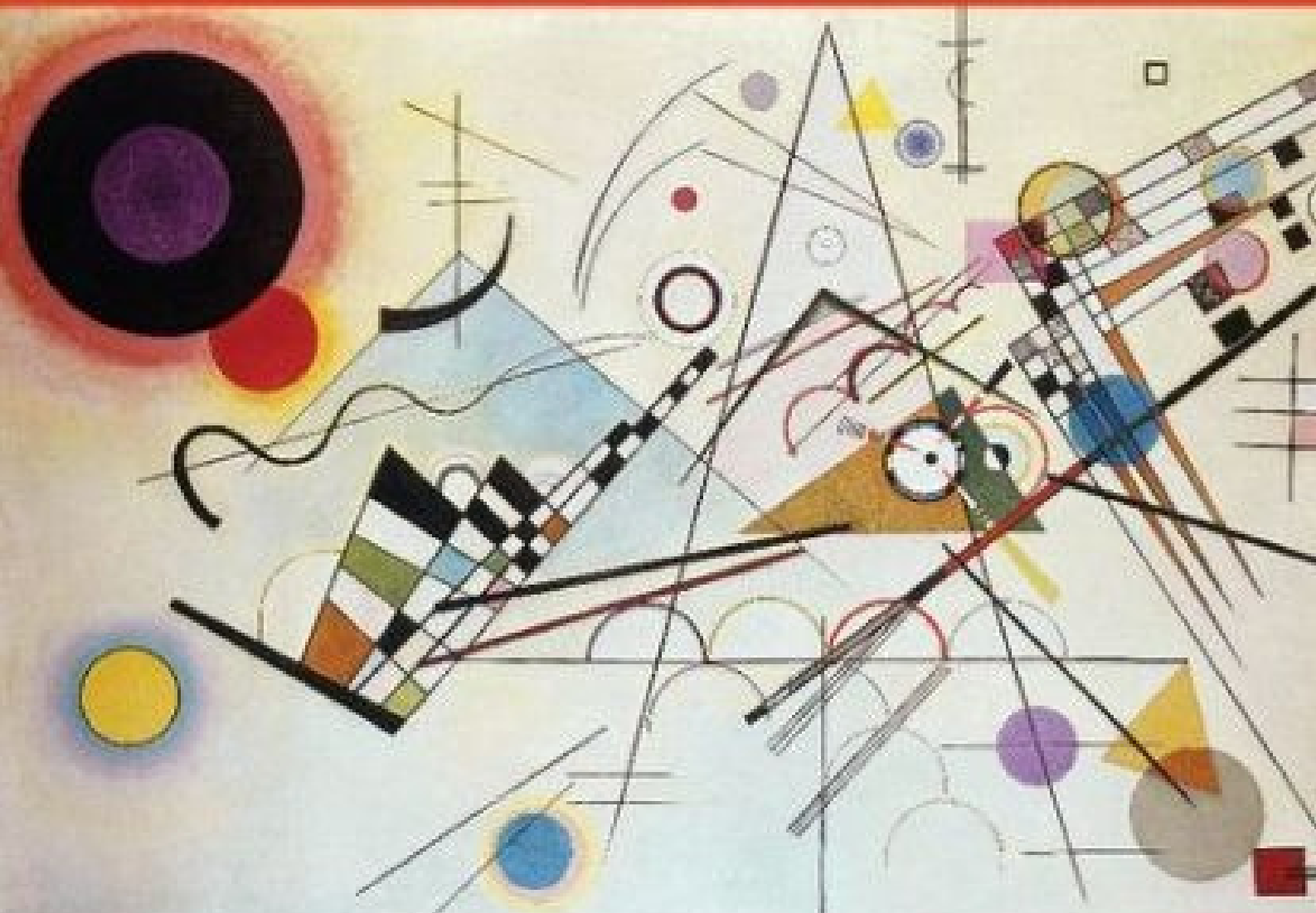
Understanding Machine Learning:
From Theory to Algorithms

Shai Shalev-Shwartz and Shai Ben-David

UNDERSTANDING MACHINE LEARNING

FROM THEORY TO ALGORITHMS



Understanding Machine Learning

Machine learning is one of the fastest growing areas of computer science, with far-reaching applications. The aim of this textbook is to introduce machine learning, and the algorithmic paradigms it offers, in a princi- pled way. The book provides an extensive theoretical account of the fundamental ideas underlying machine learning and the mathematical derivations that transform these principles into practical algorithms. Following a presentation of the basics of the field, the book covers a wide array of central topics that have not been addressed by previous textbooks. These include a discussion of the computational complexity of learning and the concepts of convexity and stability; important algorithmic paradigms including stochastic gradient descent, neural networks, and structured output learning; and emerging theoretical concepts such as the PAC-Bayes approach and compression-based bounds. Designed for an advanced undergraduate or beginning graduate course, the text makes the fundamentals and algorithms of machine learning accessible to stu- dents and nonexpert readers in statistics, computer science, mathematics, and engineering.

Shai Shalev-Shwartz is an Associate Professor at the School of Computer Science and Engineering at The Hebrew University, Israel.

Shai Ben-David is a Professor in the School of Computer Science at the University of Waterloo, Canada.

UNDERSTANDING MACHINE LEARNING

From Theory to Algorithms

Preface

The term machine learning refers to the automated detection of meaningful patterns in data. In the past couple of decades it has become a common tool in almost any task that requires information extraction from large data sets. We are surrounded by a machine learning based technology: search engines learn how to bring us the best results (while placing profitable ads), anti-spam software learns to filter our email messages, and credit card transactions are secured by a software that learns how to detect frauds. Digital cameras learn to detect faces and intelligent personal assistance applications on smart-phones learn to recognize voice commands. Cars are equipped with accident prevention systems that are built using machine learning algorithms. Machine learning is also widely used in scientific applications such as bioinformatics, medicine, and astronomy.

One common feature of all of these applications is that, in contrast to more traditional uses of computers, in these cases, due to the complexity of the patterns that need to be detected, a human programmer cannot provide an explicit, fine-detailed specification of how such tasks should be executed. Taking example from intelligent beings, many of our skills are acquired or refined through learning from our experience (rather than following explicit instructions given to us). Machine learning tools are concerned with endowing programs with the ability to "learn" and adapt.

The first goal of this book is to provide a rigorous, yet easy to follow, introduction to the main concepts underlying machine learning: What is learning? How can a machine learn? How do we quantify the resources needed to learn a given concept? Is learning always possible? Can we know if the learning process succeeded or failed?

The second goal of this book is to present several key machine learning algorithms. We chose to present algorithms that on one hand are successfully used in practice and on the other hand give a wide spectrum of different learning techniques. Additionally, we pay specific attention to algorithms appropriate for large scale learning (a.k.a. "Big Data"), since in recent years, our world has become increasingly "digitized" and the amount of data available for learning is dramatically increasing. As a result, in many applications data is plentiful and computation time is the main bottleneck. We therefore explicitly quantify both the amount of data and the amount of computation time needed to learn a given concept.

The book is divided into four parts. The first part aims at giving an initial rigorous answer to the fundamental questions of learning. We describe a generalization of Valiant's Probably Approximately Correct (PAC) learning model, which is a first solid answer to the question "what is learning?". We describe the Empirical Risk Minimization (ERM), Structural Risk Minimization (SRM), and Minimum Description Length (MDL) learning rules, which shows "how can a machine learn". We quantify the amount of data needed for learning using the ERM, SRM, and MDL rules and show how learning might fail by deriving

Contents

1 Intro duction

The subject of this book is automated learning, or, as we will more often call it, Machine Learning (ML). That is, we wish to program computers so that they can "learn" from input available to them. Roughly speaking, learning is the process of converting experience into expertise or knowledge. The input to a learning algorithm is training data, representing experience, and the output is some expertise, which usually takes the form of another computer program that can perform some task. Seeking a formal-mathematical understanding of this concept, we'll have to be more explicit about what we mean by each of the involved terms: What is the training data our programs will access? How can the process of learning be automated? How can we evaluate the success of such a process (namely, the quality of the output of a learning program)?

1.1 What Is Learning?

Let us begin by considering a couple of examples from naturally occurring animal learning. Some of the most fundamental issues in ML arise already in that context, which we are all familiar with.

Bait Shyness – Rats Learning to Avoid Poisonous Baits: When rats encounter food items with novel look or smell, they will first eat very small amounts, and subsequent feeding will depend on the flavor of the food and its physiological effect. If the food produces an ill effect, the novel food will often be associated with the illness, and subsequently, the rats will not eat it. Clearly, there is a learning mechanism in play here – the animal used past experience with some food to acquire expertise in detecting the safety of this food. If past experience with the food was negatively labeled, the animal predicts that it will also have a negative effect when encountered in the future.

Inspired by the preceding example of successful learning, let us demonstrate a typical machine learning task. Suppose we would like to program a machine that learns how to filter spam e-mails. A naive solution would be seemingly similar to the way rats learn how to avoid poisonous baits. The machine will simply memorize all previous e-mails that had been labeled as spam e-mails by the human user. When a new e-mail arrives, the machine will search for it in the set

of previous spam e-mails. If it matches one of them, it will be trashed. Otherwise, it will be moved to the user's inbox folder.
While the preceding "learning by memorization" approach is sometimes useful, it lacks an important aspect of learning systems – the ability to label unseen e-mail messages. A successful learner should be able to progress from individual examples to broader generalization. This is also referred to as inductive reasoning or inductive inference. In the bait shyness example presented previously, after the rats encounter an example of a certain type of food, they apply their attitude toward it on new, unseen examples of food of similar smell and taste. To achieve generalization in the spam filtering task, the learner can scan the previously seen e-mails, and extract a set of words whose appearance in an e-mail message is indicative of spam. Then, when a new e-mail arrives, the machine can check whether one of the suspicious words appears in it, and predict its label accordingly. Such a system would potentially be able correctly to predict the label of unseen e-mails.
However, inductive reasoning might lead us to false conclusions. To illustrate this, let us consider again an example from animal learning.

Pigeon Superstition: In an experiment performed by the psychologist B. F. Skinner, he placed a bunch of hungry pigeons in a cage. An automatic mechanism had been attached to the cage, delivering food to the pigeons at regular intervals with no reference whatsoever to the birds' behavior. The hungry pigeons went around the cage, and when food was first delivered, it found each pigeon engaged in some activity (pecking, turning the head, etc.). The arrival of food reinforced each bird's specific action, and consequently, each bird tended to spend some more time doing that very same action. That, in turn, increased the chance that the next random food delivery would find each bird engaged in that activity again. What results is a chain of events that reinforces the pigeons' association of the delivery of the food with whatever chance actions they had been performing when it was first delivered. They subsequently continue to perform these same actions diligently.[1]
What distinguishes learning mechanisms that result in superstition from useful learning? This question is crucial to the development of automated learners. While human learners can rely on common sense to filter out random meaningless learning conclusions, once we export the task of learning to a machine, we must provide well defined crisp principles that will protect the program from reaching senseless or useless conclusions. The development of such principles is a central goal of the theory of machine learning.
What, then, made the rats' learning more successful than that of the pigeons? As a first step toward answering this question, let us have a closer look at the bait shyness phenomenon in rats.
Bait Shyness revisited – rats fail to acquire conditioning between food and electric shock or between sound and nausea: The bait shyness mechanism in

[1] See: http://psychclassics.yorku.ca/Skinner/Pigeon

rats turns out to be more complex than what one may expect. In experiments carried out by Garcia (Garcia & Koelling 1996), it was demonstrated that if the unpleasant stimulus that follows food consumption is replaced by, say, electrical shock (rather than nausea), then no conditioning occurs. Even after repeated trials in which the consumption of some food is followed by the administration of unpleasant electrical shock, the rats do not tend to avoid that food. Similar failure of conditioning occurs when the characteristic of the food that implies nausea (such as taste or smell) is replaced by a vocal signal. The rats seem to have some "built in" prior knowledge telling them that, while temporal correlation between food and nausea can be causal, it is unlikely that there would be a causal relationship between food consumption and electrical shocks or between sounds and nausea.

We conclude that one distinguishing feature between the bait shyness learning and the pigeon superstition is the incorporation of prior knowledge that biases the learning mechanism. This is also referred to as inductive bias. The pigeons in the experiment are willing to adopt any explanation for the occurrence of food. However, the rats "know" that food cannot cause an electric shock and that the co-occurrence of noise with some food is not likely to affect the nutritional value of that food. The rats' learning process is biased toward detecting some kind of patterns while ignoring other temporal correlations between events.

It turns out that the incorporation of prior knowledge, biasing the learning process, is inevitable for the success of learning algorithms (this is formally stated and proved as the "No-Free-Lunch theorem" in Chapter 5). The development of tools for expressing domain expertise, translating it into a learning bias, and quantifying the effect of such a bias on the success of learning is a central theme of the theory of machine learning. Roughly speaking, the stronger the prior knowledge (or prior assumptions) that one starts the learning process with, the easier it is to learn from further examples. However, the stronger these prior assumptions are, the less flexible the learning is – it is bound, a priori, by the commitment to these assumptions. We shall discuss these issues explicitly in Chapter 5.

1.2 When Do We Need Machine Learning?

When do we need machine learning rather than directly program our computers to carry out the task at hand? Two aspects of a given problem may call for the use of programs that learn and improve on the basis of their "experience": the problem's complexity and the need for adaptivity.

Tasks That Are Too Complex to Program.

- Tasks Performed by Animals/Humans: There are numerous tasks that we human beings perform routinely, yet our introspection concerning how we do them is not sufficiently elaborate to extract a well

defined program. Examples of such tasks include driving, speech recognition, and image understanding. In all of these tasks, state of the art machine learning programs, programs that "learn from their experience," achieve quite satisfactory results, once exposed to sufficiently many training examples.

- Tasks beyond Human Capabilities: Another wide family of tasks that benefit from machine learning techniques are related to the analysis of very large and complex data sets: astronomical data, turning medical archives into medical knowledge, weather prediction, analysis of genomic data, Web search engines, and electronic commerce. With more and more available digitally recorded data, it becomes obvious that there are treasures of meaningful information buried in data archives that are way too large and too complex for humans to make sense of. Learning to detect meaningful patterns in large and complex data sets is a promising domain in which the combination of programs that learn with the almost unlimited memory capacity and ever increasing processing speed of computers opens up new horizons.

Adaptivity. One limiting feature of programmed tools is their rigidity – once the program has been written down and installed, it stays unchanged. However, many tasks change over time or from one user to another. Machine learning tools – programs whose behavior adapts to their input data – offer a solution to such issues; they are, by nature, adaptive to changes in the environment they interact with. Typical successful applications of machine learning to such problems include programs that decode handwritten text, where a fixed program can adapt to variations between the handwriting of different users; spam detection programs, adapting automatically to changes in the nature of spam e-mails; and speech recognition programs.

1.3 Types of Learning

Learning is, of course, a very wide domain. Consequently, the field of machine learning has branched into several subfields dealing with different types of learning tasks. We give a rough taxonomy of learning paradigms, aiming to provide some perspective of where the content of this book sits within the wide field of machine learning.

We describe four parameters along which learning paradigms can be classified.

Supervised versus Unsupervised Since learning involves an interaction between the learner and the environment, one can divide learning tasks according to the nature of that interaction. The first distinction to note is the difference between supervised and unsupervised learning. As an

illustrative example, consider the task of learning to detect spam e-mail versus the task of anomaly detection. For the spam detection task, we consider a setting in which the learner receives training e-mails for which the label spam/not-spam is provided. On the basis of such training the learner should figure out a rule for labeling a newly arriving e-mail message. In contrast, for the task of anomaly detection, all the learner gets as training is a large body of e-mail messages (with no labels) and the learner's task is to detect "unusual" messages.

More abstractly, viewing learning as a process of "using experience to gain expertise," supervised learning describes a scenario in which the "experience," a training example, contains significant information (say, the spam/not-spam labels) that is missing in the unseen "test examples" to which the learned expertise is to be applied. In this setting, the acquired expertise is aimed to predict that missing information for the test data. In such cases, we can think of the environment as a teacher that "supervises" the learner by providing the extra information (labels). In unsupervised learning, however, there is no distinction between training and test data. The learner processes input data with the goal of coming up with some summary, or compressed version of that data. Clustering a data set into subsets of similar objets is a typical example of such a task.

There is also an intermediate learning setting in which, while the training examples contain more information than the test examples, the learner is required to predict even more information for the test exam- ples. For example, one may try to learn a value function that describes for each setting of a chess board the degree by which White's position is bet- ter than the Black's. Yet, the only information available to the learner at training time is positions that occurred throughout actual chess games, labeled by who eventually won that game. Such learning frameworks are mainly investigated under the title of reinforcement learning.

Active versus Passive Learners

Learning paradigms can vary by the role played by the learner. We distinguish between "active" and "passive" learners. An active learner interacts with the environment at training time, say, by posing queries or performing experiments, while a passive learner only observes the information provided by the environment (or the teacher) without influencing or directing it. Note that the learner of a spam filter is usually passive – waiting for users to mark the e-mails com- ing to them. In an active setting, one could imagine asking users to label specific e-mails chosen by the learner, or even composed by the learner, to enhance its understanding of what spam is.

Helpfulness of the Teacher

When one thinks about human learning, of a baby at home or a student at school, the process often involves a helpful teacher, who is trying to feed the learner with the information most use-

ful for achieving the learning goal. In contrast, when a scientist learns about nature, the environment, playing the role of the teacher, can be best thought of as passive – apples drop, stars shine, and the rain falls without regard to the needs of the learner. We model such learning scenarios by postulating that the training data (or the learner's experience) is generated by some random process. This is the basic building block in the branch of "statistical learning." Finally, learning also occurs when the learner's input is generated by an adversarial "teacher." This may be the case in the spam filtering example (if the spammer makes an effort to mislead the spam filtering designer) or in learning to detect fraud. One also uses an adversarial teacher model as a worst-case scenario, when no milder setup can be safely assumed. If you can learn against an adversarial teacher, you are guaranteed to succeed interacting any odd teacher.

Online versus Batch Learning Protocol The last parameter we mention is the distinction between situations in which the learner has to respond online, throughout the learning process, and settings in which the learner has to engage the acquired expertise only after having a chance to process large amounts of data. For example, a stockbroker has to make daily decisions, based on the experience collected so far. He may become an expert over time, but might have made costly mistakes in the process. In contrast, in many data mining settings, the learner – the data miner – has large amounts of training data to play with before having to output conclusions.

In this book we shall discuss only a subset of the possible learning paradigms. Our main focus is on supervised statistical batch learning with a passive learner (for example, trying to learn how to generate patients' prognoses, based on large archives of records of patients that were independently collected and are already labeled by the fate of the recorded patients). We shall also briefly discuss online learning and batch unsupervised learning (in particular, clustering).

1.4 Relations to Other Fields

As an interdisciplinary field, machine learning shares common threads with the mathematical fields of statistics, information theory, game theory, and optimization. It is naturally a subfield of computer science, as our goal is to program machines so that they will learn. In a sense, machine learning can be viewed as a branch of AI (Artificial Intelligence), since, after all, the ability to turn expe- rience into expertise or to detect meaningful patterns in complex sensory data is a cornerstone of human (and animal) intelligence. However, one should note that, in contrast with traditional AI, machine learning is not trying to build automated imitation of intelligent behavior, but rather to use the strengths and

special abilities of computers to complement human intelligence, often performing tasks that fall way beyond human capabilities. For example, the ability to scan and process huge databases allows machine learning programs to detect patterns that are outside the scope of human perception.

The component of experience, or training, in machine learning often refers to data that is randomly generated. The task of the learner is to process such randomly generated examples toward drawing conclusions that hold for the environment from which these examples are picked. This description of machine learning highlights its close relationship with statistics. Indeed there is a lot in common between the two disciplines, in terms of both the goals and techniques used. There are, however, a few significant differences of emphasis; if a doctor comes up with the hypothesis that there is a correlation between smoking and heart disease, it is the statistician's role to view samples of patients and check the validity of that hypothesis (this is the common statistical task of hypothe- sis testing). In contrast, machine learning aims to use the data gathered from samples of patients to come up with a description of the causes of heart disease. The hope is that automated techniques may be able to figure out meaningful patterns (or hypotheses) that may have been missed by the human observer.

In contrast with traditional statistics, in machine learning in general, and in this book in particular, algorithmic considerations play a major role. Ma- chine learning is about the execution of learning by computers; hence algorith- mic issues are pivotal. We develop algorithms to perform the learning tasks and are concerned with their computational efficiency. Another difference is that while statistics is often interested in asymptotic behavior (like the convergence of sample-based statistical estimates as the sample sizes grow to infinity), the theory of machine learning focuses on finite sample bounds. Namely, given the size of available samples, machine learning theory aims to figure out the degree of accuracy that a learner can expect on the basis of such samples.

There are further differences between these two disciplines, of which we shall mention only one more here. While in statistics it is common to work under the assumption of certain presubscribed data models (such as assuming the normality of data-generating distributions, or the linearity of functional dependencies), in machine learning the emphasis is on working under a "distribution-free" set- ting, where the learner assumes as little as possible about the nature of the data distribution and allows the learning algorithm to figure out which models best approximate the data-generating process. A precise discussion of this issue requires some technical preliminaries, and we will come back to it later in the book, and in particular in Chapter 5.

1.5 How to Read This Book

The first part of the book provides the basic theoretical principles that underlie machine learning (ML). In a sense, this is the foundation upon which the rest

of the book is built. This part could serve as a basis for a minicourse on the theoretical foundations of ML.

The second part of the book introduces the most commonly used algorithmic approaches to supervised machine learning. A subset of these chapters may also be used for introducing machine learning in a general AI course to computer science, Math, or engineering students.

The third part of the book extends the scope of discussion from statistical classification to other learning models. It covers online learning, unsupervised learning, dimensionality reduction, generative models, and feature learning.

The fourth part of the book, Advanced Theory, is geared toward readers who have interest in research and provides the more technical mathematical tech- niques that serve to analyze and drive forward the field of theoretical machine learning.

The Appendixes provide some technical tools used in the book. In particular, we list basic results from measure concentration and linear algebra.

A few sections are marked by an asterisk, which means they are addressed to more advanced students. Each chapter is concluded with a list of exercises. A solution manual is provided in the course Web site.

1.5.1 Possible Course Plans Based on This Book

A 14 Week Introduction Course for Graduate Students:

1. Chapters 2–4.
2. Chapter 9 (without the VC calculation).
3. Chapters 5–6 (without proofs).
4. Chapter 10.
5. Chapters 7, 11 (without proofs).
6. Chapters 12, 13 (with some of the easier proofs).
7. Chapter 14 (with some of the easier proofs).
8. Chapter 15.
9. Chapter 16.
10. Chapter 18.
11. Chapter 22.
12. Chapter 23 (without proofs for compressed sensing).
13. Chapter 24.
14. Chapter 25.

A 14 Week Advanced Course for Graduate Students:

1. Chapters 26, 27.
2. (continued)
3. Chapters 6, 28.
4. Chapter 7.
5. Chapter 31.

6. Chapter 30.
7. Chapters 12, 13.
8. Chapter 14.
9. Chapter 8.
10. Chapter 17.
11. Chapter 29.
12. Chapter 19.
13. Chapter 20.
14. Chapter 21.

1.6 Notation

Most of the notation we use throughout the book is either standard or defined on the spot. In this section we describe our main conventions and provide a table summarizing our notation (Table 1.1). The reader is encouraged to skip this section and return to it if during the reading of the book some notation is unclear.

We denote scalars and abstract objects with lowercase letters (e.g. x and λ). Often, we would like to emphasize that some object is a vector and then we use boldface letters (e.g. x and λ). The ith element of a vector x is denoted by xi. We use uppercase letters to denote matrices, sets, and sequences. The meaning should be clear from the context. As we will see momentarily, the input of a learning algorithm is a sequence of training examples. We denote by z an abstract example and by S = z1, . . . , zm a sequence of m examples. Historically, S is often referred to as a training set; however, we will always assume that S is a sequence rather than a set. A sequence of m vectors is denoted by x1, . . . , xm. The ith element of xt is denoted by xt,i.

Throughout the book, we make use of basic notions from probability. We denote by D a distribution over some set,[2] for example, Z. We use the notation z D to denote that z is sampled according to D. Given a random variable f : Z → R, its expected value is denoted by Ez D[f(z)]. We sometimes use the shorthand E[f] when the dependence on z is clear from the context. For f : Z → {true,false} we also use Pz D[f(z)] to denote D({z : f(z) = true}). In the next chapter we will also introduce the notation Dm to denote the probability over Zm induced by sampling (z1,...,zm) where each point zi is sampled from D independently of the other points.

In general, we have made an effort to avoid asymptotic notation. However, we occasionally use it to clarify the main results. In particular, given f : R → R+ we write f = O(g) if there exist x0, α ∈ R+ such that for all x > x0 we have f(x) ≤ αg(x). We write f = o(g) if for every α > 0 there exists

[2] To be mathematically precise, D should be defined over some σ-algebra of subsets of Z. The user who is not familiar with measure theory can skip the few footnotes and remarks regarding more formal measurability definitions and assumptions.

Table 1.1 Summary of notation

symbol	meaning
R	the set of real numbers
Rd	the set of d-dimensional vectors over R
R+	the set of non-negative real numbers
N	the set of natural numbers
O, o, Θ, ω, Ω, Õ	asymptotic notation (see text)
1[Boolean expression]	indicator function (equals 1 if expression is true and 0 o.w.)
[a]+	=max
[n]	{0, a}
x v	the set
xi, vi, wi	{1,...,n}(forn∈N)
w(	(co tors
⟩ ‖x‖	the ———
x v	∑lumn)veci=
‖ xor	√thelementofavectord
‖	
x	
A Rd,k	∑i=1xivi(inn`e`rproduct)= ⟨x,x⟩ (the2`normofx)=di=1|xi|(the1normofx)=maxi|xi|(the
A ‖	
A>	
Ai, j	the number of nonzero elements of x
A[i, j]	a d
x x(i)>	× k matrix over R
x1, . . . , xm	the transpose of A
xi,j	the (i, j) element of A
w(1), . . . w(T)	thed
. ,	×dmatrixAs.t.A=(whereRdi,jxixjx∈)
v(t)	a sequence of m vectors
vi	the jth element of the ith vector in the sequence
X	the values of a vector w during an iterative algorithm
Y	the ith element of the vector w(t)
Z	instances domain (a set)
H	labels domain (a set)
`: Z×R+ →	examples domain (a set)
D	hypothesis class (a set)
h(A)	loss function
z D	a distribution over some set (usually over Z or over X)
S = z1,...,zm	the probability of a set A
S Dm	⊆ Z according to D
P,E	sampling z according to
P D[f(z)]	D
E f z	a sequence of m examples
N D[C()]	sampling S = z1,...,zm i.i.d. according to
f(u)	D
f(x)	probability and expectation of a random variable
f''()	D({z : f(z) = true}) for f : Z → {true, false}
∂f (w)	expectation of the random variable f : Z
∂wi	→ R
∇f(w)	Gaussian distribution with expectation μ and covariance C
f(w)	the derivative of a function f : R
f(x)	→Ratx
min f(x)	the second derivative of a function f : R
x∈C()	→Ratx
max f(x)	the partial derivative of a function f : Rd
x∈C()	→ R at w w.r.t. wi
argmin f(x)	the gradient of a function f : Rd
x∈C()	→Ratw
argmax f(x)	the differential set of a function f : Rd
x∈C()	→Ratw
log	=min
	{f(x) : x ∈ C} (minimal value of f over C)
	=max
	{f(x) : x ∈ C} (maximal value of f over C)
	the set
	{x ∈ C : f(x) = minz∈C f(z)}

x0 such that for all x > x0 we have f(x) ≤ αg(x). We write f = Ω(g) if there exist x0,α ∈ R+ such that for all x > x0 we have f(x) ≥ αg(x). The notation f = ω(g) is defined analogously. The notation f = Θ(g) means that f = O(g) and g = O(f). Finally, the notation f = Õ(g) means that there exists k ∈ N such that f(x) = O(g(x) logk(g(x))).

The inner product between vect

√∑orsxandwisdenoted` byx,w.Wheneverwedonotspecifythevectorspaceweassumethatitisthed-dimensionalEucl

∑∑` 2normwhenitisclearfrom1/pthecontext.Wealsouseotherpnorms,||w||=(||ppiwi),andinparticular||w||1=i|wi|an

{f(x) : x ∈ C}. To be mathematically more precise, we should use infx∈C f(x) whenever the minimum is not achievable. However, in the context of this book the distinction between infimum and minimum is often of little interest. Hence, to simplify the presentation, we sometimes use the min notation even when inf is more adequate. An analogous remark applies to max versus sup.

Part I

Foundations

2 A Gentle Start

Let us begin our mathematical analysis by showing how successful learning can be achieved in a relatively simplified setting. Imagine you have just arrived in some small Pacific island. You soon find out that papayas are a significant ingredient in the local diet. However, you have never before tasted papayas. You have to learn how to predict whether a papaya you see in the market is tasty or not. First, you need to decide which features of a papaya your prediction should be based on. On the basis of your previous experience with other fruits, you decide to use two features: the papaya's color, ranging from dark green, through orange and red to dark brown, and the papaya's softness, ranging from rock hard to mushy. Your input for figuring out your prediction rule is a sample of papayas that you have examined for color and softness and then tasted and found out whether they were tasty or not. Let us analyze this task as a demonstration of the considerations involved in learning problems.

Our first step is to describe a formal model aimed to capture such learning tasks.

2.1 A Formal Model – The Statistical Learning Framework

- The learner's input: In the basic statistical learning setting, the learner has access to the following:
  - Domain set: An arbitrary set, X. This is the set of objects that we may wish to label. For example, in the papaya learning problem mentioned before, the domain set will be the set of all papayas. Usually, these domain points will be represented by a vector of features (like the papaya's color and softness). We also refer to domain points as instances and to X as instance space.
  - Label set: For our current discussion, we will restrict the label set to be a two-element set, usually $\{0,1\}$ or $\{-1,+1\}$. Let Y denote our set of possible labels. For our papayas example, let Y be $\{0, 1\}$, where 1 represents being tasty and 0 stands for being not-tasty.
  - Training data: $S = ((x_1, y_1) \ldots (x_m, y_m))$ is a finite sequence of pairs in $X \times Y$: that is, a sequence of labeled domain points. This is the input that the learner has access to (like a set of papayas that have been

tasted and their color, softness, and tastiness). Such labeled examples are often called training examples. We sometimes also refer to S as a training set.[1]

- The learner's output: The learner is requested to output a prediction rule, $h : X \to Y$. This function is also called a predictor, a hypothesis, or a classifier. The predictor can be used to predict the label of new domain points. In our papayas example, it is a rule that our learner will employ to predict whether future papayas he examines in the farmers' market are going to be tasty or not. We use the notation A(S) to denote the hypothesis that a learning algorithm, A, returns upon receiving the training sequence S.
- A simple data-generation model We now explain how the training data is generated. First, we assume that the instances (the papayas we encounter) are generated by some probability distribution (in this case, representing the environment). Let us denote that probability distribution over X by D. It is important to note that we do not assume that the learner knows anything about this distribution. For the type of learning tasks we discuss, this could be any arbitrary probability distribution. As to the labels, in the current discussion we assume that there is some "correct" labeling function, $f : X \to Y$, and that $y_i = f(x_i)$ for all i. This assumption will be relaxed in the next chapter. The labeling function is unknown to the learner. In fact, this is just what the learner is trying to figure out. In summary, each pair in the training data S is generated by first sampling a point x_i according to D and then labeling it by f.
- Measures of success: We define the error of a classifier to be the probability that it does not predict the correct label on a random data point generated by the aforementioned underlying distribution. That is, the error of h is the probability to draw a random instance x, according to the distribution D, such that h(x) does not equal f(x).
Formally, given a domain subset,[2] $A \subset X$, the probability distribution, D, assigns a number, D(A), which determines how likely it is to observe a point $x \in A$. In many cases, we refer to A as an event and express it using a function $\pi : X \to \{0,1\}$, namely, $A = \{x \in X : \pi(x) = 1\}$. In that case, we also use the notation $P_{x \sim D}[\pi(x)]$ to express D(A).
We define the error of a prediction rule, $h : X \to Y$, to be
$L_{(D,f)}(h) \stackrel{\text{def}}{=} P_{x \sim D}[h(x) \neq f(x)] \stackrel{\text{def}}{=} D(\{x : h(x) \neq f(x)\})$ (2.1)
That is, the error of such h is the probability of randomly choosing an example x for which $h(x) \neq f(x)$. The subscript (D,f) indicates that the error is measured with respect to the probability distribution D and the

[1] Despite the "set" notation, S is a sequence. In particular, the same example may appear twice in S and some algorithms can take into account the order of examples in S.

[2] Strictly speaking, we should be more careful and require that A is a member of some σ-algebra of subsets of X, over which D is defined. We will formally define our measurability assumptions in the next chapter.

correct labeling function f. We omit this subscript when it is clear from the context. L(D,f)(h) has several synonymous names such as the generalization error, the risk, or the true error of h, and we will use these names interchangeably throughout the book. We use the letter L for the error, since we view this error as the loss of the learner. We will later also discuss other possible formulations of such loss.

• A note about the information available to the learner The learner is blind to the underlying distribution D over the world and to the labeling function f. In our papayas example, we have just arrived in a new island and we have no clue as to how papayas are distributed and how to predict their tastiness. The only way the learner can interact with the environment is through observing the training set.

In the next section we describe a simple learning paradigm for the preceding setup and analyze its performance.

2.2 Empirical Risk Minimization

As mentioned earlier, a learning algorithm receives as input a training set S, sampled from an unknown distribution D and labeled by some target function f, and should output a predictor $h_S : X \to Y$ (the subscript S emphasizes the fact that the output predictor depends on S). The goal of the algorithm is to find h_S that minimizes the error with respect to the unknown D and f.

Since the learner does not know what D and f are, the true error is not directly available to the learner. A useful notion of error that can be calculated by the learner is the training error – the error the classifier incurs over the training sample:

$$L_S(h) \stackrel{\text{def}}{=} \frac{|\{i \in [m] : h(x_i) = y_i\}|}{m}, \quad (2.2)$$

where $[m] = \{1, \ldots, m\}$.

The terms empirical error and empirical risk are often used interchangeably for this error.

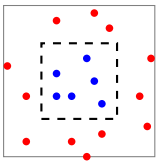
Since the training sample is the snapshot of the world that is available to the learner, it makes sense to search for a solution that works well on that data. This learning paradigm – coming up with a predictor h that minimizes $L_S(h)$ – is called Empirical Risk Minimization or ERM for short.

