

- weak learning, 40