2.2.1 Something May Go Wrong – Overfitting

Although the ERM rule seems very natural, without being careful, this approach may fail miserably.

To demonstrate such a failure, let us go back to the problem of learning to

predict the taste of a papaya on the basis of its softness and color. Consider a sample as depicted in the following:



Assume that the probability distribution D is such that instances are distributed uniformly within the gray square and the labeling function, f, determines the label to be 1 if the instance is within the inner blue square, and 0 otherwise. The area of the gray square in the picture is 2 and the area of the blue square is 1. Consider the following predictor

$$\{:yiif\exists i\in[m]s.t.xi=xhS(x)=(2.3)0otherwise.$$

While this predictor might seem rather artificial, in Exercise 1 we show a natural representation of it using polynomials. Clearly, no matter what the sample is, $L_S(h_S) = 0$, and therefore this predictor may be chosen by an ERM algorithm (it is one of the empirical-minimum-cost hypotheses; no classifier can have smaller error). On the other hand, the true error of any classifier that predicts the label 1 only on a finite number of instances is, in this case, 1/2. Thus, $L_D(h_S) = 1/2$. We have found a predictor whose performance on the training set is excellent, yet its performance on the true "world" is very poor. This phenomenon is called overfitting. Intuitively, overfitting occurs when our hypothesis fits the training data "too well" (perhaps like the everyday experience that a person who provides a perfect detailed explanation for each of his single actions may raise suspicion).

2.3 Empirical Risk Minimization with Inductive Bias

We have just demonstrated that the ERM rule might lead to overfitting. Rather than giving up on the ERM paradigm, we will look for ways to rectify it. We will search for conditions under which there is a guarantee that ERM does not overfit, namely, conditions under which when the ERM predictor has good performance with respect to the training data, it is also highly likely to perform well over the underlying data distribution.

A common solution is to apply the ERM learning rule over a restricted search space. Formally, the learner should choose in advance (before seeing the data) a set of predictors. This set is called a hypothesis class and is denoted by $\mathcal{H}$. Each $h \in \mathcal{H}$ is a function mapping from $\mathcal{X}$ to $\mathcal{Y}$. For a given class $\mathcal{H}$, and a training sample, S, the ERM$_{\mathcal{H}}$ learner uses the ERM rule to choose a predictor $h \in \mathcal{H}$,

with the lowest possible error over S. Formally,

$$\text{ERM}_H(S) \in \operatorname*{argmin}_{h \in H} L_S(h),$$

where argmin stands for the set of hypotheses in H that achieve the minimum value of $L_S(h)$ over H. By restricting the learner to choosing a predictor from H, we bias it toward a particular set of predictors. Such restrictions are often called an inductive bias. Since the choice of such a restriction is determined before the learner sees the training data, it should ideally be based on some prior knowledge about the problem to be learned. For example, for the papaya taste prediction problem we may choose the class H to be the set of predictors that are determined by axis aligned rectangles (in the space determined by the color and softness coordinates). We will later show that ERM_H over this class is guaranteed not to overfit. On the other hand, the example of overfitting that we have seen previously, demonstrates that choosing H to be a class of predictors that includes all functions that assign the value 1 to a finite set of domain points does not suffice to guarantee that ERM_H will not overfit.

A fundamental question in learning theory is, over which hypothesis classes ERM_H learning will not result in overfitting. We will study this question later in the book.

2.3.1 Finite Hypothesis Classes

Intuitively, choosing a more restricted hypothesis class better protects us against overfitting but at the same time might cause us a stronger inductive bias. We will get back to this fundamental tradeoff later.

The simplest type of restriction on a class is imposing an upper bound on its size (that is, the number of predictors in H). In this section, we show that if H is a finite class then ERM_H will not overfit, provided it is based on a sufficiently large training sample (this size requirement will depend on the size of H).

Limiting the learner to prediction rules within some finite hypothesis class may be considered as a reasonably mild restriction. For example, H can be the set of all predictors that can be implemented by a C++ program written in at most 109 bits of code. In our papayas example, we mentioned previously the class of axis aligned rectangles. While this is an infinite class, if we discretize the representation of real numbers, say, by using a 64 bits floating-point representation, the hypothesis class becomes a finite class.

Let us now analyze the performance of the ERM_H learning rule assuming that H is a finite class. For a training sample, S, labeled according to some $f : X \to Y$, let h_S denote a result of applying ERM_H to S, namely,

$$h_S \in \operatorname*{argmin}_{h \in H} L_S(h). \quad (2.4)$$

In this chapter, we make the following simplifying assumption (which will be relaxed in the next chapter).

definition 2.1 (The Realizability Assumption) There exists h? ∈ H s.t. L(D)(h?,f)=0.Notethatthisassumptionimpliesthatwithprobability1over random samples, S, where the instances of S are sampled according to D and are labeled by f, we have L h?S( ) = 0.

The realizability assumption implies that for every ERM hypothesis we have that[3] LS(hS) = 0. However, we are interested in the true risk of hS, L(D,f) (hS), rather than its empirical risk.

Clearly,anyguaranteeontheerrorwithrespecttotheunderlyingdistribution, D, for an algorithm that has access only to a sample S should depend on the relationship between D and S. The common assumption in statistical machine learning is that the training sample S is generated by sampling points from the distribution D independently of each other. Formally,

• The i.i.d. assumption: The examples in the training set are independently and identically distributed (i.i.d.) according to the distribution D. That is, every xi in S is freshly sampled according to D and then labeled according to the labeling function, f. We denote this assumption by S Dm where m is the size of S, and Dm denotes the probability over m-tuples induced by applying D to pick each element of the tuple independently of the other members of the tuple.

Intuitively, the training set S is a window through which the learner gets partial information about the distribution D over the world and the labeling function, f. The larger the sample gets, the more likely it is to reflect more accurately the distribution and labeling used to generate it. Since L(D,f)(hS) depends on the training set, S, and that training set is picked by a random process, there is randomness in the choice of the predictor hS and, consequently, in the risk L(D,f)(hS). Formally, we say that it is a random variable. It is not realistic to expect that with full certainty S will suffice to direct the learner toward a good classifier (from the point of view of D), as there is always some probability that the sampled training data happens to be very nonrepresentative of the underlying D. If we go back to the papaya tasting example, there is always some (small) chance that all the papayas we have happened to taste were not tasty, in spite of the fact that, say, 70% of the papayas in our island are tasty. In such a case, ERM H(S) may be the constant function that labels every papaya as “not tasty” (and has 70% error on the true distribution of papapyas in the island). We will therefore address the probability to sample a training set for which L(D,f)(hS) is not too large. Usually, we denote the probability of getting a nonrepresentative sample by δ, and call (1 − δ) the confidence parameter of our prediction.

On top of that, since we cannot guarantee perfect label prediction, we introduce another parameter for the quality of prediction, the accuracy parameter

[3] Mathematically speaking, this holds with probability 1. To simplify the presentation, we sometimes omit the “with probability 1” specifier.

commonly denoted by ε. We interpret the event L(D,f)(hS) > ε as a failure of the learner, while if L(D,f)(hS) ≤ ε we view the output of the algorithm as an approximately correct predictor. Therefore (fixing some labeling function f : X → Y), we are interested in upper bounding the probability to sample m-tuple of instances that will lead to failure of the learner. Formally, let S |x = (x1,...,xm) be the instances of the training set. We would like to upper bound

$$D^m(\{S|_x : L_{(D,f)}(h_S) > \epsilon\}).$$

Let { ∈ H HB be the set of "bad" hypotheses, that is, In addition, let HB=h :L(D,f)(h)>ε}.

M = {S|x : ∃h ∈ HB, LS(h) = 0}

be the set of misleading samples: Namely, for every S |x ∈ M, there is a "bad" hypothesis, h ∈ HB, that looks like a "good" hypothesis on S|x. Now, recall that we would like to bound the probability of the event L(D,f)(hS) > ε. But, since the realizability assumption implies that LS(hS) = 0, it follows that the event L(D,f)(hS) > ε can only happen if for some h ∈ HB we have LS(h) = 0. In other words, this event will only happen if our sample is in the set of misleading samples, M. Formally, we have shown that

$$\{S|_x : L_{(D,f)}(h_S) > \epsilon\} \subseteq M .$$

Note that we can rewrite M as

Hence,

$$M=\{S|x:LS(h)=0\}. \quad (2.5) \qquad D^m(\{S|x:L_{h}\ \epsilon\ D^m\ m_{(D,f)}(S)>\}) \le (M)=D(\cup h\in H\{S|x:LS(h)=0\}).B$$

(2.6)

Next, we upper bound the right-hand side of the preceding equation using the union bound – a basic property of probabilities.

lemma 2.2 (Union Bound) For any two sets A,B and a distribution D we have

$$D(A \cup B) \le D(A) + D(B).$$

Applying the union bound to the righ

} ≤

∑t-handsideofEquation(2.6)yieldsDm({S|:L((h>εDmxD,f)S))({S|x:LS(h)=0}).(2.7)h∈HB

Next, let us bound each summand of the right-hand side of the preceding inequality.Fixsome"bad"hypothesish B.Theeventh∈LS(h)=0isequivalent

to the event $\forall i, h(x_i) = f(x_i)$. Since the examples in the training set are sampled i.i.d. we get that

$$D^m(\{S : L_{(D,f)}(h) = 0\}) = D^m(\{S : \forall i, h(x_i) = f(x_i)\}) = \prod_{i=1}^{m} D(\{x_i : h(x_i) = f(x_i)\}). \quad (2.8)$$

For each individual sampling of an element of the training set we have

$$D(\{x_i : h(x_i) = y_i\}) = 1 - L_{(D,f)}(h) \leq 1 - \epsilon,$$

where the last inequality follows from the fact that $h \in H_B$. Combining the previous equation with Equation (2.8) and using the inequality $1 - \epsilon \leq e^{-\epsilon}$ we obtain that for every $h \in H_B$,

$$D^m(\{S : L_{(D,f)}(h) = 0\}) \leq (1-\epsilon)^m \leq e^{-\epsilon m}. \quad (2.9)$$

Combining this equation with Equation (2.7) we conclude that

$$D^m(\{S : L_{(D,f)}(h) > \epsilon\}) \leq |H_B| e^{-\epsilon m} \leq |H| e^{-\epsilon m} \leq |H| e^{-\epsilon m}.$$

A graphical illustration which explains how we used the union bound is given in Figure 2.1.

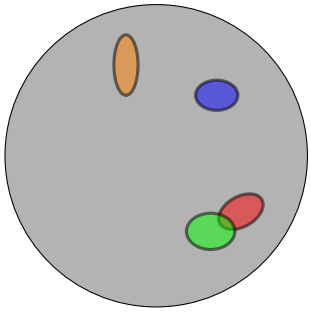


Figure 2.1 Each point in the large circle represents a possible m-tuple of instances. Each colored oval represents the set of "misleading" m-tuple of instances for some "bad" predictor $h \in H_B$. The ERM can potentially overfit whenever it gets a misleading training set S. That is, for some $h \in H_B$ we have $L_S(h) = 0$. Equation (2.9) guarantees that for each individual bad hypothesis, $h \in H_B$, at most $(1 - \epsilon)$-fraction of the training sets would be misleading. In particular, the larger m is, the smaller each of these colored ovals becomes. The union bound formalizes the fact that the area representing the training sets that are misleading with respect to some $h \in H_B$ (that is, the training sets in M) is at most the sum of the areas of the colored ovals. Therefore, it is bounded by $|H_B|$ times the maximum size of a colored oval. Any sample S outside the colored ovals cannot cause the ERM rule to overfit.

corollary 2.3 Let H be a finite hypothesis class. Let $\delta \in (0,1)$ and $\epsilon > 0$

and let m be an integer that satisfies

$$m \geq \frac{\log(|H|/\delta)}{\epsilon}.$$

Then, for any labeling function, f, and for any distribution, D, for which the realizability assumption holds (that is, for some $h \in H$, $L(D,f)(h_S) = 0$), with probability of at least $1-\delta$ over the choice of an i.i.d. sample S of size m, we have that for every ERM hypothesis, h_S, it holds that $L(D,f)(h_S) \leq \epsilon$.

The preceeding corollary tells us that for a sufficiently large m, the ERM rule over a finite hypothesis class will be probably (with confidence $1-\delta$) approximately (up to an error of ϵ) correct. In the next chapter we formally define the model of Probably Approximately Correct (PAC) learning.

2.4 Exercises

1. Overfitting of polynomial matching: We have shown that the predictor defined in Equation (2.3) leads to overfitting. While this predictor seems to be very unnatural, the goal of this exercise is to show that it can be described as a thresholded polynomial. That is, show that given a training set $S = \{(x_i,f(x_i))\}_{i=1}^m \subseteq \mathbb{R}^d \times \{0,1\}$, there exists a polynomial p_S such that $h_S(x) = 1$ if and only if $p_S(x) \geq 0$, where h_S is as defined in Equation (2.3).

It follows that learning the class of all thresholded polynomials using the ERM rule may lead to overfitting.

2. Let H be a class of binary classifiers over a domain X. Let D be an unknown distribution over X, and let f be the target hypothesis in H. Fix some $h \in H$. Show that the expected value of $L_S(h)$ over the choice of S equals $L_{(D,f)}(h)$, namely,

$$E_{S \sim D^m}[L_S(h)] = L_{(D,f)}(h).$$

3. Axis aligned rectangles: An axis aligned rectangle classifier in the plane is a classifier that assigns the value 1 to a point if and only if it is inside a certain rectangle. Formally, given real numbers $a_1 \leq b_1, a_2 \leq b_2$, define the classifier $h_{(a_1,b_1,a_2,b_2)}$ by

$$h_{(a_1,b_1,a_2,b_2)}(x_1,x_2) = \begin{cases} 1 & \text{if } a_1 \leq x_1 \leq b_1 \text{ and } a_2 \leq x_2 \leq b_2 \\ 0 & \text{otherwise} \end{cases} \quad (2.10)$$

The class of all axis aligned rectangles in the plane is defined as

$$H^2_{rec} = \{h_{(a_1,b_1,a_2,b_2)} : a_1 \leq b_1, a_2 \leq b_2\}.$$

Note that this is an infinite size hypothesis class. Throughout this exercise we rely on the realizability assumption.

1. Let A be the algorithm that returns the smallest rectangle enclosing all positive examples in the training set. Show that A is an ERM.
2. Show that if A receives a training set of size $\geq \frac{4\log(4/\delta)}{\varepsilon}$ then, with probability of at least $1-\delta$ it returns a hypothesis with error of at most ε.
Hint: Fix some distribution D over X, let $R^* = R(a_1^*, b_1^*, a_2^*, b_2^*)$ be the rectangle that generates the labels, and let f be the corresponding hypothesis. Let $a_1^* \geq a_1$ be a number such that the probability mass (with respect to D) of the rectangle $R = R(a_1, a_1^*, a_2, b_2)$ is exactly $\varepsilon/4$. Similarly, let b_1, a_2, b_2 be numbers such that the probability masses of the rectangles $R = R(b_1^*, b_1, a_2^*, b_2^*)$, $R = R(a_1^*, b_1^*, a_2^*, a_2)$, $R4 = R(a_1^*, b_1^*, b_2^*, b_2)$ are all exactly $\varepsilon/4$. Let R(S) be the rectangle returned by A. See illustration in Figure 2.2.

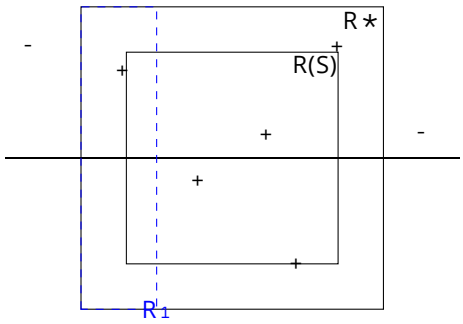


Figure 2.2 Axis aligned rectangles.

- Show that $R(S) \subseteq R^*$.
- Show that if S contains (positive) examples in all of the rectangles R_1, R_2, R_3, R_4, then the hypothesis returned by A has error of at most ε.
- For each $i \in \{1,\ldots,4\}$, upper bound the probability that S does not contain an example from R_i.
- Use the union bound to conclude the argument.

3. Repeat the previous question for the class of axis aligned rectangles in $\mathbb{R}^d$.
4. Show that the runtime of applying the algorithm A mentioned earlier is polynomial in d, $1/\varepsilon$, and in $\log(1/\delta)$.

3 A Formal Learning Model

In this chapter we define our main formal learning model – the PAC learning model and its extensions. We will consider other notions of learnability in Chapter 7.

3.1 PAC Learning

In the previous chapter we have shown that for a finite hypothesis class, if the ERM rule with respect to that class is applied on a sufficiently large training sample (whose size is independent of the underlying distribution or labeling function) then the output hypothesis will be probably approximately correct. More generally, we now define Probably Approximately Correct (PAC) learning.

definition 3.1 (PAC Learnability) A hypothesis class

H is PAC learnable

ifthereexistafunctionm 2

H : (0,1) → N and a learning algorithm with the following property: For every ε,δ

∈ (0,1), for every distribution D over X, and for every labeling function f :

X → {0,1}, if the realizable assumption holds with respect to

H,D,f, then when running the learning algorithm on m ≥ m

H(ε, δ) i.i.d. examples generated by D and labeled by f, the algorithm returns a hypothesis h such that, with probability of at least 1

− δ (over the choice of the examples), L(

D,f)(h) ≤ ε.

The definition of Probably Approximately Correct learnability contains two approximation parameters. The accuracy parameter ε determines how far the output classifier can be from the optimal one (this corresponds to the "approximately correct"), and a confidence parameter δ indicating how likely the classifier is to meet that accuracy requirement (corresponds to the "probably" part of "PAC"). Under the data access model that we are investigating, these approximations are inevitable. Since the training set is randomly generated, there may always be a small chance that it will happen to be noninformative (for example, there is always some chance that the training set will contain only one domain point, sampled over and over again). Furthermore, even when we are lucky enough to get a training sample that does faithfully represent , because it is just a finite sample, there may always be some fine details of that it fails

D

D

to reflect. Our accuracy parameter, ε, allows "forgiving" the learner's classifier for making minor errors.

Sample Complexity

The function $m : (0 1)2$ $H , \rightarrow N$ determines the sample complexity of learning H: that is, how many examples are required to guarantee a probably approximately correct solution. The sample complexity is a function of the accuracy (ε) and confidence (δ) parameters. It also depends on properties of the hypothesis class H – for example, for a finite class we showed that the sample complexity depends on log the size of H.

Note that if H is PAC learnable, there are many functions mH that satisfy the requirements given in the definition of PAC learnability. Therefore, to be precise, we will define the sample complexity of learning H to be the "minimal function," in the sense that for any ε,δ, m H(ε,δ) is the minimal integer that satisfies the requirements of PAC learning with accuracy ε and confidence δ.

Let us now recall the conclusion of the analysis of finite hypothesis classes from the previous chapter. It can be rephrased as stating:

corollary 3.2 Every finite hypothesis class is PAC learnable with sample complexity $\lceil \log(|H|/\delta)/\epsilon \rceil \leq m_H(\epsilon,\delta)$.

There are infinite classes that are learnable as well (see, for example, Exercise 3). Later on we will show that what determines the PAC learnability of a class is not its finiteness but rather a combinatorial measure called the VC dimension.

3.2 A More General Learning Model

The model we have just described can be readily generalized, so that it can be made relevant to a wider scope of learning tasks. We consider generalizations in two aspects:

Removing the Realizability Assumption

We have required that the learning algorithm succeeds on a pair of data distribution D and labeling function f provided that the realizability assumption is met. For practical learning tasks, this assumption may be too strong (can we really guarantee that there is a rectangle in the color-hardness space that fully determines which papayas are tasty?). In the next subsection, we will describe the agnostic PAC model in which this realizability assumption is waived.

Learning Problems beyond Binary Classification

The learning task that we have been discussing so far has to do with predicting a binary label to a given example (like being tasty or not). However, many learning tasks take a different form. For example, one may wish to predict a real valued number (say, the temperature at 9:00 p.m. tomorrow) or a label picked from a finite set of labels (like the topic of the main story in tomorrow's paper). It turns out that our analysis of learning can be readily extended to such and many other scenarios by allowing a variety of loss functions. We shall discuss that in Section 3.2.2 later.

3.2.1 Releasing the Realizability Assumption – Agnostic PAC Learning

A More Realistic Model for the Data-Generating Distribution

Recall that the realizability assumption requires that there exists h? ∈ H such thatP [?x D h (x) = f(x)] = 1. In many practical problems this assumption does not hold. Furthermore, it is maybe more realistic not to assume that the labels are fully determined by the features we measure on input elements (in the case of the papayas, it is plausible that two papayas of the same color and softness will have different taste). In the following, we relax the realizability assumption by replacing the "target labeling function" with a more flexible notion, a data-labels generating distribution.

Formally, from now on, let D be a probability distribution over X × Y, where, as before, X is our domain set and Y is a set of labels (usually we will consider Y = {0, 1}). That is, D is a joint distribution over domain points and labels. One can view such a distribution as being composed of two parts: a distribution Dx over unlabeled domain points (sometimes called the marginal distribution) and a conditional probability over labels for each domain point, D((x,y)|x). In the papaya example, Dx determines the probability of encountering a papaya whose color and hardness fall in some color-hardness values domain, and the conditional probability is the probability that a papaya with color and hardness represented by x is tasty. Indeed, such modeling allows for two papayas that share the same color and hardness to belong to different taste categories.

The empirical and the True Error Revised

For a probability distribution D over X × Y, one can measure how likely h is to make an error when labeled points are randomly drawn according to D.We redefine the true error (or risk) of a prediction rule h to be

6 6

def

() = P defL

Dh[h(x)=y]=D({(x,y):h(x)=y}).(3.1)(x,y)

□

We would like to find a predictor, h, for which that error will be minimized. However, the learner does not know the data generating D. What the learner does have access to is the training data, S. The definition of the empirical risk

remains the same as before, namely,

$$|\{i \in [m] : h(x_i) = y_i\}| \quad \text{LS}(h) \stackrel{\text{def}}{=} \frac{6}{m}.$$

Given S, a learner can compute LS(h) for any function $h : X \to \{0,1\}$. Note that LS(h) = LD(uniform over S)(h).

The Goal

We wish to find some hypothesis, $h : X \to Y$, that (probably approximately) minimizes the true risk, LD(h).

The Bayes Optimal Predictor.

Given any probability distribution D over X × {0,1}, the best label predicting function from X to {0,1} will be

$$\{1 \text{ if } P[y=1|x] \geq 1/2 \quad f_D(x) = 0 \text{ otherwise}$$

It is easy to verify (see Exercise 7) that for every probability distribution D, the Bayes optimal predictor f_D is optimal, in the sense that no other classifier, $g : X \to \{0, 1\}$ has a lower error. That is, for every classifier g, $L_D(f_D) \leq L_D(g)$. Unfortunately, since we do not know D, we cannot utilize this optimal predictor f_D. What the learner does have access to is the training sample. We can now present the formal definition of agnostic PAC learnability, which is a natural extension of the definition of PAC learnability to the more realistic, nonrealizable, learning setup we have just discussed.

Clearly, we cannot hope that the learning algorithm will find a hypothesis whose error is smaller than the minimal possible error, that of the Bayes predictor.

Furthermore, as we shall prove later, once we make no prior assumptions about the data-generating distribution, no algorithm can be guaranteed to find a predictor that is as good as the Bayes optimal one. Instead, we require that the learning algorithm will find a predictor whose error is not much larger than the best possible error of a predictor in some given benchmark hypothesis class. Of course, the strength of such a requirement depends on the choice of that hypothesis class.

definition 3.3 (Agnostic PAC Learnability) A hypothesis class H is agnostic PAC learnable if there exist a function $m_H : (0, 1)^2 \to \mathbb{N}$ and a learning algorithm with the following property: For every ε, δ ∈ (0, 1) and for every distribution D over X ×Y, when running the learning algorithm on m ≥ mH(ε, δ) i.i.d. examples generated by D, the algorithm returns a hypothesis h such that, with probability of at least 1 − δ (over the choice of the m training examples),

$$L_{(D,f)}(h) \leq \min_{h' \in H} L_{(D,f)}(h') + \varepsilon.$$

Clearly, if the realizability assumption holds, agnostic PAC learning provides the same guarantee as PAC learning. In that sense, agnostic PAC learning generalizes the definition of PAC learning. When the realizability assumption does not hold, no learner can guarantee an arbitrarily small error. Nevertheless, under the definition of agnostic PAC learning, a learner can still declare success if its error is not much larger than the best error achievable by a predictor from the class H. This is in contrast to PAC learning, in which the learner is required to achieve a small error in absolute terms and not relative to the best error achievable by the hypothesis class.

3.2.2 The Scope of Learning Problems Modeled

We next extend our model so that it can be applied to a wide variety of learning tasks. Let us consider some examples of different learning tasks.

• MulticlassClassificationOurclassificationdoesnothavetobebinary.

Take, for example, the task of document classification: We wish to design a program that will be able to classify given documents according to topics (e.g., news, sports, biology, medicine). A learning algorithm for such a task will have access to examples of correctly classified documents and, on the basis of these examples, should output a program that can take as input a new document and output a topic classification for that document. Here, the domain set is the set of all potential documents. Once again, we would usually represent documents by a set of features that could include counts of different key words in the document, as well as other possibly relevant features like the size of the document or its origin. The label set in this task will be the set of possible document topics (so Y will be some large finite set). Once we determine our domain and label sets, the other components of our framework look exactly the same as in the papaya tasting example; Our training sample will be a finite sequence of (feature vector, label) pairs, the learner's output will be a function from the domain set to the label set, and, finally, for our measure of success, we can use the probability, over (document, topic) pairs, of the event that our predictor suggests a wrong lab el.

• RegressionInthistask,onewishestofindsomesimplepattern inthedata– a functional relationship between the X and Y components of the data. For example, one wishes to find a linear function that best predicts a baby's birth weight on the basis of ultrasound measures of his head circumference, abdominal circumference, and femur length. Here, our domain set X is some subset of R3 (the three ultrasound measurements), and the set of "labels," Y, is the the set of real numbers (the weight in grams). In this context, it is more adequate to call Y the target set. Our training data as well as the learner's output are as before (a finite sequence of (x,y) pairs, and a function from X to Y respectively). However, our measure of success is

different. We may evaluate the quality of a hypothesis function, $h : X \to Y$, by the expected square difference between the true labels and their predicted values, namely,

$$L_D(h) \stackrel{\text{def}}{=} E_{(x,y)\sim D}(h(x) - y)^2. \quad (3.2)$$

To accommodate a wide range of learning tasks we generalize our formalism of the measure of success as follows:

Generalized Loss Functions

Given any set H (that plays the role of our hypotheses, or models) and some domain Z let ℓ be any function from $H \times Z$ to the set of nonnegative real numbers, $\ell : H \times Z \to \mathbb{R}_+$.

We call such functions loss functions.

Note that for prediction problems, we have that $Z = X \times Y$. However, our notion of the loss function is generalized beyond prediction tasks, and therefore it allows Z to be any domain of examples (for instance, in unsupervised learning tasks such as the one described in Chapter 22, Z is not a product of an instance domain and a label domain).

We now define the risk function to be the expected loss of a classifier, $h \in H$, with respect to a probability distribution D over Z, namely,

$$L_D(h) \stackrel{\text{def}}{=} E_{z\sim D}[\ell(h,z)]. \quad (3.3)$$

That is, we consider the expectation of the loss of h over objects z picked randomly according to D. Similarly, we define the empirical risk to be the expected loss over a given sample $S = (z_1,\ldots,z_m) \in Z^m$, namely,

$$L_S(h) \stackrel{\text{def}}{=} \frac{1}{m}\sum_{i=1}^{m} \ell(h,z_i). \quad (3.4)$$

The loss functions used in the preceding examples of classification and regression tasks are as follows:

- 0–1 loss: Here, our random variable z ranges over the set of pairs $X \times Y$ and the loss function is

$$\ell_{0-1}(h,(x,y)) = \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases}$$

This loss function is used in binary or multiclass classification problems. One should note that, for a random variable α, taking the values $\{0, 1\}$, $E_D[\alpha] = P_D[\alpha = 1]$. Consequently, for this loss function, the definitions of $L_D(h)$ given in Equation (3.3) and Equation (3.1) coincide.

- Square loss: Here, our random variable z ranges over the set of pairs $X \times Y$ and the loss function is

$$\ell(h,(x,y)) \stackrel{\text{def}}{=} (h(x) - y)^2.$$

This loss function is used in regression problems.

We will later see more examples of useful instantiations of loss functions.

To summarize, we formally define agnostic PAC learnability for general loss functions.

definition 3.4 (Agnostic PAC Learnability for General Loss Functions) A hypothesis class H is agnostic PAC learnable with respect to a set Z and a loss function ` : H×Z→R, mH,2+ifthereexistafunction:(01)→N and a learning algorithm with the following property: For every ε,δ ∈ (0,1) and for every distribution D over Z, when running the learning algorithm on m ≥ mH(ε,δ) i.i.d. examples generated by D, the algorithm returns h ∈ H such that, with probability of at least 1 − δ (over the choice of the m training examples),

L
D(h)≤minL(h′)+ε,h′D ∈H

where L
D(h) = Ez D[`(h, z)].

Remark 3.1 (A Note About Measurability*) In the aforementioned definition, for every h ∈ H, we view the function `(h, ·) : Z → R+ as a random variable and define L
D(h) to be the expected value of this random variable. For that, we need to require that the function `(h, ·) is measurable. Formally, we assume that there is a σ-algebra of subsets of Z, over which the probability D is defined, and that the preimage of every initial segment in R+ is in this σ-algebra. In the specific case of binary classification with the 0 −1 loss, the σ-algebra is over X × {0,1} and our assumption on ` is equivalent to the assumption that for every h, the set {(x, h(x)) : x ∈ X} is in the σ-algebra.

Remark 3.2 (Proper versus Representation-Independent Learning*) In the preceding definition, we required that the algorithm will return a hypothesis from H. In some situations, H is a subset of a set H′, and the loss function can be naturally extended to be a function from H′ × Z to the reals. In this case, we may allow the algorithm to return a hypothesis h′ ∈ H′, as long as it satisfies the requirement L

3.3 Summary

In this chapter we defined our main formal learning model, PAC learning. The basic model relies on the realizability assumption, while the agnostic variant does D(h) ≤ minh∈H LD(h) + ε. Allowing the algorithm to output a hypothesis from H′ is called representation independent learning, while proper learning occurs when the algorithm must output a hypothesis from H. Representation independent learning is sometimes called “improper learning,” although there is nothing improper in representation independent learning.

not impose any restrictions on the underlying distribution over the examples. We also generalized the PAC model to arbitrary loss functions. We will sometimes refer to the most general model simply as PAC learning, omitting the "agnostic" prefix and letting the reader infer what the underlying loss function is from the context. When we would like to emphasize that we are dealing with the original PAC setting we mention that the realizability assumption holds. In Chapter 7 we will discuss other notions of learnability.

3.4 Bibliographic Remarks

Our most general definition of agnostic PAC learning with general loss func- tions follows the works of Vladimir Vapnik and Alexey Chervonenkis (Vapnik & Chervonenkis 1971). In particular, we follow Vapnik's general setting of learning (Vapnik 1982, Vapnik 1992, Vapnik 1995, Vapnik 1998).

PAC learning was introduced by Valiant (1984). Valiant was named the winner of the 2010 Turing Award for the introduction of the PAC model. Valiant's definition requires that the sample complexity will be polynomial in $1/\epsilon$ and in $1/\delta$, as well as in the representation size of hypotheses in the class (see also Kearns & Vazirani (1994)). As we will see in Chapter 6, if a problem is at all PAC learnable then the sample complexity depends polynomially on $1/\epsilon$ and $\log(1/\delta)$. Valiant's definition also requires that the runtime of the learning algorithm will be polynomial in these quantities. In contrast, we chose to distinguish between the statistical aspect of learning and the computational aspect of learning. We will elaborate on the computational aspect later on in Chapter 8, where we introduce the full PAC learning model of Valiant. For expository reasons, we use the term PAC learning even when we ignore the runtime aspect of learning. Finally, the formalization of agnostic PAC learning is due to Haussler (1992).

3.5 Exercises

1. Monotonicity of Sample Complexity: Let H be a hypothesis class for a binary classification task. Suppose that H is PAC learnable and its sample complexity is given by $m_H(\cdot, \cdot)$. Show that m_H is monotonically nonincreasing in each of its parameters. That is, show that given $\delta \in (0, 1)$, and given $0 < \epsilon_1 \leq \epsilon_2 < 1$, we have that $m_H(\epsilon_1,\delta) \geq m_H(\epsilon_2,\delta)$. Similarly, show that given $\epsilon \in (0, 1)$, and given $0 < \delta_1 \leq \delta_2 < 1$, we have that $m_H(\epsilon, \delta_1) \geq m_H(\epsilon, \delta_2)$.

2. Let X be a discrete domain, and let $H = \{h_z \in X\} \cup \{h^- \text{Singleton} = z: \}$, where for each $z \in X$, h_z is the function defined by $h_z(x) = 1$ if $x = z$ and $h_z(x) = 0$ if $x \neq z$. h^- is simply the all-negative hypothesis, namely, $\forall x \in X$, $h^-(x) = 0$. The realizability assumption here implies that the true hypothesis f labels negatively all examples in the domain, perhaps except one.

1. Describe an algorithm that implements the ERM rule for learning $H_{Singleton}$ in the realizable setup.
2. Show that $H_{Singleton}$ is PAC learnable. Provide an upper bound on the sample complexity.
3. Let $X = R^2$, $Y = \{0,1\}$, and let H be the class of concentric circles in the plane, that is, $H = \{h_r : r \in \mathbb{R}_+\}$, where $h_r(x) = \mathbb{1}[\|x\| \leq r]$. Prove that H is PAC learnable (assume realizability), and its sample complexity is bounded by

$$\lceil \rceil \leq \log(1/\delta) m_H(\varepsilon,\delta).\varepsilon$$

4. In this question, we study the hypothesis class of Boolean conjunctions defined as follows. The instance space is $X = \{0, 1\}^d$ and the label set is $Y = \{0, 1\}$. A literal over the variables $x_1, \ldots, x_d$ is a simple Boolean function that takes the form $f(x) = x_i$, for some $i \in [d]$, or $f(x) = 1 - x_i$ for some $i \in [d]$. We use the notation $\bar{x}_i$ as a shorthand for $1 - x_i$. A conjunction is any product of literals. In Boolean logic, the product is denoted using the $\wedge$ sign. For example, the function $h(x) = x_1 \cdot (1 - x_2)$ is written as $x_1 \wedge \bar{x}_2$.

We consider the hypothesis class of all conjunctions of literals over the d variables. The empty conjunction is interpreted as the all-positive hypothesis (namely, the function that returns $h(x) = 1$ for all x). The conjunction $x_1 \wedge \bar{x}_1$ (and similarly any conjunction involving a literal and its negation) is allowed and interpreted as the all-negative hypothesis (namely, the conjunction that returns $h(x) = 0$ for all x). We assume realizability: Namely, we assume that there exists a Boolean conjunction that generates the labels. Thus, each example $(x,y) \in X \times Y$ consists of an assignment to the d Boolean variables $x_1, \ldots, x_d$, and its truth value (0 for false and 1 for true).

For instance, let $d = 3$ and suppose that the true conjunction is $x_1 \wedge \bar{x}_2$. Then, the training set S might contain the following instances:

$$((1, 1, 1), 0), ((1, 0, 1), 1), ((0, 1, 0), 0)((1, 0, 0), 1).$$

Prove that the hypothesis class of all conjunctions over d variables is PAC learnable and bound its sample complexity. Propose an algorithm that implements the ERM rule, whose runtime is polynomial in $d \cdot m$.

5. Let X be a domain and let $D_1, D_2, \ldots, D_m$ be a sequence of distributions over X. Let H be a finite class of binary classifiers over X and let $f \in H$. Suppose we are getting a sample S of m examples, such that the instances are independent but are not identically distributed; the ith instance is sampled from D_i and then y_i is set to be $f(x_i)$. Let $\bar{D}_m$ denote the average, that is, $\bar{D}_m = (D_1 + \cdots + D_m)/m$.

Fix an $\exists \in H \quad \leq |Hr|$

Hint: Use the geometric-arithmetic mean inequality.

6. Let H be a hypothesis class of binary classifiers. Show that if H is agnostic PAC learnable, then H is PAC learnable as well. Furthermore, if A is a successful agnostic PAC learner for H, then A is also a successful PAC learner for H.

7. (*) The Bayes optimal predictor: Show that for every probability distribution D, the Bayes optimal predictor f_D is optimal, in the sense that for every classifier g from X to $\{0,1\}$, $L_D(f_D) \leq L_D(g)$.

8. (*) We say that a learning algorithm A is better than B with respect to some probability distribution, D, if

$$L_D(A(S)) \leq L_D(B(S))$$

for all samples $S \in (X \times \{0,1\})^m$. We say that a learning algorithm A is better than B, if it is better than B with respect to all probability distributions D over $X \times \{0,1\}$.

1. A probabilistic label predictor is a function that assigns to every domain point x a probability value, $h(x) \in [0,1]$, that determines the probability of predicting the label 1. That is, given such an h and an input, x, the label for x is predicted by tossing a coin with bias $h(x)$ toward Heads and predicting 1 iff the coin comes up Heads. Formally, we define a probabilistic label predictor as a function, $h : X \to [0,1]$. The loss of such h on an example (x,y) is defined to be $|h(x) - y|$, which is exactly the probability that the prediction of h will not be equal to y. Note that if h is deterministic, that is, returns values in $\{0,1\}$, then $|h(x) - y| = 1[h(x) \neq y]$.
Prove that for every data-generating distribution D over $X \times \{0,1\}$, the Bayes optimal predictor has the smallest risk (w.r.t. the loss function $\ell(h,(x,y)) = |h(x)-y|$, among all possible label predictors, including probabilistic ones).

2. Let X be a domain and $\{0,1\}$ be a set of labels. Prove that for every distribution D over $X \times \{0,1\}$, there exist a learning algorithm A_D that is better than any other learning algorithm with respect to D.

3. Prove that for every learning algorithm A there exist a probability distribution, D, and a learning algorithm B such that A is not better than B w.r.t. D.

9. Consider a variant of the PAC model in which there are two example oracles: one that generates positive examples and one that generates negative examples, both according to the underlying distribution D on X. Formally, given a target function $f : X \to \{0,1\}$, let D^+ be the distribution over

of the two training sets, and possibly over the nondeterministic decisions made by the learning algorithm), both $L(D+,f)(h) \leq \varepsilon$ and $L(D-,f)(h) \leq \varepsilon$.

1. (*) Show that if

H is PAC learnable (in the standard one-oracle model),

then

H is PAC learnable in the two-oracle model.

2. (**) Define h+ to be the always-plus hypothesis and h− to be the always-minus hypothesis. Assume that h+, h− $\in$ H. Show that if H is PAC learnable in the two-oracle model, then

H is PAC learnable in the standard one-oracle model.

4 Learning via Uniform Convergence

The first formal learning model that we have discussed was the PAC model. In Chapter 2 we have shown that under the realizability assumption, any finite hypothesis class is PAC learnable. In this chapter we will develop a general tool, uniform convergence, and apply it to show that any finite class is learnable in the agnostic PAC model with general loss functions, as long as the range loss function is bounded.

4.1 Uniform Convergence Is Sufficient for Learnability

The idea behind the learning condition discussed in this chapter is very simple. Recall that, given a hypothesis class, H, the ERM learning paradigm works as follows: Upon receiving a training sample, S, the learner evaluates the risk (or error) of each h in H on the given sample and outputs a member of H that minimizes this empirical risk. The hope is that an h that minimizes the empirical risk with respect to S is a risk minimizer (or has risk close to the minimum) with respect to the true data probability distribution as well. For that, it suffices to ensure that the empirical risks of all members of H are good approximations of their true risk. Put another way, we need that uniformly over all hypotheses in the hypothesis class, the empirical risk will be close to the true risk, as formalized in the following.

definition 4.1 (ε-representative sample) A training set S is called ε-representative (w.r.t. domain Z, hypothesis class H, loss function ` , and distribution D) if

$$\forall h \in H, \; |L_S(h) - L_D(h)| \leq \varepsilon .$$

The next simple lemma states that whenever the sample is (ε/2)-representative, the ERM learning rule is guaranteed to return a good hypothesis.

lemma 4.2 Assume that a training set S is ε 2-representative (w.r.t. domain Z, hypothesis class H, loss function ` , and distribution D). Then, any output of ERM H(S), namely, any $h_S \in \operatorname{argmin}_{h \in H} L_S(h)$, satisfies

$$L_D(h_S) \leq \min_{h \in H} L_D(h) + \varepsilon .$$

Proof Foreveryh ∈H,

L(h)L(h)+εL(h)+ε εε – – – –

D S≤ S S 2≤ S 2≤LD(h)+2+2=LD(h)+ε,

where the first and third inequalities are due to the assumption that S is ε –

representative (Definition 4.1) and the second inequality holds since hS is an ERM predictor. 2□

The preceding lemma implies that to ensure that the ERM rule is an agnostic PAC learner, it suffices to show that with probability of at least 1 – δ over the random choice of a training set, it will be an ε-representative training set. The uniform convergence condition formalizes this requirement.

definition 4.3 (Uniform Convergence) We say that a hypothesis class Hhas the uniform convergence property (w.r.t. a domain Z and a loss function `) if there exists a function mUC : (0,1)2 H →Nsuchthatforeveryε,δ∈(0,1)and for every probability distribution D over Z, if S is a sample of m ≥ mUCH (ε,δ) examples drawn i.i.d. according to D, then, with probability of at least 1 – δ, S is ε-representative.

Similar to the definition of sample complexity for PAC learning, the function mUC H measures the (minimal) sample complexity of obtaining the uniform convergence property, namely, how many examples we need to ensure that with probability of at least 1 – δ the sample would be ε-representative.

The term uniform here refers to having a fixed sample size that works for all members of H and over all possible probability distributions over the domain. The following corollary follows directly from Lemma 4.2 and the definition of uniform convergence.

Corollary 4.4 If H has the uniform convergence property with a function mUC H then the class is agnostically PAC learnable with the sample complexity m (ε,δ ≤mUCHε/,δ.Furthermore,inthatcase,theHparadigm is a successful agnostic PAC learner for H (2) ERM

4.2 Finite Classes Are Agnostic PAC Learnable

In view of Corollary 4.4, the claim that every finite hypothesis class is agnostic PAC learnable will follow once we establish that uniform convergence holds for a finite hypothesis class.

To show that uniform convergence holds we follow a two step argument, similar to the derivation in Chapter 2. The first step applies the union bound while the second step employs a measure concentration inequality. We now explain these two steps in detail.

Fix some ε,δ. We need to find a sample size m that guarantees that for any D, with probability of at least 1 – δ of the choice of S = (z1,...,zm) sampled

i.i.d from
D we have that for all h ∈ H, |LS(h) − LD(h)| ≤ ε. That is,

$$D^m(\{S : \forall h \in H, |L_S(h) - L_D(h)| \leq \epsilon\}) \geq 1 - \delta.$$

Equivalently, we need to show that

$$D^m(\{S : \exists h \in H, |L_S(h) - L_D(h)| > \epsilon\}) < \delta.$$

Writing

$$\{S : \exists h \in H, |L_S(h) - L_D(h)| > \epsilon\} = \bigcup_{h \in H} \{S : |L_S(h) - L_D(h)| > \epsilon\},$$

and applying the union bound (Lemma 2.2)

$$\sum_{h \in H} D^m(\{S : |L_S(h) - L_D(h)| > \epsilon\}) \geq D^m(\{S : \exists h \in H, |L_S(h) - L_D(h)| > \epsilon\}). \tag{4.1}$$

Our second step will be to argue that each summand of the right-hand side of this inequality is small enough (for a sufficiently large m). That is, we will show that for any fixed hypothesis, h, (which is chosen in advance prior to the sampling of the training set), the gap between the true and empirical risks, |LS(h) − LD(h)|, is likely to be small.

$$\bar{\theta} = \frac{1}{m} \sum_{i=1}^{m} \theta_i \text{ and } L_D(h) = \mu.$$

Let us

D(h). By the linearity of expectation, it follows that LD(h) is also the expected value of LS(h). Hence, the quantity

|LD(h)−LS(h)| is the deviation of the random variable LS(h) from its expectation. We therefore need to show that the measure of LS(h) is concentrated around its expected value.

A basic statistical fact, the law of large numbers, states that when m goes to infinity, empirical averages converge to their true expectation. This is true for LS(h), since it is the empirical average of m i.i.d random variables. However, since the law of large numbers is only an asymptotic result, it provides no information about the gap between the empirically estimated error and its true value for any given, finite, sample size.

Instead, we will use a measure concentration inequality due to Hoeffding, which quantifies the gap between empirical averages and their expected value.

lemma 4.5 (Hoeffding's Inequality) Let θ1,...,θm be a sequence of i.i.d. random variables and assume that for all i, E[θi] = μ and P[a ≤ θi ≤ b] = 1. Then, for any ε > 0

$$P\left[\left|\frac{1}{m}\sum_{i=1}^{m}\theta_i - \mu\right| > \epsilon\right] \leq 2\exp(-2m\epsilon^2/(b-a)^2).$$

The proof can be found in Appendix B.

Getting back to our problem, let θi be the random variable `(h,zi). Since h is fixed and z1, . . . , zm are sampled i.i.d., it follows that θ1, . . . , θm are also i.i.d. random variables. Furthermore, LS(h) =

further assume that the range of ` is [0, 1] and therefore θi ∈[0, 1]. We therefore obtain that

$$D^m(\{S:|L_S(h)-L_D(h)|>\varepsilon\}) = P\left[\left|\theta_i - \mu\right| > \varepsilon\right] \leq 2\exp\left(-2m\varepsilon^2\right). \tag{4.2}$$

Combining this with Equation (4.1) yields

$$\exists h \in H,\ |L_S(h)-L_D(h)|>\varepsilon \leq \sum_{h\in H} 2\exp\left(-2m\varepsilon^2\right) = 2|H|\exp\left(-2m\varepsilon^2\right).$$

Finally, if we choose

$$m \geq \frac{\log(2|H|/\delta)}{2\varepsilon^2}$$

then

$$D^m(\{S:\exists h\in H,\ |L_S(h) - L_D(h)|>\varepsilon\}) \leq \delta.$$

corollary4.6 Let beafinitehypothesisclass,let beadomain,andlet

` : H × Z → [0,1] be a loss function. Then, H enjoys the uniform convergence property with sample complexity

$$m_H^{UC}(\varepsilon,\delta) \leq \left\lceil \frac{\log(2|H|/\delta)}{2\varepsilon^2} \right\rceil.$$

Furthermore, the class is agnostically PAC learnable using the ERM algorithm with sample complexity

⌈ ⌉≤2log(2|H|/δ)mε,δmUCH()H(ε/2,δ)≤.ε2 _____

Remark 4.1 (The “Discretization Trick”) While the preceding corollary only applies to finite hypothesis classes, there is a simple trick that allows us to get a very good estimate of the practical sample complexity of infinite hypothesis classes. Consider a hypothesis class that is parameterized by d parameters. For example, let

X = R, Y = {±1}, and the hypothesis class, H, be all functions of the form hθ(x) = sign(x

− θ). That is, each hypothesis is parameterized by one parameter, θ

∈ R, and the hypothesis outputs 1 for all instances larger than θ and outputs

−1 for instances smaller than θ. This is a hypothesis class of an infinite size. However, if we are going to learn this hypothesis class in practice, using a computer, we will probably maintain real numbers using floating point representation, say, of 64 bits. It follows that in practice, our hypothesis class is parameterized by the set of scalars that can be represented using a 64 bits floating point number. There are at most 264 such numbers; hence the actual size of our hypothesis class is at most 264. More generally, if our hypothesis class is parameterized by d numbers, in practice we learn a hypothesis class of size at most 264d. Applying Corollary 4.6 we obtain that the sample complexity of such

classes is bounded by $\frac{128d+2\log(2/\delta)}{\epsilon^2}$. This upper bound on the sample complexity has the deficiency of being dependent on the specific representation of real numbers used by our machine. In Chapter 6 we will introduce a rigorous way to analyze the sample complexity of infinite size hypothesis classes. Nevertheless, the discretization trick can be used to get a rough estimate of the sample complexity in many practical situations.

4.3 Summary

If the uniform convergence property holds for a hypothesis class H then in most cases the empirical risks of hypotheses in H will faithfully represent their true risks. Uniform convergence suffices for agnostic PAC learnability using the ERM rule. We have shown that finite hypothesis classes enjoy the uniform convergence property and are hence agnostic PAC learnable.

4.4 Bibliographic Remarks

Classes of functions for which the uniform convergence property holds are also called Glivenko-Cantelli classes, named after Valery Ivanovich Glivenko and Francesco Paolo Cantelli, who proved the first uniform convergence result in the 1930s. See (Dudley, Gine & Zinn 1991). The relation between uniform convergence and learnability was thoroughly studied by Vapnik – see (Vapnik 1992, Vapnik 1995, Vapnik 1998). In fact, as we will see later in Chapter 6, the fundamental theorem of learning theory states that in binary classification problems, uniform convergence is not only a sufficient condition for learnability but is also a necessary condition. This is not the case for more general learning problems (see (Shalev-Shwartz, Shamir, Srebro & Sridharan 2010)).

4.5 Exercises

1. In this exercise, we show that the (ϵ,δ) requirement on the convergence of errors in our definitions of PAC learning, is, in fact, quite close to a simpler looking requirement about averages (or expectations). Prove that the following two statements are equivalent (for any learning algorithm A, any probability distribution D, and any loss function whose range is $[0, 1]$):

1. For every $\epsilon,\delta > 0$, there exists $m(\epsilon,\delta)$ such that $\forall m \geq m(\epsilon, \delta)$ $P[L_{D}(A(S))>\epsilon]<\delta$

2. $\lim_{m\to\infty} \mathbb{E}_{S\sim D^m}[L_D(A(S))] = 0$

(where E_S denotes the expectation over samples S of size m).

2. Bounded loss functions: In Corollary 4.6 we assumed that the range of the loss function is [0, 1]. Prove that if the range of the loss function is [a, b] then the sample complexity satisfies

$$\lceil \rceil 2\log(2)()()\leq\leq|H|b-a2mUCH\varepsilon,\delta mH(\varepsilon/2,\delta).2 \quad /\delta \quad \varepsilon$$

5 The Bias-Complexity Tradeoff

In Chapter 2 we saw that unless one is careful, the training data can mislead the learner, and result in overfitting. To overcome this problem, we restricted the search space to some hypothesis class $\mathcal{H}$. Such a hypothesis class can be viewed as reflecting some prior knowledge that the learner has about the task – a belief that one of the members of the class $\mathcal{H}$ is a low-error model for the task. For example, in our papayas taste problem, on the basis of our previous experience with other fruits, we may assume that some rectangle in the color-hardness plane predicts (at least approximately) the papaya's tastiness.

Is such prior knowledge really necessary for the success of learning? Maybe there exists some kind of universal learner, that is, a learner who has no prior knowledge about a certain task and is ready to be challenged by any task? Let us elaborate on this point. A specific learning task is defined by an unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, where the goal of the learner is to find a predictor $h : \mathcal{X} \to \mathcal{Y}$, whose risk, $L_{\mathcal{D}}(h)$, is small enough. The question is therefore whether there exist a learning algorithm A and a training set size m, such that for every distribution $\mathcal{D}$, if A receives m i.i.d. examples from $\mathcal{D}$, there is a high chance it outputs a predictor h that has a low risk.

The first part of this chapter addresses this question formally. The No-Free-Lunch theorem states that no such universal learner exists. To be more precise, the theorem states that for binary classification prediction tasks, for every learner there exists a distribution on which it fails. We say that the learner fails if, upon receiving i.i.d. examples from that distribution, its output hypothesis is likely to have a large risk, say, ≥ 0.3, whereas for the same distribution, there exists another learner that will output a hypothesis with a small risk. In other words, the theorem states that no learner can succeed on all learnable tasks – every learner has tasks on which it fails while other learners succeed.

Therefore, when approaching a particular learning problem, defined by some distribution $\mathcal{D}$, we should have some prior knowledge on $\mathcal{D}$. One type of such prior knowledge is that $\mathcal{D}$ comes from some specific parametric family of distributions. We will study learning under such assumptions later on in Chapter 24. Another type of prior knowledge on $\mathcal{D}$, which we assumed when defining the PAC learning model, is that there exists h in some predefined hypothesis class $\mathcal{H}$, such that $L_{\mathcal{D}}(h) = 0$. A softer type of prior knowledge on $\mathcal{D}$ is assuming that $\min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$ is small. In a sense, this weaker assumption on $\mathcal{D}$ is a prerequisite for using the

agnostic PAC model, in which we require that the risk of the output hypothesis will not be much larger than $\min_{h \in H} L_D(h)$.

In the second part of this chapter we study the benefits and pitfalls of using a hypothesis class as a means of formalizing prior knowledge. We decompose the error of an ERM algorithm over a class H into two components. The first component reflects the quality of our prior knowledge, measured by the minimal risk of a hypothesis in our hypothesis class, $\min_{h \in H} L_D(h)$. This component is also called the approximation error, or the bias of the algorithm toward choosing a hypothesis from H. The second component is the error due to overfitting, which depends on the size or the complexity of the class H and is called the estimation error. These two terms imply a tradeoff between choosing a more complex H (which can decrease the bias but increases the risk of overfitting) or a less complex H (which might increase the bias but decreases the potential overfitting).

5.1 The No-Free-Lunch Theorem

In this part we prove that there is no universal learner. We do this by showing that no learner can succeed on all learning tasks, as formalized in the following theorem:

theorem 5.1 (No-Free-Lunch) Let A be any learning algorithm for the task of binary classification with respect to the $0-1$ loss over a domain X. Let m be any number smaller than $|X|/2$, representing a training set size. Then, there exists a distribution D over $X \times \{0,1\}$ such that:

1. There exists a function $f : X \to \{0, 1\}$ with $L_D(f) = 0$.
2. With probability of at least $1/7$ over the choice of $S \sim D^m$ we have that $L_D(A(S)) \geq 1/8$.

This theorem states that for every learner, there exists a task on which it fails, even though that task can be successfully learned by another learner. Indeed, a trivial successful learner in this case would be an ERM learner with the hypothesis class $H = \{f\}$, or more generally, ERM with respect to any finite hypothesis class that contains f and whose size satisfies the equation $m \geq 8 \log(7|H|/6)$ (see Corollary 2.3).

Proof Let C be a subset of X of size $2m$. The intuition of the proof is that any learning algorithm that observes only half of the instances in C has no information on what should be the labels of the rest of the instances in C. Therefore, there exists a "reality," that is, some target function f, that would contradict the labels that $A(S)$ predicts on the unobserved instances in C.

Note that there are $T = 2^{2m}$ possible functions from C to $\{0, 1\}$. Denote these functions by $f_1,\ldots,f_T$. For each such function, let D_i be a distribution over

C × { 0 1 } defined by

{1/|C|ify=f(x)D{x,y}ii(())=0otherwise.

That is, the probability to choose a pair (x,y) is 1/

|C| if the label y is indeed the true label according to fi, and the probability is 0 if y

6= fi(x). Clearly, L

D(f)=0.ii

We will show that for every algorithm, A, that receives a training set of m examples from C ×{0, 1} and returns a function A(S) : C → {0, 1}, it holds that max E [L

D(A(S))]≥1/4. (5.1)

∈ ii[T]S Dmi

Clearly, this means that for every algorithm, A′, that receives a training set of m examples from X ×{0, 1} there exist a function f : X → {0, 1} and a distribution D over X × {0, 1}, such that LD(f) = 0 and

E [L A′

D((S))]≥1/4. (5.2)S

Dm

It is easy to verify that the preceding suffices for showing that P[L A′

D((S))≥ 1/8]

≥ 1/7, which is what we need to prove (see Exercise 1).

We now turn to proving that Equation (5.1) holds. There are k = (2m)m possible sequences of m examples from C. Denote these sequences by S1, . . . , Sk. Also,ifS=(x,...,x)w Sij 1 medenoteby

F S jthe sequence containing the instances inSlabeledby thefunctiofSij ni,namely,

the distribution is j = ((x1, fi(x1)), . . . , (xm, fi(xm))). If

Using the fact that “maximum” is larger than “average” and that “average” is larger than “minimum,” we have Dthent A Si,...,Siihepossibletrainingsetscanreceiveare1k, and all these training sets have the same probability of being sampled. Therefore,

1 k

∑kTk11maxL(A(Si))≥L((SiDA))∈kijDTkij=1=1=1

2 [L(A(S))]= L(A(Sij)). (5.3)

S k j

1 ∑ijkT1∑LDT=1

k — — i (A(Sj i))

∑i=1T1min

≥ L iD((— (5.4)

∈ iASj)).j[k]Ti=1

Next, fix some j ∈ [k]. Denote Sj = (x1,...,xm v1,...,vp)andlet bethe examples in C that do not appear in . Clearly, Sip, . Therefore, for every

function $h : C \to \{0,1\}$ and every i we have

$$\begin{aligned} D(h) &= \frac{1}{2m}\sum_{x\in C} [h(x) = f_i(x)] \\ &\geq \frac{1}{2m}\sum_{r=1}^{p} [h(v_r) = f_i(v_r)] \\ &\geq \frac{1}{2p}\sum_{r=1}^{p} [h(v_r) = f_i(v_r)] . \end{aligned} \tag{5.5}$$

Hence,

$$\begin{aligned} &\frac{1}{T}\sum_{i=1}^{T} L_D(A(S_j)) \geq \frac{1}{2T}\sum_{i=1}^{T}\sum_{r=1}^{p} [A(S_i)(v_r) = f_i(v_r)] \\ &= \frac{1}{2}\sum_{r=1}^{p}\frac{1}{T}\sum_{i=1}^{T} [A(S_i)(v_r) = f_i(v_r)] \\ &\geq \frac{1}{2}\cdot \min_{r\in[p]}\frac{1}{T}\sum_{i=1}^{T} [A(S_i)(v_r) = f_i(v_r)] . \end{aligned} \tag{5.6}$$

Next, fix some $r \in [p]$. We can partition all the functions in $f_1, \ldots, f_T$ into $T/2$ disjoint pairs, where for a pair $(f_i, f_{i'})$ we have that for every $c \in C$, $f_i(c) \neq f_{i'}(c)$ and $f_i(v_r) \neq f_{i'}(v_r)$. Since for such a pair we must have $S_i = S_{i'}$, it follows that

$$\frac{1}{T}\sum_{i=1}^{T} [A(S_i)(v_r) = f_i(v_r)] = \frac{1}{2}\cdot 1 + \frac{1}{2}\cdot 0 = \frac{1}{2} .$$

Combining this with Equation (5.6), Equation (5.4), and Equation (5.3), we obtain that Equation (5.1) holds, which concludes our proof. □

5.1.1 No-Free-Lunch and Prior Knowledge

How does the No-Free-Lunch result relate to the need for prior knowledge? Let us consider an ERM predictor over the hypothesis class H of all the functions f from X to $\{0, 1\}$. This class represents lack of prior knowledge: Every possible function from the domain to the label set is considered a good candidate. According to the No-Free-Lunch theorem, any algorithm that chooses its output from hypotheses in H, and in particular the ERM predictor, will fail on some learning task. Therefore, this class is not PAC learnable, as formalized in the following corollary:

corollary 5.2 Let X be an infinite domain set and let H be the set of all functions from X to $\{0,1\}$. Then, H is not PAC learnable.

Proof Assume,bywayofcontradiction,thattheclassislearnable.Choose some $\varepsilon < 1/8$ and $\delta < 1/7$. By the definition of PAC learnability, there must be some learning algorithm A and an integer $m = m(\varepsilon,\delta)$, such that for any data-generating distribution over $X \times \{0, 1\}$, if for some function $f : X \to \{0, 1\}$, $L_D(f) = 0$, then with probability greater than $1 - \delta$ when A is applied to samples S of size m, generated i.i.d. by D, $L_D(A(S)) \leq \varepsilon$. However, applying the No-Free-Lunch theorem, since $|X| > 2m$, for every learning algorithm (and in particular for the algorithm A), there exists a distribution D such that with probability greater than $1/7 > \delta$, $L_D(A(S)) > 1/8 > \varepsilon$, which leads to the desired contradiction.

How can we prevent such failures? We can escape the hazards foreseen by the No-Free-Lunch theorem by using our prior knowledge about a specific learning task, to avoid the distributions that will cause us to fail when learning that task. Such prior knowledge can be expressed by restricting our hypothesis class.

But how should we choose a good hypothesis class? On the one hand, we want to believe that this class includes the hypothesis that has no error at all (in the PAC setting), or at least that the smallest error achievable by a hypothesis from this class is indeed rather small (in the agnostic setting). On the other hand, we have just seen that we cannot simply choose the richest class – the class of all functions over the given domain. This tradeoff is discussed in the following section.

5.2 Error Decomposition

To answer this question we decompose the error of an ERM$_H$ predictor into two components as follows. Let h_S be an ERM$_H$ hypothesis. Then, we can write

$$L_D(h_S) = \varepsilon_{app} + \varepsilon_{est} \quad \text{where:} \quad \varepsilon_{app} = \min_{h \in H} L_D(h), \quad \varepsilon_{est} = L_D(h_S) - \varepsilon_{app}. \tag{5.7}$$

- The Approximation Error – the minimum risk achievable by a predictor in the hypothesis class. This term measures how much risk we have because we restrict ourselves to a specific class, namely, how much inductive bias we have. The approximation error does not depend on the sample size and is determined by the hypothesis class chosen. Enlarging the hypothesis class can decrease the approximation error.

  Under the realizability assumption, the approximation error is zero. In the agnostic case, however, the approximation error can be large.[1]

[1] In fact, it always includes the error of the Bayes optimal predictor (see Chapter 3), the minimal yet inevitable error, because of the possible nondeterminism of the world in this model. Sometimes in the literature the term approximation error refers not to $\min_{h \in H} L(h)$, but rather to the excess error over that of the Bayes optimal predictor, namely $\min_{h \in H} L(h) - \varepsilon_{Bayes}$.

• The Estimation Error – the difference between the approximation error and the error achieved by the ERM predictor. The estimation error results because the empirical risk (i.e., training error) is only an estimate of the true risk, and so the predictor minimizing the empirical risk is only an estimate of the predictor minimizing the true risk.

The quality of this estimation depends on the training set size and on the size, or complexity, of the hypothesis class. As we have shown, for a finite hypothesis class, ϵ_{est} increases (logarithmically) with $|H|$ and decreases with m. We can think of the size of H as a measure of its complexity. In future chapters we will define other complexity measures of hypothesis classes.

Since our goal is to minimize the total risk, we face a tradeoff, called the bias-complexity tradeoff. On one hand, choosing H to be a very rich class decreases the approximation error but at the same time might increase the estimation error, as a rich H might lead to overfitting. On the other hand, choosing H to be a very small set reduces the estimation error but might increase the approximation error or, in other words, might lead to underfitting. Of course, a great choice for H is the class that contains only one classifier – the Bayes optimal classifier. But the Bayes optimal classifier depends on the underlying distribution D, which we do not know (indeed, learning would have been unnecessary had we known D). Learning theory studies how rich we can make H while still maintaining reasonable estimation error. In many cases, empirical research focuses on designing good hypothesis classes for a certain domain. Here, "good" means classes for which the approximation error would not be excessively high. The idea is that although we are not experts and do not know how to construct the optimal classifier, we still have some prior knowledge of the specific problem at hand, which enables us to design hypothesis classes for which both the approximation error and the estimation error are not too large. Getting back to our papayas example, we do not know how exactly the color and hardness of a papaya predict its taste, but we do know that papaya is a fruit and on the basis of previous experience with other fruit we conjecture that a rectangle in the color-hardness space may be a good predictor.

5.3 Summary

The No-Free-Lunch theorem states that there is no universal learner. Every learner has to be specified to some task, and use some prior knowledge about that task, in order to succeed. So far we have modeled our prior knowledge by restricting our output hypothesis to be a member of a chosen hypothesis class. When choosing this hypothesis class, we face a tradeoff, between a larger, or more complex, class that is more likely to have a small approximation error, and a more restricted class that would guarantee that the estimation error will

be small. In the next chapter we will study in more detail the behavior of the estimation error. In Chapter 7 we will discuss alternative ways to express prior knowledge.

5.4 Bibliographic Remarks

(Wolpert & Macready 1997) proved several no-free-lunch theorems for optimization, but these are rather different from the theorem we prove here. The theorem we prove here is closely related to lower bounds in VC theory, as we will study in the next chapter.

5.5 Exercises

1. Prove that Equation (5.2) suffices for showing that P[L D(A(S)) ≥ 1/8] ≥ 1/7.
Hint: Let θ be a random variable that receives values in [0,1] and whose expectation satisfies E[θ] ≥ 1/4. Use Lemma B.1 to show that P[θ ≥ 1/8] ≥ 1/7.
2. Assume you are asked to design a learning algorithm to predict whether patients are going to suffer a heart attack. Relevant patient features the algorithm may have access to include blood pressure (BP), body-mass index (BMI), age (A), level of physical activity (P), and income (I).
You have to choose between two algorithms; the first picks an axis aligned rectangle in the two dimensional space spanned by the features BP and BMI and the other picks an axis aligned rectangle in the five dimensional space spanned by all the preceding features.
1. Explain the pros and cons of each choice.
2. Explain how the number of available labeled training samples will affect your choice.
3. Prove that if |X| ≥ km for a positive integer k ≥ 2, then we can replace the lower bound of 1/4 in the No-Free-Lunch theorem with k−1 1 −12k=22k.
Namely, let A be a learning algorithm for the task of binary classification. Let m be any number smaller than |X| /k, representing a training set size. Then, there exists a distribution over D X × { } X → { } such that: ≥ .
Thereexistsafunctionf: 0,1withL
D(f) = 0.
E[(())]11S
DmLDAS 2 2k.

6 The VC-Dimension

In the previous chapter, we decomposed the error of the ERM $_H$ rule into approximation error and estimation error. The approximation error depends on the fit of our prior knowledge (as reflected by the choice of the hypothesis class H) to the underlying unknown distribution. In contrast, the definition of PAC learnability requires that the estimation error would be bounded uniformly over all distributions.

Our current goal is to figure out which classes H are PAC learnable, and to characterize exactly the sample complexity of learning a given hypothesis class. So far we have seen that finite classes are learnable, but that the class of all functions (over an infinite size domain) is not. What makes one class learnable and the other unlearnable? Can infinite-size classes be learnable, and, if so, what determines their sample complexity?

We begin the chapter by showing that infinite classes can indeed be learnable, and thus, finiteness of the hypothesis class is not a necessary condition for learnability. We then present a remarkably crisp characterization of the family of learnable classes in the setup of binary valued classification with the zero-one loss. This characterization was first discovered by Vladimir Vapnik and Alexey Chervonenkis in 1970 and relies on a combinatorial notion called the Vapnik-Chervonenkis dimension (VC-dimension). We formally define the VC-dimension, provide several examples, and then state the fundamental theorem of statistical learning theory, which integrates the concepts of learnability, VC-dimension, the ERM rule, and uniform convergence.

6.1 Infinite-Size Classes Can Be Learnable

In Chapter 4 we saw that finite classes are learnable, and in fact the sample complexity of a hypothesis class is upper bounded by the log of its size. To show that the size of the hypothesis class is not the right characterization of its sample complexity, we first present a simple example of an infinite-size hypothesis class that is learnable.

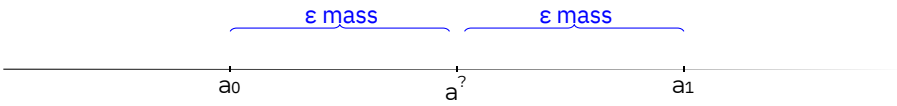
Example 6.1 Let H be the set of threshold functions over the real line, namely, $H = \{h_a : a \in \mathbb{R}\}$, where $h_a : \mathbb{R} \to \{0, 1\}$ is a function such that $h_a(x) = 1_{[x<a]}$. To remind the reader, $1_{[x<a]}$ is 1 if $x < a$ and 0 otherwise. Clearly, H is of infinite

size. Nevertheless, the following lemma shows that H is learnable in the PAC model using the ERM algorithm.

Lemma 6.1 Let H be the class of thresholds as defined earlier. Then, H is PAC learnable, using the ERM rule, with sample complexity of $m_H(\varepsilon,\delta) \leq$ $\lceil \log(2/\delta)/\varepsilon \rceil$.

Proof a Let ?beathresholdsuchthatthehypothesish?(x)=1[x<a?] achieves and let D)=0.LetDxbethemarginaldistributionoverthedomain X $a_0 < a < a_1$ besuchthat

P[x $\in(a,a?0)]=P[x\in(a?,a1)]=\varepsilon$.xxxx



(If D ?x(−∞,a)≤εweseta0=−∞andsimilarlyfora1).Givenatrainingset S, let b0 = max $\{x : (x, 1) \in S\}$ and b1 = min$\{x : (x, 0) \in S\}$ (if no example in S is positive we set b0 = $-\infty$ and if no example in S is negative we set b1 = ∞). Let bS be a threshold corresponding to an ERM hypothesis, hS, which implies that bS $\in (b0,b1)$. Therefore, a sufficient condition for $L_D(h_S) \leq \varepsilon$ is that both b0 ≥ a0 and b1 ≤ a1. In other words,

P[L $_{D^m}$ $_{\varepsilon}$ $_{D^m}$ D(hS)>ε]≤P[b0<a0∨b1>a1],S

and using the union bound we can bound the preceding by

P[L $_{D^m}$ $_{\varepsilon}$ $_{D^m}$ D(hS)>ε]≤P[b0<a0]+P[b1>a1].(6.1)S

The event b0 < a0 happens if and only if all examples in S are not in the interval (a_0,a), whose probability mass is defined to be ε, namely,

P[b]=P?me−εm0<a0[□

$\forall(x,y)\in S, x\in(a_0,a)]=(1-\varepsilon)\leq$.S $_{D^m}$ $_{\varepsilon}$ $_{D^m}$

Since we assume $m > \log(2/\delta)/\varepsilon$ it follows that the equation is at most δ/2. In the same way it is easy to see that PS $D^m[b1 > a1] \leq \delta/2$. Combining with Equation (6.1) we conclude our proof.

6.2 The VC-Dimension

We see, therefore, that while finiteness of H is a sufficient condition for learnability, it is not a necessary condition. As we will show, a property called the VC-dimension of a hypothesis class gives the correct characterization of its learnability. To motivate the definition of the VC-dimension, let us recall the No-Free-Lunch theorem (Theorem 5.1) and its proof. There, we have shown that without

restricting the hypothesis class, for any learning algorithm, an adversary can construct a distribution for which the learning algorithm will perform poorly, while there is another learning algorithm that will succeed on the same distribution. To do so, the adversary used a finite set $C \subset X$ and considered a family of distributions that are concentrated on elements of C. Each distribution was derived from a "true" target function from C to $\{0,1\}$. To make any algorithm fail, the adversary used the power of choosing a target function from the set of all possible functions from C to $\{0, 1\}$.

When considering PAC learnability of a hypothesis class H, the adversary is restricted to constructing distributions for which some hypothesis $h \in H$ achieves a zero risk. Since we are considering distributions that are concentrated on elements of C, we should study how H behaves on C, which leads to the following definition.

definition 6.2 (Restriction of H to C) Let H be a class of functions from X to $\{0, 1\}$ and let $C = \{c_1, \ldots, c_m\} \subset X$. The restriction of H to C is the set of functions from C to $\{0, 1\}$ that can be derived from H. That is,

$$H_C = \{(h(c_1), \ldots, h(c_m)) : h \in H\},$$

where we represent each function from C to $\{0, 1\}$ as a vector in $\{0, 1\}^{|C|}$.

If the restriction of H to C is the set of all functions from C to {0,1}, then we say that H shatters the set C. Formally:

definition 6.3 (Shattering) A hypothesis class H shatters a finite set $C \subset X$ if the restriction of H to C is the set of all functions from C to {0,1}. That is, $|H_C| = 2^{|C|}$.

Example6.2 Let H be the class of threshold functions over R. Take a set $C = \{c_1\}$. Now, if we take $a = c_1 + 1$, then we have $h_a(c_1) = 1$, and if we take $a = c_1$ be a training set size. Assume that there exists a set $C \subset X$ of size m that is shattered by H. Then, for any learning algorithm, A, there exist a distribution D over $X \times \{0,1\}$ and a predictor h such that $L_D(h) = 0$ but with probability of at least $1/7$ over the choice of S we have that $L_D(A(S)) \geq 1/8$.

Getting back to the construction of an adversarial distribution as in the proof of the No-Free-Lunch theorem (Theorem 5.1), we see that whenever some set C is shattered by H, the adversary is not restricted by H, as they can construct a distribution over C based on any target function from C to $\{0, 1\}$, while still maintaining the realizability assumption. This immediately yields:

$= 1$, then we have $h_a(c_1) = 0$. Therefore H_C is the set of all functions from C to $\{0, 1\}$, and H shatters C. Now take a set $C = \{c_1, c_2\}$, where $c_1 \leq c_2$. No $h \in H$ can account for the labeling $(1, 0)$, because any threshold that assigns the label 0 to c_1 must assign the label 0 to c_2 as well. Therefore not all functions from C to $\{0, 1\}$ are included in H_C, hence C is not shattered by H.

Corollary 6.4 tells us that if H shatters some set C of size 2m then we cannot learn H using m examples. Intuitively, if a set C is shattered by H, and we receive a sample containing half the instances of C, the labels of these instances give us no information about the labels of the rest of the instances in C – every possible labeling of the rest of the instances can be explained by some hypothesis in H. Philosophically,

If someone can explain every phenomenon, his explanations are worthless.

This leads us directly to the definition of the VC dimension.

definition 6.5 (VC-dimension) The VC-dimension of a hypothesis class H, denoted VCdim(H), is the maximal size of a set C ⊂ X that can be shattered by H. If H can shatter sets of arbitrarily large size we say that H has infinite VC-dimension.

A direct consequence of Corollary 6.4 is therefore:

theorem6.6 Let H be a class of infinite VC-dimension. Then, H is not PAC learnable

Since H has an infinite VC-dimension, for any training set size m, there exists a shattered set of size 2m, and the claim follows by Corollary 6.4.

We shall see later in this chapter that the converse is also true: A finite VC-dimension guarantees learnability. Hence, the VC-dimension characterizes PAC learnability. But before delving into more theory, we first show several examples.

6.3 Examples

In this section we calculate the VC-dimension of several hypothesis classes. To show that VCdim(H) = d we need to show that

1. There exists a set C of size d that is shattered by H.
2. Every set C of size d + 1 is not shattered by H.

6.3.1 Threshold Functions

Let $H_{\geq}$ be the class of threshold functions over R. Recall Example 6.2, where we have shown that for an arbitrary set C = {c1}, $H_{\geq}$ shatters C; therefore VCdim($H_{\geq}$) ≥ 1. We have also shown that for an arbitrary set C = {c1, c2} where c1 ≤ c2, $H_{\geq}$ does not shatter C. We therefore conclude that VCdim($H_{\geq}$) = 1.

6.3.2 Intervals

Let H be the class of intervals over R, namely, H = {ha,b : a,b ∈ R,a < b}, where ha,b : R → {0,1} is a function such that ha,b(x) = 1[x∈(a,b)]. Take the set C = {1,2}. Then, H shatters C (make sure you understand why) and therefore VCdim(H) ≥ 2. Now take an arbitrary set C = {c1,c2,c3} and assume without loss of generality that c1 ≤ c2 ≤ c3. Then, the labeling (1, 0, 1) cannot be obtained by an interval and therefore H does not shatter C. We therefore conclude that VCdim(H) = 2.

6.3.3 Axis Aligned Rectangles

Let H be the class of axis aligned rectangles, formally:

$$H=\{h(a:a\leq and 1,a2,bab1,b2)121\leq b_2\}$$

where

{1ifa1≤x1≤a2andb1≤x2≤b2h(a(,x)=1,a2,b1,bx2)12(6.2)0otherwise

We shall show in the following that VCdim(H) = 4. To prove this we need to find a set of 4 points that are shattered by H, and show that no set of 5 points can be shattered by H. Finding a set of 4 points that are shattered is easy (see Figure 6.1). Now, consider any set C ⊂ R2 of 5 points. In C, take a leftmost point (whose first coordinate is the smallest in C), a rightmost point (first coordinate is the largest), a lowest point (second coordinate is the smallest), and a highest point (second coordinate is the largest). Without loss of generality, denote C = {c1,...,c5} and let c5 be the point that was not selected. Now, define the labeling (1,1,1,1,0). It is impossible to obtain this labeling by an axis aligned rectangle. Indeed, such a rectangle must contain c1,...,c4; but in this case the rectangle contains c5 as well, because its coordinates are within the intervals defined by the selected points. So, C is not shattered by H, and therefore VCdim(H) = 4.

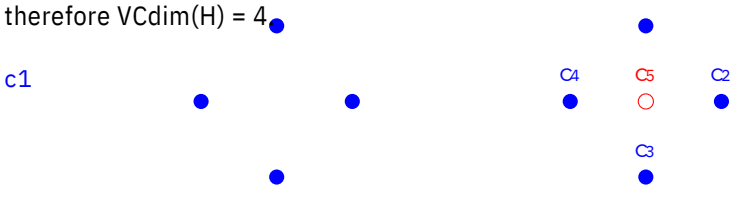


Figure 6.1 Left: 4 points that are shattered by axis aligned rectangles. Right: Any axis aligned rectangle cannot label c5 by 0 and the rest of the points by 1.

6.3.4 Finite Classes

Let H be a finite class. Then, clearly, for any set C we have |HC| ≤ |H| and thus C cannot be shattered if |H| < 2|C|. This implies that VCdim(H) ≤ log2(|H|). This shows that the PAC learnability of finite classes follows from the more general statement of PAC learnability of classes with finite VC-dimension, which we shall see in the next section. Note, however, that the VC-dimension of a finite class H can be significantly smaller than log2( ). For example, let X = {1,...,k}, for some integer k, and consider the class of threshold functions (as defined in Example 6.2). Then, |H| = k but VCdim( ) = 1. Since k can be arbitrarily large, the gap between log2( ) and VCdim( ) can be arbitrarily large.

6.3.5 VC-Dimension and the Number of Parameters

In the previous examples, the VC-dimension happened to equal the number of parameters defining the hypothesis class. While this is often the case, it is not always true. Consider, for example, the domain X = R, and the hypothesis class H={hθ:θ∈R}wherehθ: 0,1}isdefinedbyhθ(x)=d0.5sin(θx)e.It is possible to prove that VCdim( ) = ∞, namely, for every d, one can find d points that are shattered by (see Exercise 8).

6.4 The Fundamental Theorem of PAC learning

We have already shown that a class of infinite VC-dimension is not learnable. The converse statement is also true, leading to the fundamental theorem of statistical learning theory:

theorem 6.7 (The Fundamental Theorem of Statistical Learning) Let bea hypothesisclassoffunctionsfromadomain to andletthelossfunction bethe loss.Then,thefollowingareequivalent:

1. hastheuniformconvergenceproperty.
2. Any ERM rule is a successful agnostic PAC learner for .
3. isagnosticPAClearnable.
4. isPAClearnable.
5. Any ERM rule is a successful PAC learner for .
6. hasafiniteVC-dimension.

The proof of the theorem is given in the next section.

Not only does the VC-dimension characterize PAC learnability; it even determines the sample complexity.

theorem 6.8 (The Fundamental Theorem of Statistical Learning – Quantitative Version) Let beahypothesisclassoffunctionsfromadomain andletthelossfunctionbethe loss.Assumethat Then, there are absolute constants C ,C such that:

1. H has the uniform convergence property with sample complexity
$$C1\frac{d + \log(1/\delta)}{\le H()\le 22\varepsilon}\ m^{UC}\ \varepsilon,\delta\quad C\ \frac{d + \log(1/\delta)}{\varepsilon 2}$$
2. H is agnostic PAC learnable with sample complexity
$$\le \varepsilon\, 2\, H\qquad C1\frac{d + \log(1/\delta)}{}\qquad m\ \ (\varepsilon, \delta) \le\ C2\frac{d + \log(1/\delta)}{\varepsilon 2}$$
3. H is PAC learnable with sample complexity
$$d+\log(1/\delta)\ dog(1/\varepsilon)+\log(1/\delta)\qquad\qquad C1$$
$$\le\ mH(\varepsilon,\delta)\le C2\varepsilon\ \varepsilon$$

The proof of this theorem is given in Chapter 28.

Remark 6.3 We stated the fundamental theorem for binary classification tasks. A similar result holds for some other learning problems such as regression with the absolute loss or the squared loss. However, the theorem does not hold for all learning tasks. In particular, learnability is sometimes possible even though the uniform convergence property does not hold (we will see an example in Chapter 13, Exercise 2). Furthermore, in some situations, the ERM rule fails but learnability is possible with other learning rules.

6.5 Proof of Theorem 6.7

We have already seen that 1 → 2 in Chapter 4. The implications 2 → 3 and 3 → 4 are trivial and so is 2 → 5. The implications 4 → 6 and 5 → 6 follow from the No-Free-Lunch theorem. The difficult part is to show that 6 → 1. The proof is based on two main claims:

- If VCdim(H) = d, then even though H might be infinite, when restricting it to a finite set $C \subset X$, its "effective" size, $|H_C|$, is only $O(|C|)$. That is, the size of H_C grows polynomially rather than exponentially with $|C|$. This claim is often referred to as Sauer's lemma, but it has also been stated and proved independently by Shelah and by Perles. The formal statement is given in Section 6.5.1 later.
- In Section 4 we have shown that finite hypothesis classes enjoy the uniform convergence property. In Section 6.5.2 later we generalize this result and show that uniform convergence holds whenever the hypothesis class has a "small effective size." By "small effective size" we mean classes for which $|H_C|$ grows polynomially with $|C|$.

6.5.1 Sauer's Lemma and the Growth Function

We defined the notion of shattering, by considering the restriction of H to a finite set of instances. The growth function measures the maximal "effective" size of H on a set of m examples. Formally:

definition 6.9 (Growth Function) Let H be a hypothesis class. Then the growth function of H, denoted $\tau_H : N \to N$, is defined as

$$\tau_H(m) = \max_{C \subset X : |C| = m} |H_C|$$

In words, $\tau_H(m)$ is the number of different functions from a set C of size m to {0,1} that can be obtained by restricting H to C.

Obviously, if VCdim(H)=dthenforanym≤dwehaveτ mH(m)=2.In such cases, H induces all possible functions from C to {0, 1}. The following beautiful lemma, proposed independently by Sauer, Shelah, and Perles, shows that when m becomes larger than the VC-dimension, the growth function increases polynomially rather than exponentially with m.

l e m m a 6.10 (Sauer-Shelah-Perles)

Proof of Sauer's Lemma *

∑LetHbeanyhypothesisclasswithVCdim()dd<.Then,forallm,τmmH()≤i=0i.Inparticular,ifm>d+1thenτdH(

To prove the lemma it suffices to prove the following stronger claim: For any C= {c1, . . . , cm} we have

$$\forall H, |H_C| \leq |\{B \subseteq C : H \text{ shatters } B\}|. \quad (6.3)$$

The reason why Equation (6.3) is sufficient to prove the lemma is that if VCdim( ) d then no set whose size is larger than d is shattered by H and therefore

$$|\{B \subseteq C : H \text{ shatters } B\}| \leq \sum_{i=0}^{d} \binom{m}{i}$$

When m > d + 1 the right-hand side of the preceding is at most (em/d)d (see Lemma A.5 in Appendix A).

We are left with proving Equation (6.3) and we do it using an inductive argument. For m = 1, no matter what H is, either both sides of Equation (6.3) equal 1 or both sides equal 2 (the empty set is always considered to be shattered by H). Assume Equation (6.3) holds for sets of size k < m and let us prove it for sets of size m. Fix HandC={c,...,c} C′1 m.Denote ={c2,...,cm}andin addition, define the following two sets:

$$Y_0 = \{(y_2,\ldots,y_m) : (0,y_2,\ldots,y_m) \in H \vee (1,y_2,\ldots,y_m) \in H\}$$

and

$$Y_1 = \{(y_2,\ldots,y_m) : (0,y_2,\ldots,y_m) \in H \wedge (1,y_2,\ldots,y_m) \in H\}.$$

It is easy to verify that $|H_C| = |Y_0| + |Y_1|$. Additionally, since Y0 = HC′, using the induction assumption (applied on and C′) we have that

$$|Y_1| = |\{B \subseteq C' : H \text{ shatters } B\}| = |\{B \subseteq C : c_1 \in B \wedge H \text{ shatters } B\}|.$$

Next, define

$$H' = \{h \in H : \exists h' \in H \text{ s.t. } (1 - h'(c'_1), h(c_2), \ldots, h(c_m)) = (h(c_1), h(c_2), \ldots, h(c_m))\},$$

namely, H' contains pairs of hypotheses that agree on C' and differ on c_1. Using this definition, it is clear that if H' shatters a set $B \subseteq C'$ then it also shatters the set $B \cup \{c_1\}$ and vice versa. Combining this with the fact that $Y_1 = H'_{C'}$ and using the inductive assumption (now applied on H' and C') we obtain that

$$|Y_1| = |H'_{C'}| \leq |\{B \subseteq C' : H' \text{ shatters } B\}| = |\{B \subseteq C' : H' \text{ shatters } B \cup \{c_1\}\}| = |\{B \subseteq C : c_1 \in B \wedge H \text{ shatters } B\}| \leq |\{B \subseteq C : c_1 \in B \wedge H \text{ shatters } B\}|.$$

Overall, we have shown that

$$|H_C| = |Y_0| + |Y_1| \leq |\{B \subseteq C : c_1 \in 6B \wedge H \text{ shatters } B\}| + |\{B \subseteq C : c_1 \in B \wedge H \text{ shatters } B\}| = |\{B \subseteq C : H \text{ shatters } B\}|,$$

which concludes our proof.

6.5.2 Uniform Convergence for Classes of Small Effective Size

In this section we prove that if H has small effective size then it enjoys the uniform convergence property. Formally,

theorem6.11 Let H be a class and let τ be its growth function. Then, for every D and every $\delta \in (0,1)$, with probability of at least $1 - \delta$ over the choice of $S = (x_1, \ldots, x_m)$ we have

$$|L_D(h) - L_S(h)| \leq \frac{4 + \log(\tau_H(2))}{\delta \sqrt{2m}}$$

Before proving the theorem, let us first conclude the proof of Theorem 6.7.

Proof of Theorem 6.7 It suffices to prove that if the VC-dimension is finite then the uniform convergence property holds. We will prove that

$$\leq \frac{16d \log(2e/d) + \log(4/\delta)}{2\epsilon^2}$$

From Sauer's lemma we have that for $m > d$, $\tau_H(2m) \leq (2em/d)^d$. Combining this with Theorem 6.11 we obtain that with probability of at least $1 - \delta$,

$$|L_S(h) - L_D(h)| \leq \frac{4 + d \log(2em/d)}{\delta \sqrt{2m}}$$

For simplicity assume that $\log(2) \geq 4$; hence,

$$|L_S(h) - L_D(h)| \leq \delta \sqrt{\frac{2d \log(2em/d)}{m}}.$$

To ensure that the preceding is at most ε we need that

≥2dlog(m)2dlog(2e/d)m+.(δε)2(δε)2

Standard algebraic manipulations (see Lemma A.2 in Appendix A) show that a sufficient condition for the precedin

(gtoho)ldisthat≥ 2d 2d 4dlog(2e/d)m 4 log + .(δε)2 (δε)2 (δε)2

□

Remark 6.4 The upper bound on mUC H we derived in the proof Theorem 6.7 is not the tightest possible. A tighter analysis that yields the bounds given in Theorem 6.8 can be found in Chapter 28.

Proof of Theorem 6.11 *

We will start by sho

[wingthat4+log(τ(2m))Esup|HLD(h)−LS(h)|≤√.(6.4)S Dmh∈H2m

Since the random variable suph ∈H |LD(h) − LS(h)| is nonnegative, the proof of the theorem follows directly from the preceding using Markov's inequality (see Section B.1).

To bound the left-hand side of Equation (6.4) we first note that for every h ∈H,wecanrewriteL(h)=E′[L(h)],whereS′=′.,z′DS DmS′z1,..misan additional i

[.i.d.sample.Therefore], E supLD(h)LS(h)=EsupELS′(h)LS(h)

]|−|−.S Dmh∈HS Dmh∈HS′ DmAgeneralizatio

nofthetriangleinequalityyields

E[S′()− LhLS(h)] ≤E|L′(h)−L(h)|,S′ SS Dm S′ DmandthefactthatsupermumofexpectationissmumyieldssupE|LS′(h)−LS(h)|≤ E sup|L ′(h)−L (h)|.

Formally, the previous two inequalities follow from Jensen's inequality. Combining all we

[obtain]]Esup|LD(h)−LS(h)|≤Esup|LS′(h)−LS(h)|S Dmh∈H S,S′ Dmh∈H

$$\frac{1}{m}\,E_{S,S'}\sup_{h\in H}\left|\sum_{i=1}^{m}\left(\ell(h,z'_i)-\ell(h,z_i)\right)\right| \tag{6.5}$$

The expectation on the right-hand side is over a choice of two i.i.d. samples S=z,...,zandS′ ′ z′1 m =z1,...,m.Sinceallofthese2mvectorsarechosen i.i.d., nothing will change if we replace the name of the random vector zi with the nameoftherandomvectorz′ ′ i. If we do it, instead of the term (` (h, zi) – ` (h, zi)) in Equation (6.5) we will have the term −(` (h,z′i)− ` (h,zi)).Itfollowsthatfor every σ ∈ {±1}m we have that Equation (6.5) equals

$$\mathbb{E}\left[\sup_{h\in H}\left(\frac{1}{m}\sum_{i=1}^{m}\sigma_i(\ell(h,z'_i)-\ell(h,z_i))\right)\right]$$

Since this holds for every σ ∈ {±1}m, it also holds if we sample each component of σ uniformly at random from the uniform distribution over {±1}, denoted U±. Hence, Equation (6.5) also equals

$$\mathbb{E}_{\sigma\sim U_\pm}\left[\sup_{h\in H}\left(\frac{1}{m}\sum_{i=1}^{m}\sigma_i(\ell(h,z'_i)-\ell(h,z_i))\right)\right]$$

and by the linearity of expectation it also equals

$$\mathbb{E}_{\sigma\sim U_\pm}\left[\sup_{h\in H}\left(\frac{1}{m}\sum_{i=1}^{m}\sigma_i\ell(h,z'_i)+\frac{1}{m}\sum_{i=1}^{m}(-\sigma_i)\ell(h,z_i)\right)\right]$$

Next, fix S and S′, and let C be the instances appearing in S and S′. Then, we can take the supremum only over h ∈ HC. Therefore, (1/m) Eσ∼U± [sup_{h∈HC} (Σ_{i=1}^{m} σi(ℓ(h,zi)−ℓ(h,zi)) + Σ_{i=1}^{m} (−σi)ℓ(h,zi))]

$$= \frac{1}{m}\,\mathbb{E}_{\sigma\sim U_\pm}\left[\max_{h\in H_C}\left(\sum_{i=1}^{m}\sigma_i\ell(h,z'_i)+\sum_{i=1}^{m}(-\sigma_i)\ell(h,z_i)\right)\right].$$

Fix some h ∈ HC and denote θ = (1/m) Σ_{i=1}^{m} σi(ℓ(h,zi)−ℓ(h,zi)). Since E[θh]=0 and θh is an average of independent variables, each of which takes values in [−1, 1], we have by Hoeffding's inequality that for every ρ > 0,

$$P[|\theta|>\rho]\le 2\exp(-2m\rho^2).$$

Applying the union bound over h ∈ HC, we obtain that for any ρ > 0,

$$P\left[\max_{h\in H_C}|\theta_h|>\rho\right]\le 2|H_C|\exp(-2m\rho^2).$$

Finally, Lemma A.4 in Appendix A tells us that the preceding implies $4+\mathbb{E}\max_{h\in H_C}|\theta_h|\le\sqrt{\log(|H_C|)}$. Combining all with the definition of τH, we have shown that

$$\left(4+\sqrt{\log(\tau_H(2m))}\right)\sup_{h\in H}|L_D(h)-L_S(h)|\le\sqrt{S}$$

6.6 Summary

The fundamental theorem of learning theory characterizes PAC learnability of classes of binary classifiers using VC-dimension. The VC-dimension of a class is a combinatorial property that denotes the maximal sample size that can be shattered by the class. The fundamental theorem states that a class is PAC learnable if and only if its VC-dimension is finite and specifies the sample complexity required for PAC learning. The theorem also shows that if a problem is at all learnable, then uniform convergence holds and therefore the problem is learnable using the ERM rule.

6.7 Bibliographic remarks

The definition of VC-dimension and its relation to learnability and to uniform convergence is due to the seminal work of Vapnik & Chervonenkis (1971). The relation to the definition of PAC learnability is due to Blumer, Ehrenfeucht, Haussler & Warmuth (1989).

Several generalizations of the VC-dimension have been proposed. For exam- ple, the fat-shattering dimension characterizes learnability of some regression problems (Kearns, Schapire & Sellie 1994, Alon, Ben-David, Cesa-Bianchi & Haussler 1997, Bartlett, Long & Williamson 1994, Anthony & Bartlet 1999), and the Natarajan dimension characterizes learnability of some multiclass learning problems (Natarajan 1989). However, in general, there is no equivalence between learnability and uniform convergence. See (Shalev-Shwartz, Shamir, Srebro & Sridharan 2010, Daniely, Sabato, Ben-David & Shalev-Shwartz 2011).

Sauer's lemma has been proved by Sauer in response to a problem of Erdos (Sauer 1972). Shelah (with Perles) proved it as a useful lemma for Shelah's theory of stable models (Shelah 1972). Gil Kalai tells[1] us that at some later time, Benjy Weiss asked Perles about such a result in the context of ergodic theory, and Perles, who forgot that he had proved it once, proved it again. Vapnik and Chervonenkis proved the lemma in the context of statistical learning theory.

6.8 Exercises

1. Show the following monotonicity property of VC-dimension: For every two hypothesis classes if $H' \subseteq H$ then $VCdim(H') \leq VCdim(H)$.
2. Given some finite domain set, X, and a number $k \leq |X|$, figure out the VC-dimension of each of the following classes (and prove your claims):
   1. $H_X^k = \{h \in \{0,1\}^X : |\{x : h(x) = 1\}| = k\}$. That is, these to fall functions that assign the value 1 to exactly k elements of X.

[1]

2. $H_{at\text{-}most\text{-}k} = \{h_S : S \subseteq \{1, \ldots, n\}, |S| \leq k\}$, where $h_S(x) = 1$ if and only if $x_i = 1$ for all $i \in S$.
3. Let $X = \{0,1\}^n$. For a set $I \subseteq \{1, 2, \ldots, n\}$ we define a parity function h_I as follows. On a binary vector $x = (x_1, x_2, \ldots, x_n)$, $h_I(x) = \left(\sum_{i \in I} x_i\right) \bmod 2$. (That is, h_I computes parity of bits in I.) What is the VC-dimension of the class of all such parity functions, $H_{n\text{-}parity} = \{h_I : I \subseteq \{1, 2, \ldots, n\}\}$?
4. We proved Sauer's lemma by proving that for every class H of finite VC-dimension d, and every subset A of the domain,

$$|H_A| \leq |\{B \subseteq A : H \text{ shatters } B\}| \leq \sum_{i=0}^{d} \binom{|A|}{i}.$$

Show that there are cases in which the previous two inequalities are strict (namely, the $\leq$ can be replaced by $<$) and cases in which they can be replaced by equalities. Demonstrate all four combinations of $=$ and $<$.

5. VC-dimension of axis aligned rectangles in R^d: Let H^d_{rec} be the class of axis aligned rectangles in R^d. We have already seen that $VCdim(H^2_{rec}) = 4$. Prove that in general, $VCdim(H^d_{rec}) = 2d$.
6. VC-dimension of Boolean conjunctions: Let H^d_{con} be the class of Boolean conjunctions over the variables $x_1, \ldots, x_d$ ($d \geq 2$). We already know that this class is finite and thus (agnostic) PAC learnable. In this question we calculate $VCdim(H^d_{con})$.
   1. Show that $|H^d_{con}| \leq 3^d + 1$.
   2. Conclude that $VCdim(H^d_{con}) \leq d \log 3$.
   3. Show that H^d_{con} shatters the set of unit vectors $\{e_i : i \leq d\}$.
   4. (**) Show that $VCdim(H^d_{con}) \leq d$.

   Hint: Assume by contradiction that there exists a set $C = \{c_1, \ldots, c_{d+1}\}$ that is shattered by H^d_{con}. Let $h_1, \ldots, h_{d+1}$ be hypotheses in H^d_{con} that satisfy

$$h_i(c_j) = \begin{cases} 0 & i = j \\ 1 & \text{otherwise} \end{cases} \quad \forall i, j \in [d+1],$$

   For each $i \in [d + 1]$, h_i (or more accurately, the conjunction that corresponds to h_i) contains some literal ℓ_i which is false on c_i and true on c_j for each $j \neq i$. Use the Pigeonhole principle to show that there must be a pair $i < j \leq d + 1$ such that ℓ_i and ℓ_j use the same x_k and use that fact to derive a contradiction to the requirements from the conjunctions h_i, h_j.
   5. Consider the class H^d_{mon} of monotone Boolean conjunctions over $\{0,1\}^d$.

Asin Hdcon, the empty conjunction is interpreted as the all-positive hypothesis. We augment Hdmcwithth h−oneall-negativehypothesis.Show that VCdim(Hdmcon)=d.

7. We have shown that for a finite hypothesis class H, VCdim(H) ≤ blog(|H|)c. However, this is just an upper bound. The VC-dimension of a class can be much lower than that:

1. Find an example of a class H of functions over the real interval X = [0,1] such that H is infinite while VCdim(H) = 1.
2. Give an example of a finite hypothesis class H over the domain X = [0, 1], where VCdim(H) = blog2(|H|)c.

8. (*) It is often the case that the VC-dimension of a hypothesis class equals (or can be bounded above by) the number of parameters one needs to set in order to define each hypothesis in the class. For instance, if H is the class of axis aligned rectangles in Rd, then VCdim(H) = 2d, which is equal to the number of parameters used to define a rectangle in Rd. Here is an example that shows that this is not always the case. We will see that a hypothesis class might be very complex and even not learnable, although it has a small number of parameters.

Consider the domain X = R, and the hypothesis class

$$H=\{x\to 7\,\mathrm{dsin}(\theta x)\mathrm{e}:\theta\in R\}$$

(here, we take d−1e = 0). Prove that VCdim(H) = ∞.

Hint: There is more than one way to prove the required result. One option is by applying the following lemma: If 0.x1x2x3 . . ., is the binary expansion of x ∈ (0, 1), then for any natural number m, dsin(2mπx)e = (1 − xm), provided that ∃k ≥ m s.t. xk = 1.

9. Let H be the class of signed intervals, that is, ——

$$H = \{h_{a,b,s} : a \le b,\ s \in \{-1, 1\}\}\ \text{wh}$$

{eresifx∈[a,b]ha,b,s(x)=H
−s ifx∈/[a,b] →
Calculate VCdim(H H
H). H

10. Let H be a class of functions from X to {0,1}.

1. Prove that if VCdim(H) ≥ d, for any d, then for some probability distribution D over X × {0, 1}, for every sample size, m,

≥d−mE[LD(A(S))]minLD(h)+S
Dm h∈H 2d

1. Prove that
VCdim (
∪ri=1Hi)≤4dlog(2d)+2log(r).
Hint: Take a set of k examples and assume that they are shattered by the union class. Therefore, the union class can produce all 2k possible labelings on these examples. Use Sauer's lemma to show that the union class cannot produce more than rkd labelings. Therefore, 2k < rkd. Now use Lemma A.2.
2. (*) Prove that for r = 2 it holds that
VCdim (
H1 ∪ H2)≤2d+1.
12. Dudley classes: In this question we discuss an algebraic framework for defining concept classes over Rn and show a connection between the VC dimension of such classes and their algebraic properties. Given a function f:Rn → R we define the corresponding function, POS(f)(x) = 1[f(x)>0]. For a class F of real valued functions we define a corresponding class of functions POS(F) = {POS(f) : f ∈ F}. We say that a family, F, of real valued functions is linearly closed if for all f,g ∈ F and r ∈ R, (f + rg) ∈ F (where addition and scalar multiplication of functions are defined point wise, namely, for all x ∈ Rn, (f + rg)(x) = f(x) + rg(x)). Note that if a family of functions is linearly closed then we can view it as a vector space over the reals. For a nction :Rn →Ran FFdeffug dafamilyoffunctions,let+g={f+g:f∈F}. Hypothesis classes that have a representation as POS(F + g) for some vector space of functions
F and some function g are called Dudley classes.
1. Show that for every g : Rn → R and every vector space of functions F as defined earlier, VCdim(POS(
F + g)) = VCdim(POS(F)).
2. (**) For every linearly closed family of real valued functions F, the VC-dimension of the corresponding class POS(F) equals the linear dimension of F (as a vector space). Hint: Let f1, . . . , fd be a basis for the vector space F.Considerthemappingx→7(f(x),...,f(x))(froRnRd1dmto).Note that this mapping induces a matching between functions over Rn of the form POS(f) and homogeneous linear halfspaces in Rd (the VC-dimension of the class of homogeneous linear halfspaces is analyzed in Chapter 9).
3. Show that each of the following classes can be represented as a Dudley class:
1.TheclassHSofhalfspacesoverRnn(seeChapter9).
2.TheclassHHSofallhomogeneoushalfspacesoveRnnr(seeChapter9).
3.TheclassBofallfunctionsdefinedby(open)bRddallsin.Usethe Dudley representation to figure out the VC-dimension of this class.
4.LetPdndenotetheclassoffunctionsdefinedbypolynomialinequalities of degree ≤ d, namely,
Pdn=

where, for x = (x1. . . . , xn), hp(x) = 1[p(x) ≥0] (the degree of a multi-variable polynomial is the maximal sum of variable exponents over all ofitsterms.Forexample,thedegreeofp(x)=3x32+4 21x2 x3x7is5).

1. Use the Dudley representation to figure out the VC-dimension of the classPd1–theclassofalld-degreepolynomialsoverR.
2. Prove that the class of all polynomial classifiers over R has infinite VC-dimension.
3. Use the Dudley representation to figure out the VC-dimension of the class Pdn (as a function of d and n).

7 Nonuniform Learnability

The notions of PAC learnability discussed so far in the book allow the sample sizes to depend on the accuracy and confidence parameters, but they are uniform with respect to the labeling rule and the underlying data distribution. Consequently, classes that are learnable in that respect are limited (they must have a finite VC-dimension, as stated by Theorem 6.7). In this chapter we consider more relaxed, weaker notions of learnability. We discuss the usefulness of such notions and provide characterization of the concept classes that are learnable using these definitions.

We begin this discussion by defining a notion of "nonuniform learnability" that allows the sample size to depend on the hypothesis to which the learner is compared. We then provide a characterization of nonuniform learnability and show that nonuniform learnability is a strict relaxation of agnostic PAC learnability. We also show that a sufficient condition for nonuniform learnability is that $\mathcal{H}$ is a countable union of hypothesis classes, each of which enjoys the uniform convergence property. These results will be proved in Section 7.2 by introducing a new learning paradigm, which is called Structural Risk Minimization (SRM). In Section 7.3 we specify the SRM paradigm for countable hypothesis classes, which yields the Minimum Description Length (MDL) paradigm. The MDL paradigm gives a formal justification to a philosophical principle of induction called Occam's razor. Next, in Section 7.4 we introduce consistency as an even weaker notion of learnability. Finally, we discuss the significance and usefulness of the different notions of learnability.

7.1 Nonuniform Learnability

"Nonuniform learnability" allows the sample size to be nonuniform with respect to the different hypotheses with which the learner is competing. We say that a hypothesis h is (ϵ, δ)-competitive with another hypothesis h' if, with probability higher than $(1-\delta)$,

$$L_{(D,f)}(h) \leq L_{(D,f)}(h') + \epsilon.$$

In PAC learnability, this notion of "competitiveness" is not very useful, as we are looking for a hypothesis with an absolute low risk (in the realizable case) or

with a low risk compared to the minimal risk achieved by hypotheses in our class (in the agnostic case). Therefore, the sample size depends only on the accuracy and confidence parameters. In nonuniform learnability, however, we allow the sample size to be of the form m H(ε, δ, h); namely, it depends also on the h with which we are competing. Formally,

definition 7.1 A hypothesis class H is nonuniformly learnable if there exist a learning algorithm, A, and a function mNUL : (0 1)2 H , ×H→Nsuchthat,forevery ε,δ ∈ (0, 1) and for every h ∈ H, if m ≥ mNULH (ε, δ, h) then for every distribution D, with probability of at least 1 − δ over the choice of S Dm, it holds that

L D(A(S)) ≤ LD(h)+ε

A hypothesis class is agnostically PAC learnable if there exist a learning algorithm, andafunction suchthat forevery and foreverydistribution if, thenwithprobabilityofatleast overthechoiceof itholdsthat

A m:(0,1)2 H →N ε,δ∈(0,1) m

At this point it might be useful to recall the definition of agnostic PAC learnability (Definition 3.3).

D D

$$L_D(A(S)) \le \min_{h' \in H} L_D(h') + \varepsilon.$$

Note that this implies that for every h ∈ H

S m ≥mH(ε,δ) 1−δ

$$L_{D(A(S))} \le L_D(h) + \varepsilon.$$

In both types of learnability, we require that the output hypothesis will be (ε,δ)-competitive with every other hypothesis in the class. But the difference between these two notions of learnability is the question of whether the sample size m may depend on the hypothesis h to which the error of A(S) is compared.

Note that that nonuniform learnability is a relaxation of agnostic PAC learnability. That is, if a class is agnostic PAC learnable then it is also nonuniformly learnable.

7.1.1 Characterizing Nonuniform Learnability

Our goal now is to characterize nonuniform learnability. In the previous chapter we have found a crisp characterization of PAC learnable classes, by showing that a class of binary classifiers is agnostic PAC learnable if and only if its VC-dimension is finite. In the following theorem we find a different characterization for nonuniform learnable classes for the task of binary classification.

theorem 7.2 Ahypothesisclass ofbinaryclassifiersisnonuniformlylearnable if and only if it is a countable union of agnostic PAC learnable hypothesis classes.

The proof of Theorem 7.2 relies on the following result of independent interest:

theorem7.3 Let
H be a hypot
∈N H
Uhesisclassthatcanbewrittenasacountableunionofhypothesisclasses,=nHn,whereeachHnenjoystheuniformcon
H

Recall that in Chapter 4 we have shown that uniform convergence is sufficient for agnostic PAC learnability. Theorem 7.3 generalizes this result to nonuniform learnability. The proof of this theorem will be given in the next section by introducing a new learning paradigm. We now tu

UrntoprovingTheorem7.2.ProofofTheorem7.2FirstassumethatH=n∈NHnwhereeachHnisag-nosticPAClearnable.
follows that each Hn has the uniform convergence property. Therefore, using Theorem 7.3 we obtain that
H is nonuniform learnable.
For the other direction, assume that
H is nonuniform learnable using some algorithm A. For every n
∈ N, let Hn = {h ∈ H : mNULH (1/8,1/7,h) ≤ n}.
Clearly,
H=∪nHn.Inaddition,u mNUL∈N singthedefinitionofHweknowthat for any distribution
D that satisfies the realizability assumption with respect to □
H,withprobabilityofatleast6/7oS DnnverwehavethatLD(A(S))≤1/8.
Using the fundamental theorem of statistical learning, this implies that the VC-dimension of
Hn must be finite, and therefore Hn is agnostic PAC learnable.

The following example shows that nonuniform learnability is a strict relaxation of agnostic PAC learnability; namely, there are hypothesis classes that are nonuniform learnable but are not agnostic PAC learnable.

Example7.1 Considerabinaryclassificationproblemwiththeinstancedomain
b eing
X = R. For every n ∈ N let Hn be the class of polynomial classifiers of degree n; namely,
Hn is the set of all classifiers of th
Ueformh(x)=sign(p(x))wherep:R→Risapolynomialofdegreen.LetH=n∈NHn.Therefore,Histheclassofallpolynomial
a while VCdim(Hn) = n + 1 (see Exercise 12). Hence, H is not PAC learnable, while on the basis of Theorem 7.3,
H is nonuniformly learnable.

7.2 Structural Risk Minimization

So far, we have encoded our prior knowledge by specifying a hypothesis class H, which we believe includes a good predictor for the learning task at hand. Yet another way to express our prior knowledge is by specifying preferences over hypotheses within
H. In the Structural Risk Minimization
U(SRM)paradigm,wedosobyfirstassumingthatHcanbewrittenasH=n∈NHnandthen
specifyingaweightfunction,w:N→[0,1],whichassignsaweighttoeach hypothesis class, Hn, such that a higher weight reflects a stronger preference for the hypothesis class. In this section we discuss how to learn with such prior knowledge. In the next section we describe a couple of important weighting schemes, including Minimum Description Length.

∪Concretely,letHbeahypothesisclassthatcanbewrittenasH=n∈NHn.Forexample,Hmaybetheclassofa
the class of polynomial classifiers of degree n (see Example 7.1). Assume that for each n, the class
Hn enjoys the uniform convergence property (see Definition 4.3 in Chapter 4) with a sample complexity function mUC
H(ε,δ).Letusalsodefinen
the function εn : N
× (0, 1) → (0, 1) by

εn(m, δ) = min

{ε∈(0,1):mUCH(ε,δ)≤m}. (7.1)n

In words, we have a fixed sample size m, and we are interested in the lowest possible upper bound on the gap between empirical and true risks achievable by using a sample of m examples.

From the definitions of uniform convergence and εn, it follows that for every m and δ, with probability of at least 1
− δ over the choice of S Dm we have that

∀h ∈ Hn, |LD(h) − LS(h

∑)|≤εn(m,δ).(7.2)Letw:N→∞[0,1]beafunctionsuchthatn=1w(n)≤1.Werefertowasaweightfunctionovertl
can reflect the importance that the learner attributes to each hypothesis class, or some measure of the complexity of different hypothesis classes. If
H is a finite union of N hypothesis classes, one can simply assign the same weight of 1/N to all hypothesis classes. This equal weighting corresponds to no a priori preference to any hypothesis class. Of course, if one believes (as prior knowledge) that a certain hypothesis class is more likely to contain the correct target function, then it should be assigned a larger weight, reflecting this prior knowledge. When H is a (countable) infinite union of hypothesis classes, a uniform weighting is not possible but many other weighting schemes may work. For example, one can choosew(n)= 6 or ( )=2−n
π2n2 wn .Laterinthischapterwewillprovideanother convenient way to define weighting functions using description languages.

The SRM rule follows a "bound minimization" approach. This means that the goal of the paradigm is to find a hypothesis that minimizes a certain upper bound on the true risk. The bound that the SRM rule wishes to minimize is given in the following theorem.

theorem7.4 Let :
→ [0 1] be a function suc ∈
− δ D
∪∑∞wN,hthatn=1w(n)≤1.LetHbeahypothesisclassthatcanbewrittenasH=n∈NHn,whereforeachn,Hn
n ∈ ∈ H
m UC
H.Letεbeasdefinedin Equation (7.1). Then,forevery δ(0,1)andnn
distribution |LD(h) − LS (h) | ≤εn(m, w(n) · δ).
D,withprobabilityofatleast1 overthechoiceofS m,the
Therefore for every δ= (0, 1) and distribution D, with probability of at least
following bound holds (simultaneously) for every n and h:

1 − δ it holds that

$$\forall h \in H, L_D(h) \le L_S(h) + \min_{n:h \in H_n} \varepsilon_n(m, w(n)\cdot\delta). \quad (7.3)$$

Proof For each n define $\delta_n = w(n)\delta$. Applying the assumption that uniform convergence holds for all n with the rate given in Equation (7.2), we obtain that if we fix n in advance, then with probability of at least $1 - \delta_n$ over the choice of $S \sim D^m$,

$$\forall h \in H_n, |L_D(h) - L_S(h)| \le \varepsilon_n(m, \delta_n).$$

Applying th □

∑eunionbound∑overn=1,2,...,weobtainthatwithprobabilityofatleast1−nδn=1−δnw(n)≥1−δ,thepprecedingholdsfora

Denote

$$n(h) = \min \{n : h \in H_n\}, \quad (7.4)$$

and then Equation (7.3) implies that

$$L_D(h) \le L_S(h) + \varepsilon_{n(h)}(m, w(n(h)) \cdot \delta).$$

Structural Risk Minimization (SRM)

The SRM paradigm searches for h that minimizes this bound, as formalized in the following pseudocode:

prior k

Unowledge: $H = \bigcup_n H_n$ where H_n has uniform convergence with $m^{UC}_{H_n}$ and $w : \mathbb{N} \to [0,1]$ where $\sum_n w(n) \le 1$

define: ε_n as in Equation (7.1) ; $n(h)$ as in Equation (7.4)

input: training set $S \sim D^m$, confidence δ

output: $h \in \operatorname{argmin}_{h \in H} [L_S(h) + \varepsilon_{n(h)}(m, w(n(h)) \cdot \delta)]$

Unlike the ERM paradigm discussed in previous chapters, we no longer just care about the empirical risk, $L_S(h)$, but we are willing to trade some of our bias toward low empirical risk with a bias toward classes for which $\varepsilon_{n(h)}(m, w(n(h)) \cdot \delta)$ is smaller, for the sake of a smaller estimation error.

Next we show that the SRM paradigm can be used for nonuniform learning of every class, which is a countable union of uniformly converging hypothesis classes.

theorem 7.5 Let $H = \bigcup_{n \in \mathbb{N}} H_n$ be a hypothesis class such that each H_n has the uniform convergence property with sample complexity $m^{UC}_{H_n}$. Let $w : \mathbb{N} \to [0,1]$ be such that $\sum_n w(n) \le 1$. Then, H is nonuniformly learnable using the SRM rule with rate

$$m^{NUL}_H(\varepsilon, \delta, h) \le m^{UC}_{H_{n(h)}}(\varepsilon/2, \tfrac{6\delta}{(\pi\, n(h))^2}).$$

P

∑roofLetAbetheSRMalgorithmwithrespecttotheweightingfunctionw.Foreveryh∈H,

ε,andδ,letm≥mUCH(ε,w(n(h))δ).Usingthefactthatn(h)nw(n)=1,wecanapplyTheore

m7.4togetthat,withprobabilityofatleast1−δoverthechoiceofS Dm,wehavethatfor

everyh′∈H,

L(′)
≤(′ ′DhLSh)+εn(h′)(m,w(n(h))δ).
The preceding holds in particular for the hypothesis A(S) returned by the SRM rule. By the defin

[itionofSRMweobtainthatL(())min(′)+(′]DAS≤LShεnh′m,w≤h′()(n(h))δ)LS(h)+εn(h)

(m,w(n(h))δ).

Finally, if m ≥mUCH(ε/2,w(n(h))δ)thenclearlyε≤n(h)n(h)(m,w(n(h))δ)ε/2.In addition, from the uniform convergence property of each Hn we have that with probability of more than 1 − δ ,

LS (h)
≤ LD(h) + ε/2.
Combining all the preceding we obtain that L D(A(S)) ≤ LD(h) + ε, which concludes our proof.

Note that the previous theorem also proves Theorem 7.3.

Remark 7.2 (No-Free-Lunch for Nonuniform Learnability) We have shown that any countable union of classes of finite VC-dimension is nonuniformly learnable. It turns out that, for any infinite domain set, X, the class of all binary valued functions over X is not a countable union of classes of finite VC-dimension. We leave the proof of this claim as a (nontrivial) exercise (see Exercise 5). It follows that, in some sense, the no free lunch theorem holds for nonuniform learning as well: namely, whenever the domain is not finite, there exists no nonuniform learner with respect to the class of all deterministic binary classifiers (although for each such classifier there exists a trivial algorithm that learns it – ERM with respect to the hypothesis class that contains only this classifier).

It is interesting to compare the nonuniform learnability result given in Theorem 7.5 to the task of agnostic PAC learning any specific Hn separately. The prior knowledge, or bias, of a nonuniform learner for H is weaker – it is searching for a model throughout the entire class H, rather than being focused on one specific Hn. The cost of this weakening of prior knowledge is the increase in sample complexity needed to compete with any specific h ∈ Hn. For a concrete evaluation of this gap, consider the task of binary classification with the zero-one loss. Assume that for all n, VCdim(H UC n+log(1/δ)n)=n.SincemH(ε,δ)=Cn ε2(where C is the contant appearing in Theorem 6.8), a straightforward calculation shows

the index of the first class in which h resides. That cost increases with the index of the class, which can be interpreted as reflecting the value of knowing a good priority order on the hypotheses in H.

7.3 Minimum Description Length and Occam's Razor

Let H be a countable hypothesis class. T

Uhen,wecanwriteHasacountableunionofsingletonclasses,namely,H=n∈N{hn}.By

Hoeffding'sinequality(Lemma4.5),eachsingletonclasshastheuniformconvergence

propertywith

rate mUC(ε,δ) = log

√(2/δ)2ε2.Therefore,thefunctionεngiveninEquation(7.1)becomesε(m,δ)=log

[(2/δ)n2mandvtheSRMrulebecomes]−log(w(n))+log(2/δ)argminLS(h)+.hmn∈H2

Equivalently, we can think of w as a function from H to [0, 1], and then the SRM rule becomes

[v]−log(w(h))+log(2/δ)argminLS(h)+.h∈H2m

It follows that in this case, the prior knowledge is solely determined by the weight we assign to each hypothesis. We assign higher weights to hypotheses that we believe are more likely to be the correct one, and in the learning algorithm we prefer hypotheses that have higher weights.

In this section we discuss a particular convenient way to define a weight function over H, which is derived from the length of descriptions given to hypotheses. Having a hypothesis class, one can wonder about how we describe, or represent, each hypothesis in the class. We naturally fix some description language. This can be English, or a programming language, or some set of mathematical formulas. In any of these languages, a description consists of finite strings of symbols (or characters) drawn from some fixed alphabet. We shall now formalize these notions.

Let H be the hypothesis class we wish to describe. Fix some finite set Σ of symbols (or "characters"), which we call the alphabet. For concreteness, we let Σ = {0,1}. A string is a finite sequence of symbols from Σ; for example, σ = (0, 1, 1, 1, 0) is a string of length 5. We denote by |σ| the length of a string. The set of all finite length strings is denoted Σ^*. A description language for H is a function $d : H \to \Sigma^*$, mapping each member h of H to a string d(h). d(h) is called "the description of h," and its length is denoted by | h | .

We shall require that description languages be prefix-free; namely, for every distinct h,h′, d(h) is not a prefix of d(h′). That is, we do not allow that any string d(h) is exactly the first |h| symbols of any longer string d(h′). Prefix-free

lemma 7.6 (Kraft Inequality) If

∑S⊆{0,1}∗ isaprefix-freesetofstrings,then1|≤1.2σ|σ∈S

Proof Defineaprobabilitydistributionoverthemembersof S as follows: Repeatedly toss an unbiased coin, with faces labeled 0 and 1, until the sequence of outcomes is a member of S; at that point, stop. For each σ ∈ S, let P(σ) be the probability that this process generates the string σ. Note that since Sis prefix-free, for every σ ∈S, if the coin toss outcomes follow the bits of σ then we will stop only once the sequence of outcomes equals σ. We therefore get that, for every σ ∈S,P(σ)=1.Sin2|σ|ceprobabilitiesadduptoatmost1,ourproof is concluded.

In light of Kraft's inequality, any prefix-free description language of a hypothesis class, H, gives rise to a weighting function w over that hypothesis class – we will simply set w(h) = 1 . This observation immediately yields the following:
2|h|

theorem 7.7 Let beahypothesisclassandlet H→{ beaprefix-freedescriptionlanguagefor . Then,foreverysamplesize, ,everyconfidence parameter, δ > , and every probability distribution, D, with probability greater than 1−δ overthechoiceof wehavethat, S D

H d:0,1}∗

∀ ∈H|h|+ln(2/δ)hD()LS(h)+ ,2m ≤

0

where |h|isthelengthof ( ). d h

ProofChoosew(h)=12|h|,applyTheorem7.4withε(m,δ)=ln(2/δ)n2m,andnotethatln(2|h|)= |h|ln(2) ||.

As was the case with Theorem 7.4, this result suggests a learning paradigm for H – given a train ingset,S,searchforahypothesish∈Hthatminimizesthebound,L()+|h|+ln(2/δ)Sh2m.Inparticular,itsugg learning paradigm.

Minimum Description Length (MDL)

prior knowledge:
H is a countable hypothesis class
H is described by a prefix-free language over {0,1}
For every h ∈ H, |h| is the length of the representation of h

[hinput:Atrainingset Dm,confidenceδ]output:h∈argminL(h)+|h|+ln(2/δ)h∈HS2m

Example7.3 Let H be the class of all predictors that can be implemented using some programming language, say, C++. Let us represent each program using the

binary string obtained by running the gzip command on the program (this yields a prefix-free description language over the alphabet {0,1}). Then, |h| is simply the length (in bits) of the output of gzip when running on the C++ program corresponding to h.

7.3.1 Occam's Razor

Theorem 7.7 suggests that, having two hypotheses sharing the same empirical risk, the true risk of the one that has shorter description can be bounded by a lower value. Thus, this result can be viewed as conveying a philosophical message:

A short explanation (that is, a hypothesis that has a short length) tends to be more valid than a long explanation.

This is a well known principle, called Occam's razor, after William of Ockham, a 14th-century English logician, who is believed to have been the first to phrase it explicitly. Here, we provide one possible justification to this principle. The inequality of Theorem 7.7 shows that the more complex a hypothesis h is (in the sense of having a longer description), the larger the sample size it has to fit to guarantee that it has a small true risk, $L_D(h)$.

At a second glance, our Occam razor claim might seem somewhat problematic. In the context in which the Occam razor principle is usually invoked in science, the language according to which complexity is measured is a natural language, whereas here we may consider any arbitrary abstract description language. Assume that we have two hypotheses such that |h′| is much smaller than |h|. By the preceding result, if both have the same error on a given training set, S, then the true error of h may be much higher than the true error of h′, so one should prefer h′ over h. However, we could have chosen a different description language, say, one that assigns a string of length 3 to h and a string of length 100000 to h′. Suddenly it looks as if one should prefer h over h′. But these are the same h and h′ for which we argued two sentences ago that h′ should be preferable. Where is the catch here?

Indeed, there is no inherent generalizability difference between hypotheses. The crucial aspect here is the dependency order between the initial choice of language (or, preference over hypotheses) and the training set. As we know from the basic Hoeffding

√'sbound(Equation(4.2)),ifwecommittoanyhypothesisbeforeseeingthedata,thenweareguaranteedarathersmallestimationerrortermLln(2/δ)D(h)≤LS(h)+2m.Choosingadescriptionlanguage(or,equivalently,someweightingofhypotheses)isaweakformofcommittingtoahypothesis.

Rather than committing to a single hypothesis, we spread out our commitment among many. As long as it is done independently of the training sample, our generalization bound holds. Just as the choice of a single hypothesis to be evaluated by a sample can be arbitrary, so is the choice of description language.

7.4 Other Notions of Learnability – Consistency

The notion of learnability can be further relaxed by allowing the needed sample sizes to depend not only on ε, δ, and h but also on the underlying data-generating probability distribution D (that is used to generate the training sample and to determine the risk). This type of performance guarantee is captured by the notion of consistency[1] of a learning rule.

definition 7.8 (Consistency) Let Z be a domain set, let P be a set of probability distributions over Z, and let H be a hypothesis class. A learning rule A is consistent with respect to H and P if there exists a function mCON : (0 1)2 H , ×H×P→Nsuchthat,foreveryε,δ∈(0,1),everyh∈H,and every D∈P,ifm≥mNULH(ε,δ,h,D)thenwithprobabilityofatleast1−δover the choice of S Dm it holds that L D(A(S)) ≤ LD(h) + ε.

If P is the set of all distributions,[2] we say that A is universally consistent with respect to H.

The notion of consistency is, of course, a relaxation of our previous notion of nonuniform learnability. Clearly if an algorithm nonuniformly learns a class H it is also universally consistent for that class. The relaxation is strict in the sense that there are consistent learning rules that are not successful nonuniform learners. For example, the algorithm Memorize defined in Example 7.4 later is universally consistent for the class of all binary classifiers over N. However, as we have argued before, this class is not nonuniformly learnable.

Example7.4 Considertheclassificationpredictionalgorithm Memorizedefined as follows. The algorithm memorizes the training examples, and, given a test point x, it predicts the majority label among all labeled instances of x that exist in the training sample (and some fixed default label if no instance of x appears in the training set). It is possible to show (see Exercise 6) that the Memorize algorithm is universally consistent for every countable domain X and a finite label set Y (w.r.t the zero-one loss).

Intuitively, it is not obvious that the Memorize algorithm should be viewed as a learner, since it lacks the aspect of generalization, namely, of using observed data to predict the labels of unseen examples. The fact that Memorize is a consistent algorithm for the class of all functions over any countable domain set therefore raises doubt about the usefulness of consistency guarantees. Furthermore, the sharp-eyed reader may notice that the "bad learner" we introduced in Chapter 2,

[1] In the literature, consistency is often defined using the notion of either convergence in probability (corresponding to weak consistency) or almost sure convergence (corresponding to strong consistency).

[2] Formally, we assume that Z is endowed with some sigma algebra of subsets Ω, and by "all distributions" we mean all probability distributions that have Ω as their associated family of measurable subsets.

which led to overfitting, is in fact the Memorize algorithm. In the next section we discuss the significance of the different notions of learnability and revisit the No-Free-Lunch theorem in light of the different definitions of learnability.

7.5 Discussing the Different Notions of Learnability

We have given three definitions of learnability and we now discuss their usefulness. As is usually the case, the usefulness of a mathematical definition depends on what we need it for. We therefore list several possible goals that we aim to achieve by defining learnability and discuss the usefulness of the different definitions in light of these goals.

What Is the Risk of the Learned Hypothesis?

The first possible goal of deriving performance guarantees on a learning algorithm is bounding the risk of the output predictor. Here, both PAC learning and nonuniform learning give us an upper bound on the true risk of the learned hypothesis based on its empirical risk. Consistency guarantees do not provide such a bound. However, it is always possible to estimate the risk of the output predictor using a validation set (as will be described in Chapter 11).

How Many Examples Are Required to Be as Good as the Best Hypothesis ~~in~~?

When approaching a learning problem, a natural question is how many examples we need to collect in order to learn it. Here, PAC learning gives a crisp answer. However, for both nonuniform learning and consistency, we do not know in advance how many examples are required to learn $\mathcal{H}$. In nonuniform learning this number depends on the best hypothesis in $\mathcal{H}$, and in consistency it also depends on the underlying distribution. In this sense, PAC learning is the only useful definition of learnability. On the flip side, one should keep in mind that even if the estimation error of the predictor we learn is small, its risk may still be large if $\mathcal{H}$ has a large approximation error. So, for the question "How many examples are required to be as good as the Bayes optimal predictor?" even PAC guarantees do not provide us with a crisp answer. This reflects the fact that the usefulness of PAC learning relies on the quality of our prior knowledge.

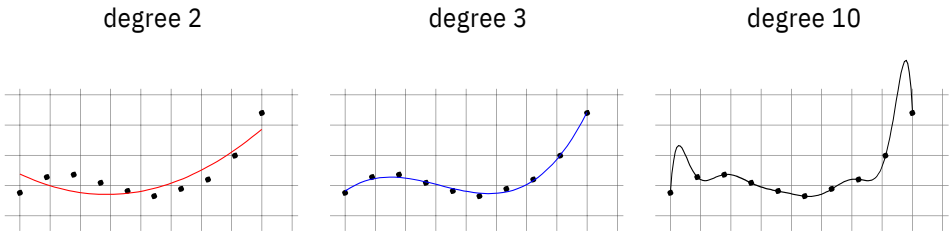
PAC guarantees also help us to understand what we should do next if our learning algorithm returns a hypothesis with a large risk, since we can bound the part of the error that stems from estimation error and therefore know how much of the error is attributed to approximation error. If the approximation error is large, we know that we should use a different hypothesis class. Similarly, if a nonuniform algorithm fails, we can consider a different weighting function over (subsets of) hypotheses. However, when a consistent algorithm fails, we have no idea whether this is because of the estimation error or the approximation error. Furthermore, even if we are sure we have a problem with the estimation

error term, we do not know how many more examples are needed to make the estimation error small.

How to Learn? How to Express Prior Knowledge?

Maybe the most useful aspect of the theory of learning is in providing an answer to the question of "how to learn." The definition of PAC learning yields the limitation of learning (via the No-Free-Lunch theorem) and the necessity of prior knowledge. It gives us a crisp way to encode prior knowledge by choosing a hypothesis class, and once this choice is made, we have a generic learning rule – ERM. The definition of nonuniform learnability also yields a crisp way to encode prior knowledge by specifying weights over (subsets of) hypotheses of $\mathcal{H}$. Once this choice is made, we again have a generic learning rule – SRM. The SRM rule is also advantageous in model selection tasks, where prior knowledge is partial. We elaborate on model selection in Chapter 11 and here we give a brief example.

Consider the problem of fitting a one dimensional polynomial to data; namely, our goal is to learn a function, $h : \mathbb{R} \to \mathbb{R}$, and as prior knowledge we consider the hypothesis class of polynomials. However, we might be uncertain regarding which degree d would give the best results for our data set: A small degree might not fit the data well (i.e., it will have a large approximation error), whereas a high degree might lead to overfitting (i.e., it will have a large estimation error). In the following we depict the result of fitting a polynomial of degrees 2, 3, and 10 to the same training set.



It is easy to see that the empirical risk decreases as we enlarge the degree. Therefore, if we choose $\mathcal{H}$ to be the class of all polynomials up to degree 10 then the ERM rule with respect to this class would output a 10 degree polynomial and would overfit. On the other hand, if we choose too small a hypothesis class, say, polynomials up to degree 2, then the ERM would suffer from underfitting (i.e., a large approximation error). In contrast, we can use the SRM rule on the set of all polynomials, while ordering subsets of $\mathcal{H}$ according to their degree, and this will yield a 3rd degree polynomial since the combination of its empirical risk and the bound on its estimation error is the smallest. In other words, the SRM rule enables us to select the right model on the basis of the data itself. The price we pay for this flexibility (besides a slight increase of the estimation error relative to PAC learning w.r.t. the optimal degree) is that we do not know in

advance how many examples are needed to compete with the best hypothesis in H.

Unlike the notions of PAC learnability and nonuniform learnability, the definition of consistency does not yield a natural learning paradigm or a way to encode prior knowledge. In fact, in many cases there is no need for prior knowledge at all. For example, we saw that even the Memorize algorithm, which intuitively should not be called a learning algorithm, is a consistent algorithm for any class defined over a countable domain and a finite label set. This hints that consistency is a very weak requirement.

Which Learning Algorithm Should We Prefer?

One may argue that even though consistency is a weak requirement, it is desirable that a learning algorithm will be consistent with respect to the set of all functions from X to Y, which gives us a guarantee that for enough training examples, we will always be as good as the Bayes optimal predictor. Therefore, if we have two algorithms, where one is consistent and the other one is not consistent, we should prefer the consistent algorithm. However, this argument is problematic for two reasons. First, maybe it is the case that for most "natural" distributions we will observe in practice that the sample complexity of the consistent algorithm will be so large so that in every practical situation we will not obtain enough examples to enjoy this guarantee. Second, it is not very hard to make any PAC or nonuniform learner consistent with respect to the class of all functions from X to Y. Concretely, consider a countable domain, X, a finite label set Y, and a hypothesis class, H, of functions from X to Y. We can make any nonuniform learner for H be consistent with respect to the class of all classifiers from X to Y using the following simple trick: Upon receiving a training set, we will first run the nonuniform learner over the training set, and then we will obtain a bound on the true risk of the learned predictor. If this bound is small enough we are done. Otherwise, we revert to the Memorize algorithm. This simple modification makes the algorithm consistent with respect to all functions from X to Y. Since it is easy to make any algorithm consistent, it may not be wise to prefer one algorithm over the other just because of consistency considerations.

7.5.1 The No-Free-Lunch Theorem Revisited

Recall that the No-Free-Lunch theorem (Theorem 5.1 from Chapter 5) implies that no algorithm can learn the class of all classifiers over an infinite domain. In contrast, in this chapter we saw that the Memorize algorithm is consistent with respect to the class of all classifiers over a countable infinite domain. To understand why these two statements do not contradict each other, let us first recall the formal statement of the No-Free-Lunch theorem.

Let X be a countable infinite domain and let $Y = \{\pm 1\}$. The No-Free-Lunch theorem implies the following: For any algorithm, A, and a training set size, m, there exist a distribution over $X \times Y$ and a function $h : X \to Y$, such that if A

will get a sample of m i.i.d. training examples, labeled by h?, then A is likely to return a classifier with a larger error.

The consistency of Memorize implies the following: For every distribution over X and a labeling function $h^? : X \to Y$, there exists a training set size m (that depends on the distribution and on h?) such that if Memorize receives at least m examples it is likely to return a classifier with a small error.

We see that in the No-Free-Lunch theorem, we first fix the training set size, and then find a distribution and a labeling function that are bad for this training set size. In contrast, in consistency guarantees, we first fix the distribution and the labeling function, and only then do we find a training set size that suffices for learning this particular distribution and labeling function.

7.6 Summary

We introduced nonuniform learnability as a relaxation of PAC learnability and consistency as a relaxation of nonuniform learnability. This means that even classes of infinite VC-dimension can be learnable, in some weaker sense of learnability. We discussed the usefulness of the different definitions of learnability.

For hypothesis classes that are countable, we can apply the Minimum Description Length scheme, where hypotheses with shorter descriptions are preferred, following the principle of Occam's razor. An interesting example is the hypothesis class of all predictors we can implement in C++ (or any other programming language), which we can learn (nonuniformly) using the MDL scheme.

Arguably, the class of all predictors we can implement in C++ is a powerful class of functions and probably contains all that we can hope to learn in prac- tice. The ability to learn this class is impressive, and, seemingly, this chapter should have been the last chapter of this book. This is not the case, because of the computational aspect of learning: that is, the runtime needed to apply the learning rule. For example, to implement the MDL paradigm with respect to all C++ programs, we need to perform an exhaustive search over all C++ pro- grams, which will take forever. Even the implementation of the ERM paradigm with respect to all C++ programs of description length at most 1000 bits re- quires an exhaustive search over 21000 hypotheses. While the sample complexity of learning this class is just $\frac{1000+\log(2/\delta)}{\epsilon} \geq 10002$, the runtime is 2^{1000}. This is a huge number – much larger than the number of atoms in the visible universe. In the next chapter we formally define the computational complexity of learning. In the second part of this book we will study hypothesis classes for which the ERM or SRM schemes can be implemented efficiently.

7.7 Bibliographic Remarks

Our definition of nonuniform learnability is related to the definition of an Occam-algorithm in Blumer, Ehrenfeucht, Haussler & Warmuth (1987). The concept of SRM is due to (Vapnik & Chervonenkis 1974, Vapnik 1995). The concept of MDL is due to (Rissanen 1978, Rissanen 1983). The relation between SRM and MDL is discussed in Vapnik (1995). These notions are also closely related to the notion of regularization (e.g. Tikhonov (1943)). We will elaborate on regularization in the second part of this book.

The notion of consistency of estimators dates back to Fisher (1922). Our presentation of consistency follows Steinwart & Christmann (2008), who also derived several no-free-lunch theorems.

7.8 Exercises

1. Prove that for any finite class H, and any description language $d : H \to \{0,1\}^*$, the VC-dimension of H is at most $2^{\sup\{|d(h)| : h \in H\}}$ – the maximum description length of a predictor in H. Furthermore, if d is a prefix-free description then $\mathrm{VCdim}(H) \leq \sup\{|d(h)| : h \in H\}$.

2. Let H = {hn : n ∈ N} be an infinite countable hypothesis class for binary classification. Show that it is impossible to assign weights to the hypotheses in H such that

• Hcouldbelearntnonuniformlyusingtheseweights.Th

∑atis,theweightingfunctionw:H→[0,1]shouldsatisfytheconditionh∈Hw(h)≤1.•Theweightswouldbemonotonicallyn

w (hi) ≤ w(hj).

∪∞3.•ConsiderahypothesisclassH=n=1Hn,whereforevery

∑n∈N,Hnisfinite.Findaweightingfunctionw:H→[0,1]suchthath∈Hw(h)≤1andsothatforallh∈H,w(h)isdeterminedb

Hn} and by |Hn(h)|.

• (*) Define such a function w when for all n Hn is countable (possibly infinite).

4. Let H be some hypothesis class. For any h ∈ H, let |h| denote the description length of h, according to some fixed description language. Consider the MDL learning paradigm in which the

[algorithmvreturns:]∈|h|+ln(2/δ)hSargminLS(h)+,h∈H2m

where S is a sample of size m. For any B > 0, let

HB={h∈H:|h|≤B},

and define

h∗B=argminL D(h).hB

Prove a bound on L (h) −Lh ∗ DSD(B)intermsofB,theconfidenceparameter δ, and the size of the training set m.

- Note: Such bounds are known as oracle inequalities in the literature: We wish to estimate how good we are compared to a reference classifier (or "oracle") h ∗ B .

5. In this question we wish to show a No-Free-Lunch result for nonuniform learnability: namely, that, over any infinite domain, the class of all functions is not learnable even under the relaxed nonuniform variation of learning.

Recall that an algorithm, A, nonuniformly learns a hypothesis class Hif thereexistsafunctionmNUL 2 H :(0,1) ×H→Nsuchthat,foreveryε,δ∈(0,1) and for every h ∈ H, if m ≥ mNULH (ε, δ, h) then for every distribution D, with probability of at least 1 − δ over the choice of S Dm, it holds that L D(A(S)) ≤ LD(h) + ε.

If such an algorithm exists then we say that H is nonuniformly learnable.

1. Let A be a nonuniform learner for a class H. For each n ∈ N define HAn = {h ∈ H : mNUL(0.1, 0.1, h) ≤ n}. Prove that each such class Hn has a finite VC-dimension.
2. Prove that if UaclassHisnonuniformlylearnablethentherareclassesHnsothatH=n∈NHnand,foreverynn∈N,VCdim(H Uiteset.Then,foreverysequenceofclasses(Hn:n∈N)suchthatH=n∈NHn,thereexistssomenforwhichVC

Hint: Given a class H that shatters some infinite set K, and a sequence of classes (Hn : n ∈ N), each having a finite VC-dimension, start by defining subsets Kn ⊆ K such that, for all n, |Kn| > VCdim(Hn) and for any n 6= m, Kn ∩ Km = ∅. Now, pick for each such Kn a function fn : Kn → {0, 1} so that no h ∈ Hn agrees with fn on the domain Kn

(.FinaUlly,defin)ef:X→{0,1}bycombiningthesefn'sandprovethatf∈H\n∈NHn.4.ConstructaclassH1offu is nonuniformly learnable but not PAC learnable.

5. Construct a class H2 of functions from the unit interval [0, 1] to {0, 1} that is not nonuniformly learnable.

6. In this question we wish to show that the algorithm Memorize is a consistent learner for every class of (binary-valued) functions over any countable domain. Let X be a countable domain and let D be a probability distribution over X.

1. Let {xi : i ∈ N} be an enumeration of the elements of X so that for all i ≤ j, D({xi}) ≤ D({xj}). Prov ∑ethatlim(xi})=0.nin

2. Given any ε > 0 prove that there existsD> 0 such that D { ∈ X (x):(x)≤εD }) < ε.

3. Prove that for every $\eta > 0$, if n is such that $D(\{x_i\}) < \eta$ for all $i > n$, then for every $m \in \mathbb{N}$,

$$P\left[\exists x_{n+1},\ldots,x_m : (D(\{x_i\}) > \eta \text{ and } x_i \in S)\right] \leq ne^{-\eta m}.$$

4. Conclude that if X is countable then for every probability distribution D over X there exists a function $m_D : (0, 1) \times (0, 1) \to \mathbb{N}$ such that for every $\varepsilon, \delta > 0$ if $m > m_D(\varepsilon, \delta)$ then

$$P\left[D(\{x : x \in S\}) > \varepsilon\right] < \delta.$$

5. Prove that Memorize is a consistent learner for every class of (binary-valued) functions over any countable domain.

8 The Runtime of Learning

So far in the book we have studied the statistical perspective of learning, namely, how many samples are needed for learning. In other words, we focused on the amount of information learning requires. However, when considering automated learning, computational resources also play a major role in determining the complexity of a task: that is, how much computation is involved in carrying out a learning task. Once a sufficient training sample is available to the learner, there is some computation to be done to extract a hypothesis or figure out the label of a given test instance. These computational resources are crucial in any practical application of machine learning. We refer to these two types of resources as the sample complexity and the computational complexity. In this chapter, we turn our attention to the computational complexity of learning.

The computational complexity of learning should be viewed in the wider context of the computational complexity of general algorithmic tasks. This area has been extensively investigated; see, for example, (Sipser 2006). The introductory comments that follow summarize the basic ideas of that general theory that are most relevant to our discussion.

The actual runtime (in seconds) of an algorithm depends on the specific machine the algorithm is being implemented on (e.g., what the clock rate of the machine's CPU is). To avoid dependence on the specific machine, it is common to analyze the runtime of algorithms in an asymptotic sense. For example, we say that the computational complexity of the merge-sort algorithm, which sorts a list of n items, is O(nlog(n)). This implies that we can implement the algo- rithm on any machine that satisfies the requirements of some accepted abstract model of computation, and the actual runtime in seconds will satisfy the following: there exist constants c and n0, which can depend on the actual machine, such that, for any value of n > n0, the runtime in seconds of sorting any n items will be at most cnlog(n). It is common to use the term feasible or efficiently computable for tasks that can be performed by an algorithm whose running time is O(p(n)) for some polynomial function p. One should note that this type of analysis depends on defining what is the input size n of any instance to which the algorithm is expected to be applied. For "purely algorithmic" tasks, as discussed in the common computational complexity literature, this input size is clearly defined; the algorithm gets an input instance, say, a list to be sorted, or an arithmetic operation to be calculated, which has a well defined size (say, the

number of bits in its representation). For machine learning tasks, the notion of an input size is not so clear. An algorithm aims to detect some pattern in a data set and can only access random samples of that data.

We start the chapter by discussing this issue and define the computational complexity of learning. For advanced students, we also provide a detailed formal definition. We then move on to consider the computational complexity of implementing the ERM rule. We first give several examples of hypothesis classes where the ERM rule can be efficiently implemented, and then consider some cases where, although the class is indeed efficiently learnable, ERM implementation is computationally hard. It follows that hardness of implementing ERM does not imply hardness of learning. Finally, we briefly discuss how one can show hardness of a given learning task, namely, that no learning algorithm can solve it efficiently.

8.1 Computational Complexity of Learning

Recall that a learning algorithm has access to a domain of examples, Z, a hypothesis class, H, a loss function, `, and a training set of examples from Z that are sampled i.i.d. according to an unknown distribution D. Given parameters ϵ, δ, the algorithm should output a hypothesis h such that with probability of at least $1 - \delta$,

$$L(h) \leq \min_{h' \in H} L(h') + \epsilon.$$

As mentioned before, the actual runtime of an algorithm in seconds depends on the specific machine. To allow machine independent analysis, we use the standard approach in computational complexity theory. First, we rely on a notion of an abstract machine, such as a Turing machine (or a Turing machine over the reals (Blum, Shub & Smale 1989)). Second, we analyze the runtime in an asymptotic sense, while ignoring constant factors, thus the specific machine is not important as long as it implements the abstract machine. Usually, the asymptote is with respect to the size of the input to the algorithm. For example, for the merge-sort algorithm mentioned before, we analyze the runtime as a function of the number of items that need to be sorted.

In the context of learning algorithms, there is no clear notion of "input size." One might define the input size to be the size of the training set the algorithm receives, but that would be rather pointless. If we give the algorithm a very large number of examples, much larger than the sample complexity of the learning problem, the algorithm can simply ignore the extra examples. Therefore, a larger training set does not make the learning problem more difficult, and, consequently, the runtime available for a learning algorithm should not increase as we increase the size of the training set. Just the same, we can still analyze the runtime as a function of natural parameters of the problem such as the target accuracy, the confidence of achieving that accuracy, the dimensionality of the

domain set, or some measures of the complexity of the hypothesis class with which the algorithm's output is compared.

To illustrate this, consider a learning algorithm for the task of learning axis aligned rectangles. A specific problem of learning axis aligned rectangles is derived by specifying ε, δ, and the dimension of the instance space. We can define a sequence of problems of the type "rectangles learning" by fixing ε, δ and varying the dimension to be d = 2, 3, 4, ... We can also define another sequence of "rect- angles learning" problems by fixing d, δ and varying the target accuracy to be ε = 1 1 2, 3, ... One can of course choose other sequences of such problems. Once a sequence of the problems is fixed, one can analyze the asymptotic runtime as a function of variables of that sequence.

Before we introduce the formal definition, there is one more subtlety we need to tackle. On the basis of the preceding, a learning algorithm can "cheat," by transferring the computational burden to the output hypothesis. For example, the algorithm can simply define the output hypothesis to be the function that stores the training set in its memory, and whenever it gets a test example x it calculates the ERM hypothesis on the training set and applies it on x. Note that in this case, our algorithm has a fixed output (namely, the function that we have just described) and can run in constant time. However, learning is still hard – the hardness is now in implementing the output classifier to obtain a label prediction. To prevent this "cheating," we shall require that the output of a learning algorithm must be applied to predict the label of a new example in time that does not exceed the runtime of training (that is, computing the output classifier from the input training sample). In the next subsection the advanced reader may find a formal definition of the computational complexity of learning.

8.1.1 Formal Definition*

The definition that follows relies on a notion of an underlying abstract machine, which is usually either a Turing machine or a Turing machine over the reals. We will measure the computational complexity of an algorithm using the number of "operations" it needs to perform, where we assume that for any machine that implements the underlying abstract machine there exists a constant c such that any such "operation" can be performed on the machine using c seconds.

definition 8.1 (The Computational Complexity of a Learning Algorithm)

We define the complexity of learning in two steps. First we consider the computational complexity of a fixed learning problem (determined by a triplet (Z, H, `) – a domain set, a benchmark hypothesis class, and a loss function). Then, in the second step we consider the rate of change of that complexity along a sequence of such tasks.

1. Given a function f : (0,1)2 → N, a learning task (Z,H, `), and a learning algorithm, A, we say that A solves the learning task in time O(f) if there exists some constant number c, such that for every probability distribution D

over Z, and input ε, δ ∈ (0, 1), when A has access to samples generated i.i.d. by D,

• A terminates after performing at most cf(ε, δ) operations

• The output of A, denoted hA, can be applied to predict the label of a new example while performing at most cf(ε, δ) operations

• The output of A is probably approximately correct; namely, with probability of at least 1 − δ (over the random samples A receives), LD(hA) ≤ min′ ′h ∈H LD(h) + ε

2. Consider a sequence of learning problems, (Zn, H ∞n, ` n)n=1,whereproblemn is defined by a domain Zn, a hypothesis class Hn, and a loss function ` n.

Let A be a learning algorithm designed for solving learning problems of this form. Given a function g : N × (0,1)2 → N, we say that the runtime of A with respect to the preceding sequence is O(g), if for all n, A solves the problem(Z 2n, Hn, ` n) in time O(fn), where fn : (0,1) → N is defined by fn(ε, δ) = g(n, ε, δ).

We say that A is an efficient algorithm with respect to a sequence (Zn, Hn, ` n) if its runtime is O(p(n, 1/ε, 1/δ)) for some polynomial p.

From this definition we see that the question whether a general learning problem can be solved efficiently depends on how it can be broken into a sequence of specific learning problems. For example, consider the problem of learning a finite hypothesis class. As we showed in previous chapters, the ERM rule over H is guaranteed to (ε,δ)-learn H if the number of training examples is order of m ( ) 2 H(ε,δ) = log |H|/δ /ε . Assuming that the evaluation of a hypothesis on an example takes a constant time, it is possible to implement the ERM rule in time O(|H|mH(ε,δ)) by performing an exhaustive search over H with a training set of size m H(ε,δ). For any fixed finite H, the exhaustive search algorithm runs in polynomial time. Furthermore, if we define a sequence of problems in which |Hn| = n, then the exhaustive search is still considered to be efficient. However, if we define a sequence of problems for which |Hn| = 2n, then the sample complexity is still polynomial in n but the computational complexity of the exhaustive search algorithm grows exponentially with n (thus, rendered inefficient).

8.2 Implementing the ERM Rule

Given a hypothesis class, H, the ERM H rule is maybe the most natural learning paradigm. Furthermore, for binary classification problems we saw that if learning is at all possible, it is possible with the ERM rule. In this section we discuss the computational complexity of implementing the ERM rule for several hypothesis classes.

Given a hypothesis class, H, a domain set Z, and a loss function ` , the corresponding ERM H rule can be defined as follows:

On a finite in $S \in Z^m$ output some $h \in H$ that minimizes the empirical loss, ∑putsamplteLh1S()=|S|z∈S().

This section studies the runtime of implementing the ERM rule for several examples of learning tasks.

8.2.1 Finite Classes

Limiting the hypothesis class to be a finite class may be considered as a reasonably mild restriction. For example, H can be the set of all predictors that can be implemented by a C++ program written in at most 10000 bits of code. Other examples of useful finite classes are any hypothesis class that can be parameterized by a finite number of parameters, where we are satisfied with a representation of each of the parameters using a finite number of bits, for example, the class of axis aligned rectangles in the Euclidean space, Rd, when the parameters defining any given rectangle are specified up to some limited precision.

As we have shown in previous chapters, the sample complexity of learning a finiteclassisupperboundedbym c H(ε,δ) = clog(c|H|/δ)/ε , where c = 1 in the realizable case and c = 2 in the nonrealizable case. Therefore, the sample complexity has a mild dependence on the size of H. In the example of C++ programs mentioned before, the number of hypotheses is 210,000 but the sample complexity is only c(10, 000 + log(c/δ))/εc.

A straightforward approach for implementing the ERM rule over a finite hypothesis class is to perform an exhaustive search. That is, for each h ∈Hwe calculate the empirical risk, LS(h), and return a hypothesis that minimizes the empirical risk. Assuming that the evaluation of ` (h,z) on a single example takes a constant amount of time, k, the runtime of this exhaustive search becomes k |H|m, where m is the size of the training set. If we let m to be the upper bound on the sample complexity mentioned, then the runtime becomes k†H|c log(c|H|/δ)/εc.

The linear dependence of the runtime on the size of H makes this approach inefficient (and unrealistic) for large classes. Formally, if we define a sequence of problems(Z ∞n, Hn, ` n)n=1 such that log(|Hn|) = n, then the exhaustive search approach yields an exponential runtime. In the example of C++ programs, if Hn is the set of functions that can be implemented by a C++ program written in at most n bits of code, then the runtime grows exponentially with n, implying that the exhaustive search approach is unrealistic for practical use. In fact, this problem is one of the reasons we are dealing with other hypothesis classes, like classes of linear predictors, which we will encounter in the next chapter, and not just focusing on finite classes.

It is important to realize that the inefficiency of one algorithmic approach (such as the exhaustive search) does not yet imply that no efficient ERM implementation exists. Indeed, we will show examples in which the ERM rule can be implemented efficiently.

8.2.2 Axis Aligned Rectangles

Let H^n be the class of axis aligned rectangles in R^n, namely,

$$H^n = \{h(a_1,\ldots,a_n,b_1,\ldots,b_n) : \forall i, a_i \leq b_i\}$$

where

$$h_{(a_1,\ldots,a_n,b_1,\ldots,b_n)}(x_1,\ldots,x_n) = \begin{cases} 1 & \text{if } \forall i,\ x_i \in [a_i, b_i] \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

Efficiently Learnable in the Realizable Case

Consider implementing the ERM rule in the realizable case. That is, we are given a training set $S = (x_1,y_1),\ldots,(x_m,y_m)$ of examples, such that there exists an axis aligned rectangle, $h \in H^n$, for which $h(x_i) = y_i$ for all i. Our goal is to find such an axis aligned rectangle with a zero training error, namely, a rectangle that is consistent with all the labels in S.

We show later that this can be done in time $O(nm)$. Indeed, for each $i \in [n]$, set $a_i = \min\{x_i : (x,1) \in S\}$ and $b_i = \max\{x_i : (x,1) \in S\}$. In words, we take a_i to be the minimal value of the i'th coordinate of a positive example in S and b_i to be the maximal value of the i'th coordinate of a positive example in S.

It is easy to verify that the resulting rectangle has zero training error and that the runtime of finding each a_i and b_i is $O(m)$. Hence, the total runtime of this procedure is $O(nm)$.

Not Efficiently Learnable in the Agnostic Case

In the agnostic case, we do not assume that some hypothesis h perfectly predicts the labels of all the examples in the training set. Our goal is therefore to find h that minimizes the number of examples for which $y_i = h(x_i)$. It turns out that for many common hypothesis classes, including the classes of axis aligned rectangles we consider here, solving the ERM problem in the agnostic setting is NP-hard (and, in most cases, it is even NP-hard to find some $h \in H$ whose error is no more than some constant $c > 1$ times that of the empirical risk minimizer in H). That is, unless P = NP, there is no algorithm whose running time is polynomial in m and n that is guaranteed to find an ERM hypothesis for these problems (Ben-David, Eiron & Long 2003).

On the other hand, it is worthwhile noticing that, if we fix one specific hypothesis class, say, axis aligned rectangles in some fixed dimension, n, then there exist efficient learning algorithms for this class. In other words, there are successful agnostic PAC learners that run in time polynomial in $1/\epsilon$ and $1/\delta$ (but their dependence on the dimension n is not polynomial).

To see this, recall the implementation of the ERM rule we presented for the realizable case, from which it follows that an axis aligned rectangle is determined by at most $2n$ examples. Therefore, given a training set of size m, we can perform an exhaustive search over all subsets of the training set of size at most $2n$ examples and construct a rectangle from each such subset. Then, we can pick

the rectangle with the minimal training error. This procedure is guaranteed to find an ERM hypothesis, and the runtime of the procedure is mO(n). It follows that if n is fixed, the runtime is polynomial in the sample size. This does not contradict the aforementioned hardness result, since there we argued that unless P=NP one cannot have an algorithm whose dependence on the dimension n is polynomial as well.

8.2.3 Boolean Conjunctions

A Boolean conjunction is a mapping from X = {0,1}n to Y = {0,1} that can be expressed as a proposition formula of the form xi ∧...∧xi∧¬xj∧...∧¬x,1k1jr for some indices i1, . . . , ik, j1, . . . , jr ∈ [n]. The function that such a proposition formula defines is

{1ifxi=⋯=xi=1andxj=⋯=xj=0h(x)=1k1r0otherwise

Let HnbetheclassofallBooleanconjunctionsover{01}n.ThesizeofHnC,Cis at most 3n +1 (since in a conjunction formula, each element of x either appears, or appears with a negation sign, or does not appear at all, and we also have the all negative formula). Hence, the sample complexity of learning HnCusingthe ERM rule is at most Log(3/δ)ε.

Efficiently Learnable in the Realizable Case

Next, we show that it is possible to solve the ERM problem for HnCintime polynomial in n and m. The idea is to define an ERM conjunction by including in the hypothesis conjunction all the literals that do not contradict any positively labeled example. Let v1,...,vm+ be all the positively labeled instances in the input sample S. We define, by induction on i ≤ m+, a sequence of hypotheses (or conjunctions). Let h0 be the conjunction of all possible literals. That is, h0 = x1 ∧ ¬x1 ∧ x2 ∧ . . . ∧ xn ∧ ¬xn. Note that h0 assigns the label 0 to all the elements of X . We obtain hi+1 by deleting from the conjunction hi all the literals that are not satisfied by vi+1. The algorithm outputs the hypothesis hm+. Note that hm+ labels positively all the positively labeled examples in S. Furthermore, for every i ≤ m+, hi is the most restrictive conjunction that labels v1,...,vi positively. Now, since we consider learning in the realizable setup, there exists a conjunction hypothesis, f ∈HnC,thatisconsistentwithalltheexamplesin S. Since hm+ is the most restrictive conjunction that labels positively all the positively labeled members of S, any instance labeled 0 by f is also labeled 0 by hm+ . It follows that hm+ has zero training error (w.r.t. S), and is therefore a legal ERM hypothesis. Note that the running time of this algorithm is O(mn).

Not Efficiently Learnable in the Agnostic Case

As in the case of axis aligned rectangles, unless P = NP, there is no algorithm whose running time is polynomial in m and n that guaranteed to find an ERM hypothesis for the class of Boolean conjunctions in the unrealizable case.

8.2.4 Learning 3-Term DNF

We next show that a slight generalization of the class of Boolean conjunctions leads to intractability of solving the ERM problem even in the realizable case. Consider the class of 3-term disjunctive normal form formulae (3-term DNF). The instance space is

X = {0,1}n and each hypothesis is represented by the Boolean formula of the form h(x) = A1(x)

$\vee$ A2(x) $\vee$ A3(x), where each Ai(x) is a Boolean conjunction (as defined in the previous section). The output of h(x) is 1 if either A1(x) or A2(x) or A3(x) outputs the label 1. If all three conjunctions output the label 0 then h(x) = 0.

Let

Hn3DNFbethehypothesisclassofallsuch3-termDNFformulae.Thesize of

Hnost33n.Hence,the Hn3DNFisatm samplecomplexityoflearning3DNFusing the ERM rule is at most 3n log(3/δ)/ε.

However, from the computational perspective, this learning problem is hard. It has been shown (see (Pitt & Valiant 1988, Kearns et al. 1994)) that unless RP = NP, there is no polynomial time algorithm that properly learns a sequence of 3-term DNF learning problems in which the dimension of the n'th problem is n. By "properly" we mean that the algorithm should output a hypothesis that is a 3-term DNF formula. In particular, since ERM

Hnoutputsa3-termDNF3DNF

formula it is a proper learner and therefore it is hard to implement it. The proof uses a reduction of the graph 3-coloring problem to the problem of PAC learning 3-term DNF. The detailed technique is given in Exercise 3. See also (Kearns & Vazirani 1994, Section 1.4).

8.3 Efficiently Learnable, but Not by a Proper ERM

In the previous section we saw that it is impossible to implement the ERM rule efficiently for the class

Hn3DNFof3-DNFformulae.Inthissectionweshowthatit is possible to learn this class efficiently, but using ERM with respect to a larger class.

Representation Independent Learning Is Not Hard

Next we show that it is possible to learn 3-term DNF formulae efficiently. There is no contradiction to the hardness result mentioned in the previous section as we now allow "representation independent" learning. That is, we allow the learning algorithm to output a hypothesis that is not a 3-term DNF formula. The ba- sic idea is to replace the original hypothesis class of 3-term DNF formula with a larger hypothesis class so that the new class is easily learnable. The learning

algorithm might return a hypothesis that does not belong to the original hypothesis class; hence the name "representation independent" learning. We emphasize that in most situations, returning a hypothesis with good predictive ability is what we are really interested in doing.

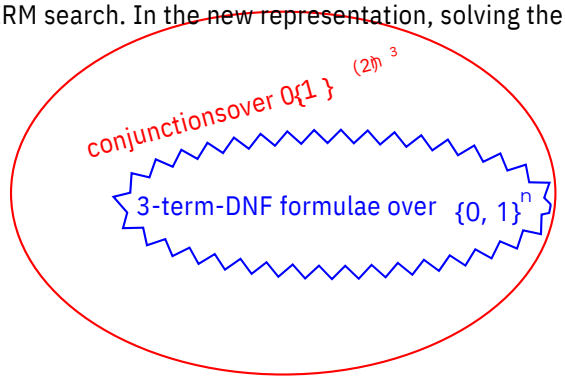
We start by noting that because $\vee$ distributes over $\wedge$, each 3-term DNF formula can be rewritten as

$$\bigwedge_{A_1} \bigvee_{A_2} \bigvee_{A_3} = \bigvee_{(u,v,w)\, u \in A_1, v \in A_2, w \in A_3}$$

Next, let us define: $\psi : \{0,1\}^n \to \{0,1\}^{(2n)^3}$ such that for each triple of literals u, v, w there is a variable in the range of ψ indicating if $u \vee v \vee w$ is true or false.

So, for each 3-DNF formula over $\{0,1\}^n$ there is a conjunction over $\{0,1\}^{(2n)^3}$, with the same truth table. Since we assume that the data is realizable, we can solve the ERM problem with respect to the class of conjunctions over $\{0,1\}^{(2n)^3}$. Furthermore, the sample complexity of learning the class of conjunctions in the higher dimensional space is at most $n^3 \log(1/\delta)/\varepsilon$. Thus, the overall runtime of this approach is polynomial in n.

Intuitively, the idea is as follows. We started with a hypothesis class for which learning is hard. We switched to another representation where the hypothesis class is larger than the original class but has more structure, which allows for a more efficient ERM search. In the new representation, solving the ERM problem is easy.



8.4 Hardness of Learning*

We have just demonstrated that the computational hardness of implementing ERM H does not imply that such a class H is not learnable. How can we prove that a learning problem is computationally hard?

One approach is to rely on cryptographic assumptions. In some sense, cryptography is the opposite of learning. In learning we try to uncover some rule underlying the examples we see, whereas in cryptography, the goal is to make sure that nobody will be able to discover some secret, in spite of having access

to some partial information about it. On that high level intuitive sense, results about the cryptographic security of some system translate into results about the unlearnability of some corresponding task. Regrettably, currently one has no way of proving that a cryptographic protocol is not breakable. Even the common assumption of P 6= NP does not suffice for that (although it can be shown to be necessary for most common cryptographic scenarios). The common approach for proving that cryptographic protocols are secure is to start with some cryptographic assumptions. The more these are used as a basis for cryptography, the stronger is our belief that they really hold (or, at least, that algorithms that will refute them are hard to come by).

We now briefly describe the basic idea of how to deduce hardness of learnability from cryptographic assumptions. Many cryptographic systems rely on the assumption that there exists a one way function. Roughly speaking, a one way function is a function f : {0,1}n → {0,1}n (more formally, it is a sequence of functions, one for each dimension n) that is easy to compute but is hard to invert. More formally, f can be computed in time poly(n) but for any randomized polynomial time algorithm A, and for every polynomial p(·),

$$P[f(A(f(x))) = f(x)] < 1p(n),$$

where the probability is taken over a random choice of x according to the uniform distribution over {0,1}n and the randomness of A.

A one way function, f, is called trapdoor one way function if, for some polynomial function p, for every n there exists a bit-string sn (called a secret key) of length ≤ p(n), such that there is a polynomial time algorithm that, for every n and every x ∈ {0, 1}n, on input (f(x), sn) outputs x. In other words, although f is hard to invert, once one has access to its secret key, inverting f becomes feasible. Such functions are parameterized by their secret key.

Now, let Fn be a family of trapdoor functions over {0,1}n that can be calculated by some polynomial time algorithm. That is, we fix an algorithm that given a secret key (representing one function in Fn) and an input vector, it calculates the value of the function corresponding to the secret key on the input vector in polynomial time. Consider the task of learning the class of the corresponding inverses, Hn = {f−1F:f∈Fn}.Sinceeachfunctioninthisclasscanbeinverted bysomesecretkeysofsizepolynomnHnnialin,theclass F can be parameterized by these keys and its size is at most 2p(n). Its sample complexity is therefore polynomial in n. We claim that there can be no efficient learner for this class. If there were such a learner, L, then by sampling uniformly at random a polynomial number of strings in {0,1}n, and computing f over them, we could generate a labeled training sample of pairs (f(x), x), which should suffice for our learner to figure out an (ε,δ) approximation of f−1 (w.r.t. the uniform distribution over the range of f), which would violate the one way property of f.

A more detailed treatment, as well as a concrete example, can be found in (Kearns & Vazirani 1994, Chapter 6). Using reductions, they also show that

the class of functions that can be calculated by small Boolean circuits is not efficiently learnable, even in the realizable case.

8.5 Summary

The runtime of learning algorithms is asymptotically analyzed as a function of different parameters of the learning problem, such as the size of the hypothe- sis class, our measure of accuracy, our measure of confidence, or the size of the domain set. We have demonstrated cases in which the ERM rule can be implemented efficiently. For example, we derived efficient algorithms for solving the ERM problem for the class of Boolean conjunctions and the class of axis aligned rectangles, under the realizability assumption. However, implementing ERM for these classes in the agnostic case is NP-hard. Recall that from the statistical perspective, there is no difference between the realizable and agnostic cases (i.e., a class is learnable in both cases if and only if it has a finite VC-dimension). In contrast, as we saw, from the computational perspective the difference is immense. We have also shown another example, the class of 3-term DNF, where implementing ERM is hard even in the realizable case, yet the class is efficiently learnable by another algorithm.

Hardness of implementing the ERM rule for several natural hypothesis classes has motivated the development of alternative learning methods, which we will discuss in the next part of this book.

8.6 Bibliographic Remarks

Valiant (1984) introduced the efficient PAC learning model in which the runtime of the algorithm is required to be polynomial in $1/\varepsilon$, $1/\delta$, and the representation size of hypotheses in the class. A detailed discussion and thorough bibliographic notes are given in Kearns & Vazirani (1994).

8.7 Exercises

1. Let H be the class of intervals on the line (formally equivalent to axis aligned rectangles in dimension $n = 1$). Propose an implementation of the ERM $_H$ learning rule (in the agnostic case) that given a training set of size m, runs in time $O(m^2)$.
Hint: Use dynamic programming.
2. Let $H_1, H_2, \ldots$ be a sequence of hypothesis classes for binary classification. Assume that there is a learning algorithm that implements the ERM rule in the realizable case such that the output hypothesis of the algorithm for each class H_n only depends on $O(n)$ examples out of the training set. Furthermore,

assume that such a hypothesis can be calculated given these O(n) examples in time O(n), and that the empirical risk of each such hypothesis can be evaluated in time O(mn). For example, if Hn is the class of axis aligned rectangles in Rn, we saw that it is possible to find an ERM hypothesis in the realizable case that is defined by at most 2n examples. Prove that in such cases, it is possible to find an ERM hypothesis for Hn in the unrealizable case in time O(mnmO(n)).

3. In this exercise, we present several classes for which finding an ERM classifier is computationally hard. First, we introduce the class of n-dimensional halfspaces, HS , for a domain X=Rnn.Thisistheclassofallfunctionsof the form hw,b(x) = sign(⟨w, x⟩ + b) where w, x ∈ Rn, ⟨w, x⟩ is their inner product, and b ∈ R. See a detailed description in Chapter 9.

1. Show that ERM H over the class H = HSn of linear predictors is computationally hard. More precisely, we consider the sequence of problems in which the dimension n grows linearly and the number of examples m is set to be some constant times n.

Max FS: Given a system of linear inequalities, Ax > v with A ∈ Rm×n and v ∈ Rm (that is a system of m linear inequalities in n variables, x = (x1,...,xn)), find a subsystem containing as many inequalities as possible that has a solution (such a subsystem is called feasible).

It has been shown (Sankaran 1993) that the problem Max FS is NP-hard.

Show that any algorithm that finds an ERMHS hypothesis for any training sample S ∈ (Rn ×{+1, −1})m can be used to solve the Max FS problem of size m,n. Hint: Define a mapping that transforms linear inequalities in n variables into labeled points in Rn, and a mapping that transforms vectors in Rn to halfspaces, such that a vector w satisfies an inequality q if and only if the labeled point that corresponds to q is classified correctly by the halfspace corresponding to w. Conclude that the problem of empirical risk minimization for halfspaces in also NP-hard (that is, if it can be solved in time polynomial in the sample size, m, and the Euclidean dimension, n, then every problem in the class NP can be solved in polynomial time).

2. Let X = Rn and let Hnk be the class of all intersections of k-many linear halfspaces in Rn. In this exercise, we wish to show that ERM Hniscom-k putationally hard for every k ≥ 3. Precisely, we consider a sequence of problems where k ≥ 3 is a constant and n grows linearly. The training set size, m, also grows linearly with n.

Given a graph G = (V, E) and a number k, determine whether there exists a so function : V → {1 . . k} that for every (u,v) E, f(u) = f(v).

Towards this goal, consider the k-coloring problem for graphs, defined as follows:

The k-coloring problem is known to be NP-hard for every k ≥ 3 (Karp 1972).

We wish to reduce the k-coloring problem to ERM
Hn:thatis,toprovek
that if there is an algorithm that solves the ERM
Hnproblemintimek
polynomial in k, n, and the sample size m, then there is a polynomial time algorithm for the graph k-coloring problem.
Given a graph G = (V,E), let
{v1 . . . vn} be the vertices in V . Construct a sample S(G)
∈ (Rn × {±1})m, where m = |V | + |E|, as follows:

- For every vi ∈ V , construct an instance ei with a negative label.
- For every edge (vi,vj) ∈ E, construct an instance (ei + ej)/2 with a positive label.

1. Prove that if there exists some h
∈ Hn that has zero error over ( )
then is -colo

⋂kSGGkrable.kHint:Leth=hbenj=1janERMclassifierinHkoverS.DefineacoloringofVbysettingf(vi)tobeth

Use the fact that halfspaces are convex sets to show that it cannot be true that two vertices that are connected by an edge have the same color.

2. Prove that if G is k-colorable then there exists some h
∈ Hnk that has zero error over S(G).

Hint: Given a coloring f of the vertices of G, we should come up with k hyperplanes, h1 . . . hk whose intersection is a perfect classifier for S(G). Let b = 0.6 for all of these hyperplanes and, for t
≤ k let the i'th weight of the t'th hyperplane, wt,i, be
−1 if f(vi) = t and 0 otherwise.

3. Based on the above, prove that for any k
≥3,theERMHnproblemisk
NP-hard.

4. In this exercise we show that hardness of solving the ERM problem is equivalent to hardness of proper PAC learning. Recall that by "properness" of the algorithm we mean that it must output a hypothesis from the hypothesis class. To formalize this statement, we first need the following definition.

definition8.2 ThecomplexityclassRandomizedPolynomial(RP)time is the class of all decision problems (that is, problems in which on any instance one has to find out whether the answer is YES or NO) for which there exists a probabilistic algorithm (namely, the algorithm is allowed to flip random coins while it is running) with these properties:

- Onanyinputinstancethealgorithmrunsinpolynomialtimeintheinput size.
- IfthecorrectanswerisNO,thealgorithmmustreturnNO.
- If the correct answer is YES, the algorithm returns YES with probability a
≥ 1/2 and returns NO with probability 1 − a.[1]

Clearly the class RP contains the class P. It is also known that RP is contained in the class NP. It is not known whether any equality holds among these three complexity classes, but it is widely believed that NP is strictly

[1] The constant 1/2 in the definition can be replaced by any constant in (0, 1).

larger than RP. In particular, it is believed that NP-hard problems cannot be solved by a randomized polynomial time algorithm.

• Show that if a class H is properly PAC learnable by a polynomial time algorithm, then the ERM H problem is in the class RP. In particular, this implies that whenever the ERM H problem is NP-hard (for example, the class of intersections of halfspaces discussed in the previous exercise), then, unless NP = RP, there exists no polynomial time proper PAC learning algorithm for H.

Hint: Assume you have an algorithm A that properly PAC learns a class H in time polynomial in some class parameter n as well as in $1/\epsilon$ and $1/\delta$. Your goal is to use that algorithm as a subroutine to contract an algorithm B for solving the ERM H problem in random polynomial time. Given a training set, $S \in (X \times \{\pm 1\}^m)$, and some $h \in H$ whose error on S is zero, apply the PAC learning algorithm to the uniform distribution over S and run it so that with probability ≥ 0.3 it finds a function $h \in H$ that has error less than $\epsilon = 1/|S|$ (with respect to that uniform distribution). Show that the algorithm just described satisfies the requirements for being a RP solver for ERM H.

Part II

From Theory to Algorithms

9 Linear Predictors

In this chapter we will study the family of linear predictors, one of the most useful families of hypothesis classes. Many learning algorithms that are being widely used in practice rely on linear predictors, first and foremost because of the ability to learn them efficiently in many cases. In addition, linear predictors are intuitive, are easy to interpret, and fit the data reasonably well in many natural learning problems.

We will introduce several hypothesis classes belonging to this family – halfspaces, linear regression predictors, and logistic regression predictors – and present relevant learning algorithms: linear programming and the Perceptron algorithm for the class of halfspaces and the Least Squares algorithm for linear regression. This chapter is focused on learning linear predictors using the ERM approach; however, in later chapters we will see alternative paradigms for learning these hypothesis classes.

First, we define the class of affine functions as

$$L_d = \{h_{w,b} : w \in R^d, b \in R\},$$

where

$$h_{w,b}(x) = \langle w, x \rangle + b = \left(\sum_{i=1}^{d} w_i x_i\right) + b.$$

It will be convenient also to use the notation

$$L = \{x \mapsto \langle w, x \rangle + b : w \in R^d, b \in R\},$$

which reads as follows: L_d is a set of functions, where each function is parameterized by $w \in R^d$ and $b \in R$, and each such function takes as input a vector x and returns as output the scalar $\langle w, x \rangle + b$.

The different hypothesis classes of linear predictors are compositions of a function $\varphi : R \to Y$ on L_d. For example, in binary classification, we can choose φ to be the sign function, and for regression problems, where $Y = R$, φ is simply the identity function.

It may be more convenient to incorporate b, called the bias, into w as an extra coordinate and add an extra coordinate with a value of 1 to all $x \in X$; namely, let $w' = (b, w_1, w_2, \ldots, w_d) \in R^{d+1}$ and let $x' = (1, x_1, x_2, \ldots, x_d) \in X'$.

R^{d+1}. Therefore,

$h_w(x) = \langle w, x \rangle + b = \langle w', x' \rangle$.

It follows that each affine function in Rd can be rewritten as a homogenous linear function in Rd+1 applied over the transformation that appends the constant 1 to each input vector. Therefore, whenever it simplifies the presentation, we will omit the bias term and refer to Ld as the class of homogenous linear functions of the form $h_w(x) = \langle w, x \rangle$.

Throughout the book we often use the general term "linear functions" for both affine functions and (homogenous) linear functions.

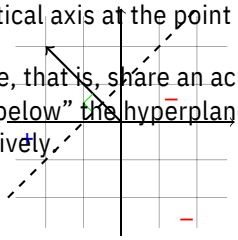
9.1 Halfspaces

The first hypothesis class we consider is the class of halfspaces, designed for binary classification problems, namely, $X = R^d$ and $Y = \{-1, +1\}$. The class of halfspaces is defined as follows:

$HS^d = \text{sign} \circ L^d = \{x \mapsto \text{sign}(h_{w,b}(x)) : h_{w,b} \in L^d\}$.

In other words, each halfspace hypothesis in HS^d is parameterized by $w \in R^d$ and $b \in R$ and upon receiving a vector x the hypothesis returns the label $\text{sign}(\langle w, x \rangle + b)$.

To illustrate this hypothesis class geometrically, it is instructive to consider the case d = 2. Each hypothesis forms a hyperplane that is perpendicular to the vector w and intersects the vertical axis at the point $(0, -b/w_2)$. The instances that are "above" the hyperplane, that is, share an acute angle with w, are labeled positively. Instances that are "below" the hyperplane, that is, share an obtuse angle with w, are labeled negatively.



(In Section 9.1.3 we will show that $VCdim(HS^d) = d + 1$. It follows that we can learn halfspaces using the ERM pa

We introduce below two solutions to finding an ERM halfspace in the realizable case. In the context of halfspaces, the realizable case is often referred to as the "separable" case, since it is possible to separate with a hyperplane all the positive examples from all the negative examples. Implementing the ERM rule

in the nonseparable case (i.e., the agnostic case) is known to be computationally hard (Ben-David & Simon 2001). There are several approaches to learning nonseparable data. The most popular one is to use surrogate loss functions, namely, to learn a halfspace that does not necessarily minimize the empirical risk with the0 – 1 loss, but rather with respect to a diffferent loss function. For example, in Section 9.3 we will describe the logistic regression approach, which can be implemented efficiently even in the nonseparable case. We will study surrogate loss functions in more detail later on in Chapter 12.

9.1.1 Linear Programming for the Class of Halfspaces

Linear programs (LP) are problems that can be expressed as maximizing a linear function subject to linear inequalities. That is,

max
$\in \langle u,w \rangle$ w
subjectto Aw
$\geq$ v
where w
$\in$ Rd is the vector of variables we wish to determine, A is an m × d matrix, and v
$\in$ Rm,u $\in$ Rd are vectors. Linear programs can be solved efficiently,[1] and furthermore, there are publicly available implementations of LP solvers.

We will show that the ERM problem for halfspaces in the realizable case can be expressed as a linear program. For simplicity, we assume the homogenous case. Let S =
{(x)mi,yi}i=1beatrainingsetofsizem.Sinceweassumethe realizable case, an ERM predictor should have zero errors on the training set. That is, we are looking for some vector w
$\in$ Rd for which
sign(
$\langle w,xi \rangle$)=yi, $\forall$ i=1,...,m.
Equivalently, we are looking for some vector w for which

yi
$\langle w,xi \rangle$ >0, $\forall$ i=1,...,m.
Let w $*$ be a vector that satisfies this condition (it must exist since we assume realizability). Define y = mini(y
$\langle w * i,x \rangle$ $\bar{w}$ wi)andlet=y.Therefore,foralli we have
1
yi
$\langle \bar{w}, xi \rangle$ = y $\langle w * i, xi \rangle$ $\geq$1.y
We have thus shown that there exists a vector that satisfies

[1] Namely, in time polynomial in m,d , and in the representation size of real numbers.

yi
$\langle w,xi \rangle \geq 1, \forall i=1,...,m.$ (9.1)

And clearly, such a vector is an ERM predictor.

To find a vector that satisfies Equation (9.1) we can rely on an LP solver as follows. Set A to be the m
× d matrix whose rows are the instances multiplied

by yi. That is, Ai,j = yi x $_{i,j}$, where xi,j is the j'th element of the vector xi. Let v be the vector (1, . . . , 1) $\in$ Rm. Then, Equation (9.1) can be rewritten as

$$\geq \qquad Aw \quad v.$$

The LP form requires a maximization objective, yet all the w that satisfy the constraints are equal candidates as output hypotheses. Thus, we set a "dummy" objective, u = (0, . . . , 0) $\in$ R d .

9.1.2 Perceptron for Halfspaces

A different implementation of the ERM rule is the Perceptron algorithm of Rosenblatt (Rosenblatt 1958). The Perceptron is an iterative algorithm that constructs a sequence of vectors w(1),w(2),.... Initially, w(1) is set to be the all-zeros vector. At iteration t, the Perceptron finds an example i that is mis- labeled by w(t), namely, an example for which sign($\langle w(t),xi \rangle$) 6= yi. Then, the Perceptron updates w(t) by adding to it the instance xi scaled by the label yi. That is, w(t+1) = w(t) + yixi. Recall that our goal is to have yi $\langle w, xi \rangle > 0$ for all i and note that

$$y \langle w(t+1)=ti,xi \rangle \; y \langle w()i+yixi,x \rangle = \langle w(t)iyi,xi \rangle + \|x\|2i.$$

Hence, the update of the Perceptron guides the solution to be "more correct" on the i'th example.

Batch Perceptron

input: A training set $(x1, y1), \ldots, (x m, ym)$
initialize: $w^{(1)} = (0, \ldots, 0)$
for $t = 1, 2, \ldots$
(if $w, xi \leq 0$) then
$w \; w \exists$ is.t.y $\langle (t)i(t+1)$ yxi
else $= (t) + i$
output w(t)

The following theorem guarantees that in the realizable case, the algorithm stops with all sample points correctly classified.

theorem9.1 Assumethat(x1,y1),...,(xm,ym)isseparable,letB=min $\{ \| w \| :$ $\forall i \in [m], yi \langle w,xi \rangle \geq 1\}$,andletR=maxi||xi||.Then,thePerceptronal-gorithm stops after at most (RB)2 iterations, and when it stops it holds that $\forall i \in [m], yi \langle w(t),xi \rangle > 0$.

Proof Bythedefinitionofthestoppingcondition,ifthePerceptronstopsit must have separated all the examples. We will show that if the Perceptron runs for T iterations, then we must have T $\leq$ (RB)2, which implies the Perceptron must stop after at most (RB)2 iterations.

Let w? be a vector that achieves the minimum in the definition of B. That is,

y 〈?i w ,xi〉 ≥ 1 for all i, and among all vectors that satisfy these constraints, w? is of minimal norm.

The idea of the proof is to show that after performing T iterations, the cosine of the angle between w? and w(T+1) is at least T $\sqrt{}$ RB. That is, we will show that

$$\frac{\langle w^*, w(T+1)\rangle}{\|w^*\|\,\|w(T+1)\|} \geq \sqrt{T}\,RB. \quad (9.2)$$

By the Cauchy-Schwartz inequality, the left-hand side of Equation (9.2) is at most 1. Therefore, Equation (9.2) would imply that

$$\sqrt{T} \geq 1 \Rightarrow T \leq (RB)^2,$$

which will conclude our proof.

To show that Equation (9.2) holds, we first show that

$$\langle w^*, w(T+1)\rangle \geq T.$$

Indeed, at the first iteration, w(1) = (0,...,0) and therefore

$$\langle w^*, w(1)\rangle = 0,$$

while on iteration t, if we update using example (xi, yi) we have that

$$\langle w^*, w(t+1)\rangle - \langle w^*, w(t)\rangle = \langle w^*, w(t+1) - w(t)\rangle$$
$$= \langle w^*, y_i x_i\rangle = y_i \langle w^*, x_i\rangle \geq 1.$$

Therefore, after performing T iterations, we get:

$$\sum_{t=1}^{T} \langle w^*, w(t+1)\rangle - \langle w^*, w(t)\rangle \geq T, \quad (9.3)$$

as required.

Next, we upper bound $\|w(t+1)\|^2 = \|w(t) + y_i x_i\|^2$ $\|w(T+1)\|$. For each iteration t we have that

$$\|w(t)\|^2 + 2y_i \langle w(t), x_i\rangle + y_i^2 \|x_i\|^2$$
$$\leq \|w(t)\|^2 + R^2 \quad (9.4)$$

where the last inequality is due to the fact that example i is necessarily such that $y_i \langle w(t), x_i\rangle \leq 0$, and the norm of x_i is at most R. Now, since $\|w(1)\|^2 = 0$, if we use Equation (9.4) recursively for T iterations, we obtain that

$$\|w(T+1)\|^2 \leq TR^2 \Rightarrow \|w(T+1)\| \leq \sqrt{T}R. \quad (9.5)$$

Combining Equation (9.3) with Equation (9.5), and using the fact that $\|w^*\| = B$, we obtain that

$$\frac{\langle w(T+1), w^*\rangle}{\|w^*\|\,\|w(T+1)\|} \geq \frac{T}{B\sqrt{T}R} = \frac{\sqrt{T}}{BR}$$

□

We have thus shown that Equation (9.2) holds, and this concludes our proof.

Remark 9.1 The Perceptron is simple to implement and is guaranteed to converge. However, the convergence rate depends on the parameter B, which in some situations might be exponentially large in d. In such cases, it would be better to implement the ERM problem by solving a linear program, as described in the previous section. Nevertheless, for many natural data sets, the size of B is not too large, and the Perceptron converges quite fast.

9.1.3 The VC Dimension of Halfspaces

To compute the VC dimension of halfspaces, we start with the homogenous case.

theorem 9.2 The VC dimension of the class of homogenous halfspaces in R^d is d.

Proof First, consider the set of vectors $e_1,\ldots,e_d$, where for every i the vector e_i is the all zeros vector except 1 in the i'th coordinate. This set is shattered by the class of homogenous halfspaces. Indeed, for every labeling $y_1,\ldots,y_d$, set $w = (y_1, \ldots, y_d)$, and then $\langle w, e_i \rangle = y_i$ for all i.

Next, let $x_1,\ldots,x_{d+1}$ be a set of $d+1$ vectors in R^d. Then, there must exist real numbers $a_1,\ldots,a_{d+1}$, not all of them are zero, such that $\sum_{i=1} a_i x_i = 0$. Let $I = \{i : a_i > 0\}$ and $J = \{j : a_j < 0\}$. Either I or J is nonempty. Let us first assume that both of them are nonempty. Then,

$$\sum_{i \in I} a_i x_i = \sum_{j \in J} |a_j| x_j .$$

Now, suppose that $x_1, \ldots, x_{d+1}$ are shattered by the class of homogenous classes. Then, there must exist a vector w such that $\langle w, x_i \rangle > 0$ for all $i \in I$ while $\langle w, x_j \rangle < 0$ for every $j \in J$. It follows that

$$\langle \sum_{i \in I} a_i x_i, w \rangle = \sum_{i \in I} a_i \langle x_i, w \rangle = \sum_{j \in J} |a_j| \langle x_j, w \rangle < 0,$$

which leads to a contradiction. Finally, if J (respectively, I) is empty then the right-most (respectively, left-most) inequality should be replaced by an equality, which still leads to a contradiction. □

theorem 9.3 The VC dimension of the class of nonhomogenous halfspaces in R^d is $d+1$.

Proof First, as in the proof of Theorem 9.2, it is easy to verify that the set of vectors $0, e_1, \ldots, e_d$ is shattered by the class of nonhomogenous halfspaces. Second, suppose that the vectors $x_1, \ldots, x_{d+2}$ are shattered by the class of nonhomogenous halfspaces. But, using the reduction we have shown in the beginning of this chapter, it follows that there are $d + 2$ vectors in R^{d+1} that are shattered by the class of homogenous halfspaces. But this contradicts Theorem 9.2. □

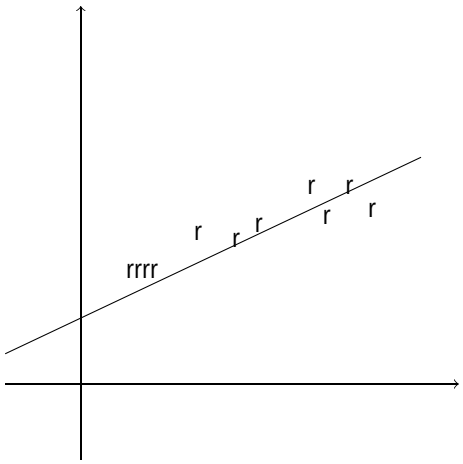


Figure 9.1 Linear regression for d = 1. For instance, the x-axis may denote the age of the baby, and the y-axis her weight.

9.2 Linear Regression

Linear regression is a common statistical tool for modeling the relationship between some "explanatory" variables and some real valued outcome. Cast as a learning problem, the domain set

X is a subset of Rd, for some d, and the label set

Y is the set of real numbers. We would like to learn a linear function h:Rd

→ R that best approximates the relationship between our variables (say, for example, predicting the weight of a baby as a function of her age and weight at birth). Figure 9.1 shows an example of a linear regression predictor for d = 1.

The hypothesis class of linear regression predictors is simply the set of linear functions,

$$H_{reg} = \{x \mapsto \langle w, x \rangle + b : w \in \mathbb{R}^d, b \in \mathbb{R}\}.$$

Next we need to define a loss function for regression. While in classification the definition of the loss is straightforward, as `(h,(x,y)) simply indicates whether h(x) correctly predicts y or not, in regression, if the baby's weight is 3 kg, both the predictions 3.00001 kg and 4 kg are "wrong," but we would clearly prefer the former over the latter. We therefore need to define how much we shall be "penalized" for the discrepancy between h(x) and y. One common way is to use the squared-loss function, namely,

`(h, (x, y)) = (h(x)
− y)2.

For this loss function, the empirical risk function is called the Mean Squared Error, namely,

∑m1LS(h)=(h(xi)−y2i).mi=1

In the next subsection, we will see how to implement the ERM rule for linear regression with respect to the squared loss. Of course, there are a variety of other loss functions that one can use, for example, the absolute value loss function, $\ell(h, (x, y)) = |h(x) - y|$. The ERM rule for the absolute value loss function can be implemented using linear programming (see Exercise 1.)

Note that since linear regression is not a binary prediction task, we cannot analyze its sample complexity using the VC-dimension. One possible analysis of the sample complexity of linear regression is by relying on the "discretization trick" (see Remark 4.1 in Chapter 4); namely, if we are happy with a representation of each element of the vector w and the bias b using a finite number of bits (say a 64 bits floating point representation), then the hypothesis class becomes finite and its size is at most $2^{64(d+1)}$. We can now rely on sample complexity bounds for finite hypothesis classes as described in Chapter 4. Note, however, that to apply the sample complexity bounds from Chapter 4 we also need that the loss function will be bounded. Later in the book we will describe more rigorous means to analyze the sample complexity of regression problems.

9.2.1 Least Squares

Least squares is the algorithm that solves the ERM problem for the hypothesis class of linear regression predictors with respect to the squared loss. The ERM problem with respect to this class, given a training set S, and using the homogenous version of L_d, is to find

$$\operatorname*{argmin}_{w} L_S(h_w) = \operatorname*{argmin}_{w} \frac{1}{m} \sum_{i=1}^{m} (\langle w, x_i \rangle - y_i)^2$$

To solve the problem we calculate the gradient of the objective function and compare it to zero. That is, we need to solve

$$\frac{2}{m} \sum_{i=1}^{m} (\langle w, x_i \rangle - y_i) x_i = 0.$$

We can rewrite the problem as the problem $Aw = b$ where

$$A = \left(\sum_{i=1}^{m} x_i x_i^\top\right) \quad \text{and} \quad b = \sum_{i=1}^{m} y_i x_i \tag{9.6}$$

Or, in matrix form:

$$A = \begin{pmatrix} x_1 \dots x_m & x_1 \dots x_m \end{pmatrix}, \quad (9.7)$$

$$b = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}. \quad (9.8)$$

If A is invertible then the solution to the ERM problem is

$$w = A^{-1} b.$$

The case in which A is not invertible requires a few standard tools from linear algebra, which are available in Appendix C. It can be easily shown that if the training instances do not span the entire space of Rd then A is not invertible. Nevertheless, we can always find a solution to the system Aw = b because b is in the range of A. Indeed, since A is symmetric we can write it using its eigenvalue decomposition as A = V DV >, where D is a diagonal matrix and V is an orthonormal matrix (that is, V >V is the identity d × d matrix). Define D+ to be the diagonal matrix such that D+ i,i = 0 if Di,i = 0 and otherwise D+ i,i = 1/Di,i. Now, define

$$A^+ = VD^+V^\top \text{ and } \hat{w} = A^+ b.$$

Let vi denote the i'th column of V . Then, we have

$$A\hat{w} = AA^+b = VDV^\top VD^+V^\top b = VDD^+V^\top b = VV^\top b = \sum_{i:D_{i,i}\neq 0} v_i v_i^\top b.$$

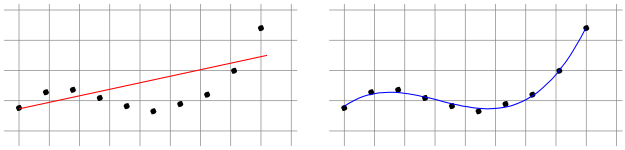
That is, Aŵ is the projection of b onto the span of those vectors vi for which Di,i = 0. Since the linear span of x1,...,xm is the same as the linear span of those vi, and b is in the linear span of the xi, we obtain that Aŵ = b, which concludes our argument.

9.2.2 Linear Regression for Polynomial Regression Tasks

Some learning tasks call for nonlinear predictors, such as polynomial predictors. Take, for instance, a one dimensional polynomial function of degree n, that is,

$$p(x) = a + ax + 2 + \cdots + n0\ 1\ a2x\ anx$$

where (a0,...,an) is a vector of coefficients of size n + 1. In the following we depict a training set that is better fitted using a 3rd degree polynomial predictor than using a linear predictor.



We will focus here on the class of one dimensional, n-degree, polynomial regression predictors, namely,

$$H_n^{poly} = \{x \mapsto p(x)\},$$

where p is a one dimensional polynomial of degree n, parameterized by a vector of coefficients (a0,...,an). Note that

X = R, since this is a one dimensional polynomial, and

Y = R, as this is a regression problem.

One way to learn this class is by reduction to the problem of linear regression, which we have already shown how to solve. To translate a polynomial regression problem to a linear regression problem, we define the mapping $\psi : R \to R^{n+1}$ such that $\psi(x) = (1, x, x^2, \ldots, x^n)$. Then we have that

$$p(\psi(x)) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \langle a, \psi(x) \rangle$$

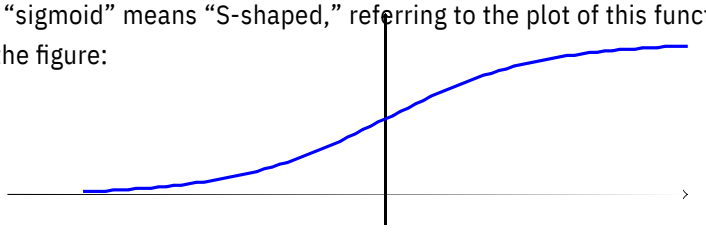
and we can find the optimal vector of coefficients a by using the Least Squares algorithm as shown earlier.

9.3 Logistic Regression

In logistic regression we learn a family of functions h from Rd to the interval [0, 1]. However, logistic regression is used for classification tasks: We can interpret h(x) as the probability that the label of x is 1. The hypothesis class associated with logistic regression is the composition of a sigmoid function $\varphi_{sig} : R \to [0, 1]$ over the class of linear functions Ld. In particular, the sigmoid function used in logistic regression is the logistic function, defined as

$$\varphi_{sig}(z) = \frac{1}{1 + \exp(-z)}. \quad (9.9)$$

The name “sigmoid” means “S-shaped,” referring to the plot of this function, shown in the figure:



The hypothesis class is therefore (where for simplicity we are using homogenous linear functions):

$$H_{sig} = \varphi_{sig} \circ L_d = \{ x \mapsto \varphi_{sig}(\langle w, x \rangle) : w \in R^d \}.$$

Note that when $\langle w,x \rangle$ is very large then $\varphi_{sig}(\langle w,x \rangle)$ is close to 1, whereas if $\langle w, x \rangle$ is very small then $\varphi_{sig}(\langle w, x \rangle)$ is close to 0. Recall that the prediction of the halfspace corresponding to a vector w is $sign(\langle w, x \rangle)$. Therefore, the predictions of the halfspace hypothesis and the logistic hypothesis are very similar whenever $|\langle w,x \rangle|$ is large. However, when $|\langle w,x \rangle|$ is close to 0 we have that $\varphi_{sig}(\langle w,x \rangle) \approx \frac{1}{2}$. Intuitively, the logistic hypothesis is not sure about the value of the label so it guesses that the label is $sign(\langle w,x \rangle)$ with probability slightly larger than 50%. In contrast, the halfspace hypothesis always outputs a deterministic prediction of either 1 or -1, even if $|\langle w,x \rangle|$ is very close to 0.

Next, we need to specify a loss function. That is, we should define how bad it is to predict some $h_w(x) \in [0, 1]$ given that the true label is $y \in \{\pm 1\}$. Clearly, we would like that $h_w(x)$ would be large if $y = 1$ and that $1 - h_w(x)$ (i.e., the probability of predicting -1) would be large if $y = -1$. Note that

$$1 - h_w(x) = 1 - \frac{1}{1+\exp(-\langle w,x \rangle)} = \frac{\exp(-\langle w,x \rangle)}{1+\exp(-\langle w,x \rangle)} = \frac{1}{1+\exp(\langle w,x \rangle)}$$

Therefore, any reasonable loss function would increase monotonically with $\frac{1}{1+\exp(y\langle w,x \rangle)}$, or equivalently, would increase monotonically with $1 + \exp(-y \langle w,x \rangle)$. The logistic loss function used in logistic regression penalizes h_w based on the log of $1 + \exp(-y \langle w, x \rangle)$ (recall that log is a monotonic function). That is,

$$\ell(h_w, (x, y)) = \log (1 + \exp(-y \langle w, x \rangle))$$

The advantage of the logistic loss function is that it is a convex function with respect to w; hence the ERM problem can be solved efficiently using standard methods. We will study how to learn with convex functions, and in particular specify a simple algorithm for minimizing convex functions, in later chapters.

Therefore, given a training set $S = (x_1, y_1), \ldots, (x_m, y_m)$, the ERM problem associated with logistic regression is

$$\operatorname{argmin}_{w \in R^d} \frac{1}{m} \sum_{i=1}^{m} \log(1 + \exp(-y_i \langle w, x_i \rangle)) \quad (9.10)$$

The ERM problem associated with logistic regression (Equation (9.10)) is identical to the problem of finding a Maximum Likelihood Estimator, a well-known statistical approach for finding the parameters that maximize the joint probability of a given data set assuming a specific parametric probability function. We will study the Maximum Likelihood approach in Chapter 24.

9.4 Summary

The family of linear predictors is one of the most useful families of hypothesis classes, and many learning algorithms that are being widely used in practice rely on linear predictors. We have shown efficient algorithms for learning linear predictors with respect to the zero-one loss in the separable case and with respect to the squared and logistic losses in the unrealizable case. In later chapters we will present the properties of the loss function that enable efficient learning.

Naturally, linear predictors are effective whenever we assume, as prior knowledge, that some linear predictor attains low risk with respect to the underlying distribution. In the next chapter we show how to construct nonlinear predictors by composing linear predictors on top of simple classes. This will enable us to employ linear predictors for a variety of prior knowledge assumptions.

9.5 Bibliographic Remarks

The Perceptron algorithm dates back to Rosenblatt (1958). The proof of its convergence rate is due to (Agmon 1954, Novikoff 1962). Least Squares regression goes back to Gauss (1795), Legendre (1805), and Adrain (1808).

9.6 Exercises

1. Show how to cast the ERM problem of linear regression with respect to the absolute value loss function, ` (h,(x,y)) = |h(x) − y|, as a linear program; namely, show how to write the p

∑roblemmmin | ⟨w,xi⟩ −yi|w i=1

as a linear program.
Hint: Start with proving that for any c ∈ R ,

|c|=minas.t.c≤aandc≥−a.a ≥0

2. Show that the matrix A defined in Equation (9.6) is invertible if and only if xxd1,...,mspanR.

3. Show that Theorem 9.1 is tight in the following sense: For any positive integer m, there exist a vector w ∗ ∈ Rd (for some appropriate d) and a sequence of examples {(x1, y1), . . . , (xm, ym)} such that the following hold:

• R=maxi‖xi‖≤1.

• ‖w ∗ ‖2=m,andforalli≤m,y ⟨x w ∗ i i, ⟩ ≥1.Notethat,usingthenotation in Theorem 9.1, we therefore get { ‖ ‖ ∀ ∈ ⟩ ≥ } √— ≤

B=min w : i [m], yi ⟨w,xi 1 m.

Thus, $(BR)^2 \leq m$.

• When running the Perceptron on this sequence of examples it makes m updates before converging.

Hint: Choose d = m and for every i choose xi = ei.

4. (*) Given any number m, find an example of a sequence of labeled examples $((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathbb{R}^3 \times \{-1,+1\})^m$ on which the upper bound of Theorem 9.1 equals m and the perceptron algorithm is bound to make m mistakes.

Hint: Set each xi to be a third dimensional vector of the form (a, b, yi), where $a^2+b^2=R^2 - 1$. Let w^* be the vector (0,0,1). Now, go over the proof of the Perceptron's upper bound (Theorem 9.1), see where we used inequalities ($\leq$) rather than equalities (=), and figure out scenarios where the inequality actually holds with equality.

5. Suppose we modify the Perceptron algorithm as follows: In the update step, instead of performing w(t+1) = w(t) + yixi whenever we make a mistake, we perform $w(t+1) = w(t) + \eta y_i x_i$ for some $\eta > 0$. Prove that the modified Perceptron will perform the same number of iterations as the vanilla Perceptron and will converge to a vector that points to the same direction as the output of the vanilla Perceptron.

6. In this problem, we will get bounds on the VC-dimension of the class of (closed) balls in Rd, that is, $\mathcal{B}^d = \{B_{v,r} : v \in \mathbb{R}, r > 0\}$, where

$$h_{B_{v,r}}(x) = \begin{cases} 1 & \text{if } \|x-v\| \leq r \\ 0 & \text{otherwise} \end{cases}$$

1. Consider the mapping $\varphi : \mathbb{R}^d \to \mathbb{R}^{d+1}$ defined by $\varphi(x) = (x, \|x\|^2)$. Show that if $x_1, \ldots, x_m$ are shattered by $\mathcal{B}^d$ then $\varphi(x_1), \ldots, \varphi(x_m)$ are shattered by the class of halfspaces in Rd+1 (in this question we assume that sign(0) = 1). What does this tell us about $VCdim(\mathcal{B}^d)$?

2. (*) Find a set of d+1 points in Rd that is shattered by $\mathcal{B}^d$. Conclude that $d + 1 \leq VCdim(\mathcal{B}^d) \leq d+2$.

10 Boosting

Boosting is an algorithmic paradigm that grew out of a theoretical question and became a very practical machine learning tool. The boosting approach uses a generalization of linear predictors to address two major issues that have been raised earlier in the book. The first is the bias-complexity tradeoff. We have seen (in Chapter 5) that the error of an ERM learner can be decomposed into a sum of approximation error and estimation error. The more expressive the hypothesis class the learner is searching over, the smaller the approximation error is, but the larger the estimation error becomes. A learner is thus faced with the problem of picking a good tradeoff between these two considerations. The boosting paradigm allows the learner to have smooth control over this tradeoff. The learning starts with a basic class (that might have a large approximation error), and as it progresses the class that the predictor may belong to grows richer.

The second issue that boosting addresses is the computational complexity of learning. As seen in Chapter 8, for many interesting concept classes the task of finding an ERM hypothesis may be computationally infeasible. A boosting algorithm amplifies the accuracy of weak learners. Intuitively, one can think of a weak learner as an algorithm that uses a simple "rule of thumb" to output a hypothesis that comes from an easy-to-learn hypothesis class and performs just slightly better than a random guess. When a weak learner can be implemented efficiently, boosting provides a tool for aggregating such weak hypotheses to approximate gradually good predictors for larger, and harder to learn, classes.

In this chapter we will describe and analyze a practically useful boosting algorithm, AdaBoost (a shorthand for Adaptive Boosting). The AdaBoost algorithm outputs a hypothesis that is a linear combination of simple hypotheses. In other words, AdaBoost relies on the family of hypothesis classes obtained by composing a linear predictor on top of simple classes. We will show that AdaBoost enables us to control the tradeoff between the approximation and estimation errors by varying a single parameter.

AdaBoost demonstrates a general theme, that will recur later in the book, of expanding the expressiveness of linear predictors by composing them on top of other functions. This will be elaborated in Section 10.3.

AdaBoost stemmed from the theoretical question of whether an efficient weak learner can be "boosted" into an efficient strong learner. This question was raised

by Kearns and Valiant in 1988 and solved in 1990 by Robert Schapire, then a graduate student at MIT. However, the proposed mechanism was not very practical. In 1995, Robert Schapire and Yoav Freund proposed the AdaBoost algorithm, which was the first truly practical implementation of boosting. This simple and elegant algorithm became hugely popular, and Freund and Schapire's work has been recognized by numerous awards.

Furthermore, boosting is a great example for the practical impact of learning theory. While boosting originated as a purely theoretical problem, it has led to popular and widely used algorithms. Indeed, as we shall demonstrate later in this chapter, AdaBoost has been successfully used for learning to detect faces in images.

10.1 Weak Learnability

Recall the definition of PAC learning given in Chapter 3: A hypothesis class, H,isPAClearnableifthereexistm 2H:(0,1)→Nandalearningalgorithm with the following property: For every ε, δ ∈ (0, 1), for every distribution D over X, and for every labeling function f : X → {±1}, if the realizable assumption holds with respect to H,D,f, then when running the learning algorithm on m ≥ mH(ε,δ) i.i.d. examples generated by D and labeled by f, the algorithm returns a hypothesis h such that, with probability of at least 1 −δ, L(D,f)(h) ≤ ε.

Furthermore, the fundamental theorem of learning theory (Theorem 6.8 in Chapter 6) characterizes the family of learnable classes and states that every PAC learnable class can be learned using any ERM algorithm. However, the definition of PAC learning and the fundamental theorem of learning theory ignores the computational aspect of learning. Indeed, as we have shown in Chapter 8, there are cases in which implementing the ERM rule is computationally hard (even in the realizable case).

However, perhaps we can trade computational hardness with the requirement for accuracy. Given a distribution D and a target labeling function f, maybe there exists an efficiently computable learning algorithm whose error is just slightly better than a random guess? This motivates the following definition.

definition 10.1 (γ-Weak-Learnability)

- Alearningalgorithm,A,isaγ-weak-learnerforaclassHifthereexistsafunction m H : (0,1) → N such that for every δ ∈ (0,1), for every distribution D over X, and for every labeling function f : X → {±1}, if the realizable assumption holds with respect to H,D,f, then when running the learning algorithm on m ≥ mH(δ) i.i.d. examples generated by D and labeled by f, the algorithm returns a hypothesis h such that, with probability of at least $\frac{1}{2}$ δ, L(D,f)(h) ≤ 1/2 − γ.
- A hypothesis class H is γ-weak-learnable if there exists a γ-weak-learner for that class.

This definition is almost identical to the definition of PAC learning, which here we will call strong learning, with one crucial difference: Strong learnability implies the ability to find an arbitrarily good classifier (with error rate at most ε for an arbitrarily small ε > 0). In weak learnability, however, we only need to output a hypothesis whose error rate is at most 1/2 − γ, namely, whose error rate is slightly better than what a random labeling would give us. The hope is that it may be easier to come up with efficient weak learners than with efficient (full) PAC learners.

The fundamental theorem of learning (Theorem 6.8) states that if a hypothesis class H has a VC dimension d, then the sample complexity of PAC learning H satisfies $m_H(\varepsilon,\delta) \geq C\frac{d+\log(1/\delta)}{\varepsilon}$, where C is a constant. Applying this with $\varepsilon = 1/2 - \gamma$ we immediately obtain that if $d = \infty$ then H is not γ-weak-learnable. This implies that from the statistical perspective (i.e., if we ignore computational complexity), weak learnability is also characterized by the VC dimension of H and therefore is just as hard as PAC (strong) learning. However, when we do consider computational complexity, the potential advantage of weak learning is that maybe there is an algorithm that satisfies the requirements of weak learning and can be implemented efficiently.

One possible approach is to take a "simple" hypothesis class, denoted B, and to apply ERM with respect to B as the weak learning algorithm. For this to work, we need that B will satisfy two requirements:

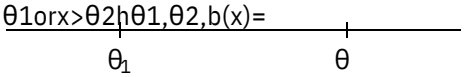
- ERMB is efficiently implementable.
- For every sample that is labeled by some hypothesis from H, any ERMB hypothesis will have an error of at most $1/2 - \gamma$.

Then, the immediate question is whether we can boost an efficient weak learner into an efficient strong learner. In the next section we will show that this is indeed possible, but before that, let us show an example in which efficient weak learnability of a class H is possible using a base hypothesis class B.

Example 10.1 (Weak Learning of 3-Piece Classifiers Using Decision Stumps) Let $X = \mathbb{R}$ and let H be the class of 3-piece classifiers, namely, $H = \{h_{\theta_1,\theta_2,b} : \theta_1, \theta_2 \in \mathbb{R}, \theta_1 < \theta_2, b \in \{\pm 1\}\}$, where

$$h_{\theta_1,\theta_2,b}(x) = \begin{cases} b & \text{if } x < \theta_1 \text{ or } x > \theta_2 \\ -b & \text{if } \theta_1 \leq x \leq \theta_2 \end{cases}$$

An example hypothesis (for b = 1) is illustrated as follows:



Let B be the class of Decision Stumps, that is, $B = \{x \mapsto \text{sign}(x - \theta)\cdot b : \theta \in \mathbb{R}, b \in \{\pm 1\}\}$. In the following we show that ERMB is a γ-weak learner for H, for $\gamma = 1/12$.

To see that, we first show that for every distribution that is consistent with H, there exists a decision stump with $L_D(h) \leq 1/3$. Indeed, just note that every classifier in H consists of three regions (two unbounded rays and a center interval) with alternate labels. For any pair of such regions, there exists a decision stump that agrees with the labeling of these two components. Note that for every distribution D over R and every partitioning of the line into three such regions, one of these regions must have D-weight of at most 1/3. Let $h \in H$ be a zero error hypothesis. A decision stump that disagrees with h only on such a region has an error of at most 1/3.

Finally, since the VC-dimension of decision stumps is 2, if the sample size is greater than $\Omega(\log(1/\delta)/\varepsilon^2)$, then with probability of at least $1 - \delta$, the ERMB rule returns a hypothesis with an error of at most $1/3 + \varepsilon$. Setting $\varepsilon = 1/12$ we obtain that the error of ERMB is at most $1/3 + 1/12 = 1/2 - 1/12$.

We see that ERMB is a γ-weak learner for H. We next show how to implement the ERM rule efficiently for decision stumps.

10.1.1 Efficient Implementation of ERM for Decision Stumps

Let $X = R^d$ and consider the base hypothesis class of decision stumps over R^d, namely,

$$H_{DS} = \{x \mapsto sign(\theta - x_i) \cdot b : \theta \in R, i \in [d], b \in \{\pm 1\}\}.$$

For simplicity, assume that b = 1; that is, we focus on all the hypotheses in H_{DS} of the form $sign(\theta - x_i)$. Let $S = ((x_1, y_1), \ldots, (x_m, y_m))$ be a training set. We will show how to implement an ERM rule, namely, how to find a decision stump that minimizes $L_S(h)$. Furthermore, since in the next section we will show that AdaBoost requires finding a hypothesis with a small risk relative to some distribution over S, we will show here how to minimize such risk functions. Concretely, let D ∑ be a probability vector in R^m (that is, all elements of D are nonnegative and $\sum_i D_i = 1$). The weak learner we describe later receives D and S and outputs a decision stump $h: X \to Y$ that minimizes the risk w.r.t. D,

$$L_D(h) = \sum_{i=1}^{m} D_i 1_{[h(x_i) \neq y_i]}$$

Note that if $D = (1/m, \ldots, 1/m)$ then $L_D(h) = L_S(h)$.

Recall that each decision stump is parameterized by an index $j \in [d]$ and a threshold $\theta \in R$. Therefore, minimizing $L_D(h)$ amounts to solving the problem

$$\min_{j \in [d]} \min_{\theta \in R} \left(\sum_{i : y_i = 1} D_i 1_{[x_{i,j} > \theta]} + \sum_{i : y_i = -1} D_i 1_{[x_{i,j} \leq \theta]} \right) \quad (10.1)$$

Fix $j \in [d]$. We now show how to find $\theta \in R$ that minimizes the above. Sort the examples so that $x_{1,j} \leq \cdots \leq x_{m,j}$. Define $\Theta_j = \{x_{1,j} - 1, \frac{x_{1,j}+x_{2,j}}{2}, \ldots, \frac{x_{m-1,j}+x_{m,j}}{2}, x_{m,j} + 1\}$. Note that for any $\theta \in R$ there exists $\theta' \in \Theta_j$ that yields the same predictions for the sample S as the

threshold θ. Therefore, instead of minimizing over θ ∈ R we can minimize over θ ∈Θj.

This already gives us an efficient procedure: Choose j ∈ [d] and θ ∈ Θj that minimize the objective value of Equation (10.1). For every j and θ ∈ Θj we have to calculate a sum over m examples; therefore the runtime of this approach would be O(dm2). We next show a simple trick that enables us to minimize the objective in time O(dm).

The observation is as follows. Suppose we have calculated the objective for θ ∈ (xi−1,j, xi,j). Let F (θ) be the value of the objective. Then, when we consider θ' ∈ (xi,j, xi+1,j) we have that

F (θ′) = F (θ) −Di1[y i1 −i=1]+D[yi=−1]=F(θ)yiDi.

Therefore, we can calculate the objective at θ′ in a constant time, given the objective at the previous threshold, θ. It follows that after a preprocessing step in which we sort the examples with respect to each coordinate, the minimization problem can be performed in time O(dm). This yields the following pseudocode.

ERM for Decision Stumps

input:

Training set $(x_1, y_1), \ldots, (x_m, y_m)$

distribution vector D

goal: Find $j \in [d]$ and $\theta \in \mathbb{R}$ that solve Equation (10.1)

initialize: $F^\star = \infty$

for $j = 1, \ldots, d$

Sort the training set using the j'th coordinate, and denote $x_{1,j} \leq x_{2,j} \leq \cdots \leq x_{m,j}$

Set $F = \sum_{i:y_i=1} D_i$

if $F < F^\star$ set $F^\star = F$, $j^\star = j$, $\theta^\star = x_{1,j} - 1$

for $i = 1, \ldots, m$

Set $F = F - y_i D_i$

if $F < F^\star$ and $i < m$ and $x_{i,j} \neq x_{i+1,j}$ set $F^\star = F$, $j^\star = j$, $\theta^\star = \frac{1}{2}(x_{i,j} + x_{i+1,j})$

output: $j^\star, \theta^\star$

10.2 AdaBo ost

AdaBoost (short for Adaptive Boosting) is an algorithm that has access to a weak learner and finds a hypothesis with a low empirical risk. The AdaBoost algorithm receives as input a training set of examples S = (x1, y1), . . . , (xm, ym), where for each i, yi = f(xi) for some labeling function f. The boosting process proceeds in a sequence of consecutive rounds. At round t, the booster first defines

a

∑distributionovertheexamplesinS,denotedD(t).Thatis,D(t)∈Rm+andm(t)boosterpassesthedistribution(t)i=1Di=1.
according to D(t) and f.) The weak learner is assumed to return a "weak"
hypothesis, ht, whose error,

6
∑mdefdef(t)εt=LD(t)(ht)=Di1[ht(xi)=y,i]i=1
is at most 1
2−γ (of course, there is a probability of at most δ that the wea

(klearn)erfails).Then,AdaBoostassignsaweightforh11tasfollows:wt=2logε−1.tThatis,theweightofhtisinverselypro
of the round, AdaBoost updates the distribution so that examples on which ht errs will get a higher probability mass while examples on which ht is correct will get a lower probability mass. Intuitively, this will force the weak learner to focus on the problematic examples in the next round. The output of the AdaBoost algorithm is a "strong" classifier that is based on a weighted sum of all the weak hypotheses. The pseudocode of AdaBoost is presented in the following.

AdaBoost

input:
training set S = (x1, y1), . . . , (xm, ym)
weak learner WL
number of rounds T
initialize D(1)=(1 1 m,..., m).
fort=1,...,T:
invoke weakle D(t)

6

(∑arnerh)t=WL(,S)m(t)computeεt=i=1Di1[yi=ht(xi)]letw11t=2logε−1t(t+1)
∑(t)Diexp(−wtyiht(xi))updateDi=,.m(t)j=1Djexp(−wtyj
(h∑forallit(xj)))=1..,mToutputthehypothesishs(x)=sign t=1wtht(x).

The following theorem shows that the training error of the output hypothesis decreases exponentially fast with the number of boosting rounds.

theorem10.2 LetSbeatrainingsetandassumethatateachiterationof AdaBoost, the weak learner returns a hypothesis for which εt ≤ 1/2 − γ. Then, the training error of the output hypothesis of AdaBoost is at most

$$L_S(h_s) = \frac{1}{m}\sum_{i=1}^{m} 1[h_s(x_i) \neq y_i] \leq \exp(-2\gamma^2 T).$$

Proof For each t , denote ∑ft=p≤ t wph.pTherefore, the output of AdaBoost

is f_T. In addition, denote

$$Z_t = \frac{1}{m}\sum_i m\, e^{-y_i f_t(x_i)}.$$

Note that for any hypothesis we have that $1[h(x)\neq y] \leq e^{-yh(x)}$. Therefore, $LS(f_T) \leq Z$, so it suffices to show $Z \leq e^{-2\gamma^2 T}$ that T. To upper bound Z_T we rewrite it as

$$Z_T = \frac{Z_T}{Z_{T-1}} \cdot \frac{Z_{T-1}}{Z_{T-2}} \cdots \frac{Z_2}{Z_1} \cdot \frac{Z_1}{Z_0}, \quad (10.2)$$

where we used the fact that $Z_0 = 1$ because $f_0 = 0$. Therefore, it suffices to show that for every round t,

$$\frac{Z_{t+1}}{Z_t} \leq e^{-2\gamma^2}. \quad (10.3)$$

To do so, we first note that using a simple inductive argument, for all i and t,

$$D_{t+1}(i) = \frac{e^{-y_i f_t(x_i)}}{\sum_{j=1}^m e^{-y_j f_t(x_j)}}.$$

Hence,

$$\frac{Z_{t+1}}{Z_t} = \sum_{i=1}^m \frac{e^{-y_i f_{t+1}(x_i)}}{Z_t} = \sum_{i=1}^m \frac{e^{-y_i f_t(x_i) - y_i w_{t+1} h_{t+1}(x_i)}}{Z_t}$$

$$= \sum_{i=1}^m D_{t+1}(i)\, e^{-y_i w_{t+1} h_{t+1}(x_i)}$$

$$= \sum_{i:y_i = h_{t+1}(x_i)} D_{t+1}(i)\, e^{-w_{t+1}} + \sum_{i:y_i \neq h_{t+1}(x_i)} D_{t+1}(i)\, e^{w_{t+1}}$$

$$= e^{-w_{t+1}}(1-\epsilon_{t+1}) + e^{w_{t+1}}\epsilon_{t+1}$$

$$= \sqrt{\frac{1-\epsilon_{t+1}}{1+\epsilon_{t+1}}}\,(1-\epsilon_{t+1}) + \sqrt{\frac{1+\epsilon_{t+1}}{1-\epsilon_{t+1}}}\,\epsilon_{t+1}$$

$$= 2\epsilon_{t+1}\sqrt{1-\epsilon_{t+1}} + (1-2\epsilon_{t+1})\sqrt{1+\epsilon_{t+1}}$$

$$= 2\epsilon_{t+1}\sqrt{1-\epsilon_{t+1}} + \sqrt{1-\epsilon_{t+1}}\sqrt{1+\epsilon_{t+1}} - 2\epsilon_{t+1}\sqrt{1+\epsilon_{t+1}}$$

$$= \sqrt{1-\epsilon_{t+1}}\left(2\epsilon_{t+1} + \sqrt{1+\epsilon_{t+1}} - 2\epsilon_{t+1}\sqrt{1+\epsilon_{t+1}}\right)$$

$$= \sqrt{1-\epsilon_{t+1}}\left(1 + 2\epsilon_{t+1}\left(1 - \sqrt{1+\epsilon_{t+1}}\right)\right).$$

By our assumption, $\epsilon_{t+1} \leq \frac{1}{2} - \gamma$. Since the function $a(1 - \sqrt{1+a})$ is monotonically increasing in $[0, 1/2]$, we obtain that

$$\frac{Z_{t+1}}{Z_t} \leq \sqrt{1-\left(\frac{1}{2}-\gamma\right)}\left(1 + 2\left(\frac{1}{2}-\gamma\right)\left(1 - \sqrt{1+\left(\frac{1}{2}-\gamma\right)}\right)\right).$$

√Finally,usingtheinequality1−a≤e−awehavethat1−44γ2γ2≤e−/2=−2γ2e.ThisshowsthatEquation(10.3)holdsandt □

Each iteration of AdaBoost involves O(m) operations as well as a single call to the weak learner. Therefore, if the weak learner can be implemented efficiently (as happens in the case of ERM with respect to decision stumps) then the total training process will be efficient.

Remark10.2 Theorem10.2assumesthatateachiterationofAdaBoost,the weak learner returns a hypothesis with weighted sample error of at most 1/2 γ . According to the definition of a weak learner, it can fail with probability δ. Using the union bound, the probability that the weak learner will not fail at all of the iterations is at least 1 − δT. As we show in Exercise 1, the dependence of the sample complexity on δ can always be logarithmic in 1/δ, and therefore invoking the weak learner with a very small δ is not problematic. We can therefore assume that δT is also small. Furthermore, since the weak learner is only applied with distributions over the training set, in many cases we can implement the weak learner so that it will have a zero probability of failure (i.e., δ = 0). This is the case, for example, in the weak learner that finds the minimum value of LD(h) for decision stumps, as described in the previous section.

Theorem 10.2 tells us that the empirical risk of the hypothesis constructed by AdaBoost goes to zero as T grows. However, what we really care about is the true risk of the output hypothesis. To argue about the true risk, we note that the output of AdaBoost is in fact a composition of a halfspace over the predictions of the T weak hypotheses constructed by the weak learner. In the next section we show that if the weak hypotheses come from a base hypothesis class of low VC-dimension, then the estimation error of AdaBoost will be small; namely, the true risk of the output of AdaBoost would not be very far from its empirical risk.

10.3 Linear Combinations of Base Hypotheses

As mentioned previously, a popular approach for constructing a weak learner is to apply the ERM rule with respect to a base hypothesis class (e.g., ERM over decision stumps). We have also seen that boosting outputs a composition of a halfspace over the predictions of the weak hypotheses. Therefore, given a base hypothesis class B (e.g., decision stumps), the output of AdaBoost will be a member of the

{followingc(lass:∑) }TL(B,T)=x→7signwtht(x):w∈RT,∀t,ht∈B.(10.4)t=1

That is, each h ∈ L(B,T) is parameterized by T base hypotheses from B and by a vector w ∈ RT. The prediction of such an h on an instance x is obtained by first applying the T base hypotheses to construct the vector ψ(x) =

(h() ()) ∈RT1x,...,hTx,andthenapplyingthe(homogenous)halfspacedefined by w on ψ(x).

In this section we analyze the estimation error of L(B,T) by bounding the VC-dimension of L(B,T) in terms of the VC-dimension of B and T. We will show that, up to logarithmic factors, the VC-dimension of L(B,T) is bounded by T times the VC-dimension of B. It follows that the estimation error of AdaBoost grows linearly with T . On the other hand, the empirical risk of AdaBoost decreases with T. In fact, as we demonstrate later, T can be used to decrease the approximation error of L(B,T). Therefore, the parameter T of AdaBoost enables us to control the bias-complexity tradeoff.

To demonstrate how the expressive power of L(B, T) increases with T , consider the simple example, in which
X = R and the base class is Decision Stumps,

$$H D S 1 = \{ x \to 7 s i g n (x - \theta) \cdot b : \theta \in R , b \in \{ \pm 1 \} \} .$$

Note that in this one dimensional case, HDS1 is in fact equivalent to (nonhomogenous) halfspaces on R.

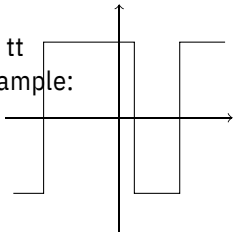
Now, let H be the rather complex class (compared to halfspaces on the line) of piece-wise constant functions. Let gr be a piece-wise constant function with at most r pieces; that is, there exist thresholds $-\infty=\theta 0<\theta 1<\theta 2<\cdots<\theta r=\infty$ such that

$$\sum r g r(x)=\alpha i 1[x\in(\theta,\theta]]\ \forall i\in\{\pm\}i-i,\alpha 1.1ii=1$$

Denote by Gr the class of all such piece-wise constant classifiers with at most r pieces.

In the following we show that GT ⊆ L(HDS1, T); namely, the class of halfspaces over T decision stumps yields all the piece-wise constant classifiers with at most T pieces.

Indeed, without loss of generality consider any g $\in$ G $-$ $-$tTwithαt=(1).This impliesthatifxisintheinterval(θ tt $-$1,θt], then g(x) = (1) . For example:



Now, the function

$$h(x) = \operatorname{sign}\left(\sum_{t=1}^{T} w_t \operatorname{sign}(x - \theta_t)\right) , \qquad (10.5)$$

where w1 = 0.5 and for t > 1, w $-$1)t, is in $_H$L($_{DS1}$, T) and is equal to g (see Exercise 2).

From this example we obtain that L(HDS1,T) can shatter any set of T + 1 instances in R; hence the VC-dimension of L(HDS1, T) is at least T +1. Therefore, T is a parameter that can control the bias-complexity tradeoff: Enlarging T yields a more expressive hypothesis class but on the other hand might increase the estimation error. In the next subsection we formally upper bound the VC-dimension of L(B, T) for any base class B.

10.3.1 The VC-Dimension of L(B, T)

The following lemma tells us that the VC-dimension of L(B, T) is upper bounded byÕ˜(VCdim(B)T)(theÕ˜notationignoresconstantsandlogarithmicfactors).

lemma10.3 LetBbeabaseclassandletL(B,T)beasdefinedinEquation (10.4). Assume that both T and VCdim(B) are at least 3. Then,

VCdim(L(B, T))

≤ T (VCdim(B) + 1) (3 log(T (VCdim(B) + 1)) + 2).

Proof Denote d = VCdim(B). Let C = {x1,...,xm} be a set that is shattered by L(B,T). Each labeling of C by h ∈ L(B, T) is obtained by first choosing h1,...,hT ∈ B and then applying a halfspace hypothesis over the vector (h()()).BySauer d1x,...,hTx'slemma,thereareatmost(em/d)differentdichotomies (i.e., labelings) induced by B over C. Therefore, we need to choose T hypotheses, out of at most (em/d)d different hypotheses. There are at most (em/d)dT ways to do it. Next, for each such choice, we apply a linear predictor, which yields at most (em/T)T dichotomies. Therefore, the overall number of dichotomies we can construct is upper bounded by

(em/d)dT (em/T)T

≤ m(d+1)T ,

where we used the assumption that both d and T are at least 3. Since we assume that C is shattered, we must have that the preceding is at least 2m, which yields

Therefore,

2m ≤(d+1)Tmlog(m).log(2)

≤ m(d+1)T .

Lemma A.1 in Chapter A tells us that a necessary condition for the above to hold is that

≤(d+1)T(d+1)Tm2log≤(d+1)T(3log((d+1)T)+2),log(2)log(2)

which concludes our proof.

In Exercise 4 we show that for some base classes, B, it also holds that VCdim(L(B,T)) ≥ Ω(VCdim(B) T).

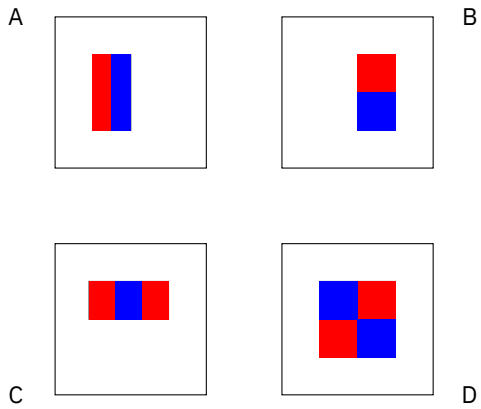


Figure 10.1 The four types of functions, g, used by the base hypotheses for face recognition. The value of g for type A or B is the difference between the sum of the pixels within two rectangular regions. These regions have the same size and shape and are horizontally or vertically adjacent. For type C, the value of g is the sum within two outside rectangles subtracted from the sum in a center rectangle. For type D, we compute the difference between diagonal pairs of rectangles.

10.4 AdaBoost for Face Recognition

We now turn to a base hypothesis that has been proposed by Viola and Jones for the task of face recognition. In this task, the instance space is images, represented as matrices of gray level values of pixels. To be concrete, let us take images of size 24 × 24 pixels, and therefore our instance space is the set of real valued matrices of size 24 × 24. The goal is to learn a classifier, $h : X \to \{\pm 1\}$, that given an image as input, should output whether the image is of a human face or not.

Each hypothesis in the base class is of the form $h(x) = f(g(x))$, where f is a decision stump hypothesis and $g : R^{24,24} \to R$ is a function that maps an image to a scalar. Each function g is parameterized by

- An axis aligned rectangle R. Since each image is of size 24 × 24, there are at most 244 axis aligned rectangles.
- A type, $t \in \{A,B,C,D\}$. Each type corresponds to a mask, as depicted in Figure 10.1.

To calculate g we stretch the mask t to fit the rectangle R and then calculate the sum of the pixels (that is, sum of their gray level values) that lie within the red rectangles and subtract it from the sum of pixels in the blue rectangles. Since the number of such functions g is at most 244 · 4, we can implement a weak learner for the base hypothesis class by first calculating all the possible outputs of g on each image, and then apply the weak learner of decision stumps described in the previous subsection. It is possible to perform the first step very

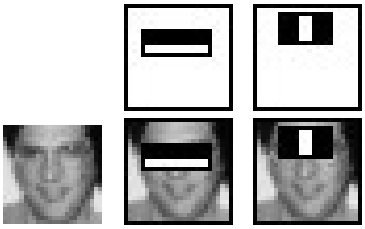


Figure 10.2 The first and second features selected by AdaBoost, as implemented by Viola and Jones. The two features are shown in the top row and then overlaid on a typical training face in the bottom row. The first feature measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalizes on the observation that the eye region is often darker than the cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

efficiently by a preprocessing step in which we calculate the integral image of each image in the training set. See Exercise 5 for details.

In Figure 10.2 we depict the first two features selected by AdaBoost when running it with the base features proposed by Viola and Jones.

10.5 Summary

Boosting is a method for amplifying the accuracy of weak learners. In this chapter we described the AdaBoost algorithm. We have shown that after T iterations of AdaBoost, it returns a hypothesis from the class L(B, T), obtained by composing a linear classifier on T hypotheses from a base class B. We have demonstrated how the parameter T controls the tradeoff between approximation and estimation errors. In the next chapter we will study how to tune parameters such as T , based on the data.

10.6 Bibliographic Remarks

As mentioned before, boosting stemmed from the theoretical question of whether an efficient weak learner can be "boosted" into an efficient strong learner (Kearns & Valiant 1988) and solved by Schapire (1990). The AdaBoost algorithm has been proposed in Freund & Schapire (1995).

Boosting can be viewed from many perspectives. In the purely theoretical context, AdaBoost can be interpreted as a negative result: If strong learning of a hypothesis class is computationally hard, so is weak learning of this class. This negative result can be useful for showing hardness of agnostic PAC learning of a class B based on hardness of PAC learning of some other class H, as long as

H is weakly learnable using B. For example, Klivans & Sherstov (2006) have shown that PAC learning of the class of intersection of halfspaces is hard (even in the realizable case). This hardness result can be used to show that agnostic PAC learning of a single halfspace is also computationally hard (Shalev-Shwartz, Shamir & Sridharan 2010). The idea is to show that an agnostic PAC learner for a single halfspace can yield a weak learner for the class of intersection of halfspaces, and since such a weak learner can be boosted, we will obtain a strong learner for the class of intersection of halfspaces.

AdaBoost also shows an equivalence between the existence of a weak learner and separability of the data using a linear classifier over the predictions of base hypotheses. This result is closely related to von Neumann's minimax theorem (von Neumann 1928), a fundamental result in game theory.

AdaBoost is also related to the concept of margin, which we will study later on in Chapter 15. It can also be viewed as a forward greedy selection algorithm, a topic that will be presented in Chapter 25. A recent book by Schapire & Freund (2012) covers boosting from all points of view, and gives easy access to the wealth of research that this field has produced.

10.7 Exercises

1. Boosting the Confidence: Let A be an algorithm that guarantees the following: There exist some constant $\delta_0 \in (0, 1)$ and a function $m_H : (0, 1) \to \mathbb{N}$ such that for every $\epsilon \in (0,1)$, if $m \geq m_H(\epsilon)$ then for every distribution D it holds that with probability of at least $1 - \delta_0$, $L_D(A(S)) \leq \min_{h \in H} L_D(h) + \epsilon$. Suggest a procedure that relies on A and learns H in the usual agnostic PAC learning model and has a sample complexity of $m_H(\epsilon, \delta) \leq k m_H(\epsilon) + \frac{k/\delta}{\epsilon}$, where $k = \lceil 2\log(4/\delta)/\log(1/\delta_0) \rceil$.

Hint: Divide the data into k + 1 chunks, where each of the first k chunks is of size $m_H(\epsilon)$ examples. Train the first k chunks using A. Argue that the probability that for all of these chunks we have $L_D(A(S)) > \min_{h \in H} L_D(h) + \epsilon$ is at most $\delta_0^k \leq \delta/2$. Finally, use the last chunk to choose from the k hypotheses that A generated from the k chunks (by relying on Corollary 4.6).

2. Prove that the function h given in Equation (10.5) equals the piece-wise constant function defined according to the same thresholds as h.

3. We have informally argued that the AdaBoost algorithm uses the weighting mechanism to "force" the weak learner to focus on the problematic examples in the next iteration. In this question we will find some rigorous justification for this argument.

Show that the error of h_t w.r.t. the distribution $D^{(t+1)}$ is exactly 1/2. That is, show that for every $t \in [T]$

$$\sum_{i=1}^{m} D_i^{(t+1)} \mathbb{1}[y_i = h_t(x_i)] = \frac{1}{2}$$

4. In this exercise we discuss the VC-dimension of classes of the form L(B,T). We proved an upper bound of O(dT log(dT)), where d = VCdim(B). Here we wish to prove an almost matching lower bound. However, that will not be the case for all classes B.

1. Note that for every class B and every number $T \geq 1$, $\text{VCdim}(B) \leq \text{VCdim}(L(B, T))$. Find a class B for which $\text{VCdim}(B) = \text{VCdim}(L(B, T))$ for every $T \geq 1$.
Hint: Take X to be a finite set.

2. Let B be the class of decision stumps over R^d. Prove that $\log(d) \leq \text{VCdim}(B_d) \leq 5 + 2\log(d)$.
Hints:
- For the upper bound, rely on Exercise 11.
- For the lower bound, assume $d = 2k$. Let A be a $k \times d$ matrix whose columns are all the d binary vectors in $\{\pm 1\}^k$. The rows of A form a set of k vectors in R^d. Show that this set is shattered by decision stumps over R^d.

3. Let $T \geq 1$ be any integer. Prove that $\text{VCdim}(L(B_d, T)) \geq 0.5\, T \log(d)$.
Hint: Construct a set of T 2^k instances by taking the rows of the matrix A from the previous question, and the rows of the matrices $2A, 3A, 4A, \ldots, TA$. Show that the resulting set is shattered by $L(B_d, T)$.

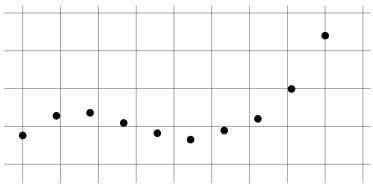
5. Efficiently Calculating the Viola and Jones Features Using an Integral Image: Let A be a 24×24 matrix representing an image.

- The integral image of A, denoted by I(A), is the matrix B such that $B_{i,j} = \sum_{i' \leq i, j' \leq j} A_{i,j}$.
- Show that I(A) can be calculated from A
- Show how every Viola and Jones feature can be calculated from I(A) in a constant amount of time (that is, the runtime does not depend on the size of the rectangle defining the feature).

11 Model Selection and Validation

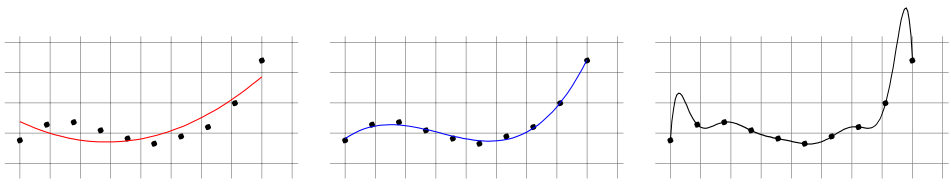
In the previous chapter we have described the AdaBoost algorithm and have shown how the parameter T of AdaBoost controls the bias-complexity trade- off. But, how do we set T in practice? More generally, when approaching some practical problem, we usually can think of several algorithms that may yield a good solution, each of which might have several parameters. How can we choose the best algorithm for the particular problem at hand? And how do we set the algorithm's parameters? This task is often called model selection.

To illustrate the model selection task, consider the problem of learning a one dimensional regression function, $h : R \to R$. Suppose that we obtain a training set as depicted in the figure.



We can consider fitting a polynomial to the data, as described in Chapter 9. However, we might be uncertain regarding which degree d would give the best results for our data set: A small degree may not fit the data well (i.e., it will have a large approximation error), whereas a high degree may lead to overfitting (i.e., it will have a large estimation error). In the following we depict the result of fitting a polynomial of degrees 2, 3, and 10. It is easy to see that the empirical risk decreases as we enlarge the degree. However, looking at the graphs, our intuition tells us that setting the degree to 3 may be better than setting it to 10. It follows that the empirical risk alone is not enough for model selection.

degree 2 degree 3 degree 10



In this chapter we will present two approaches for model selection. The first approach is based on the Structural Risk Minimization (SRM) paradigm we have described and analyzed in Chapter 7.2. SRM is particularly useful when a learning algorithm depends on a parameter that controls the bias-complexity tradeoff (such as the degree of the fitted polynomial in the preceding example or the parameter T in AdaBoost). The second approach relies on the concept of validation. The basic idea is to partition the training set into two sets. One is used for training each of the candidate models, and the second is used for deciding which of them yields the best results.

In model selection tasks, we try to find the right balance between approximation and estimation errors. More generally, if our learning algorithm fails to find a predictor with a small risk, it is important to understand whether we suffer from overfitting or underfitting. In Section 11.3 we discuss how this can be achieved.

11.1 Model Selection Using SRM

The SRM paradigm has been described and analyzed in Section 7.2. Here we show how SRM can be used for tuning the tradeoff between bias and complexity without deciding on a specific hypothesis class in advance. Consider a countable sequence of hypothesis classes $H_1, H_2, H_3, \ldots$. For example, in the problem of polynomial regression mentioned, we can take H_d to be the set of polynomials of degree at most d. Another example is taking H_d to be the class L(B, d) used by AdaBoost, as described in the previous chapter.

We assume that for every d, the class H_d enjoys the uniform convergence property (see Definition 4.3 in Chapter 4) with a sample complexity function of the form

$$m_{UC}^{H}(\epsilon,\delta) \le \frac{g(d) + \log(1/\delta)}{\epsilon^2}, \quad (11.1)$$

where $g : N \to R$ is some monotonically increasing function. For example, in the case of binary classification problems, we can take g(d) to be the VC-dimension of the class H_d multiplied by a universal constant (the one appearing in the fundamental theorem of learning; see Theorem 6.8). For the classes L(B, d) used by AdaBoost, the function g will simply grow with d.

Recall that the SRM rule follows a "bound minimization" approach, where in our case the bound is as follows: With probability of at least $1 - \delta$, for every $d \in N$ and $h \in H_d$,

$$L_D(h) \le L_S(h) + \sqrt{\frac{g(d)(\log(1/\delta) + 2\log(d) + \log(\pi^2/6))}{m}}. \quad (11.2)$$

This bound, which follows directly from Theorem 7.4, shows that for every d and every $h \in H_d$, the true risk is bounded by two terms – the empirical risk, $L_S(h)$,

and a complexity term that depends on d. The SRM rule will search for d and $h \in H_d$ that minimize the right-hand side of Equation (11.2).

Getting back to the example of polynomial regression described earlier, even though the empirical risk of the 10th degree polynomial is smaller than that of the 3rd degree polynomial, we would still prefer the 3rd degree polynomial since its complexity (as reflected by the value of the function g(d)) is much smaller.

While the SRM approach can be useful in some situations, in many practical cases the upper bound given in Equation (11.2) is pessimistic. In the next section we present a more practical approach.

11.2 Validation

We would often like to get a better estimation of the true risk of the output predictor of a learning algorithm. So far we have derived bounds on the estimation error of a hypothesis class, which tell us that for all hypotheses in the class, the true risk is not very far from the empirical risk. However, these bounds might be loose and pessimistic, as they hold for all hypotheses and all possible data distributions. A more accurate estimation of the true risk can be obtained by using some of the training data as a validation set, over which one can evalutate the success of the algorithm's output predictor. This procedure is called validation.

Naturally, a better estimation of the true risk is useful for model selection, as we will describe in Section 11.2.2.

11.2.1 Hold Out Set

The simplest way to estimate the true error of a predictor h is by sampling an additional set of examples, independent of the training set, and using the empirical error on this validation set as our estimator. Formally, let $V = (x_1, y_1), \ldots, (x_{m_v}, y_{m_v})$ be a set of fresh m_v examples that are sampled according to D (independently of the m examples of the training set S). Using Hoeffding's inequality (Lemma 4.5) we have the following:

theorem 11.1 Let h be some predictor and assume that the loss function is in $[0,1]$. Then, for every $\delta \in (0,1)$, with probability of at least $1-\delta$ over the choice of a validation set V of size m_v we have

$$|L_V(h) - L_D(h)| \leq \sqrt{\frac{\log(2/\delta)}{2m_v}}.$$

The bound in Theorem 11.1 does not depend on the algorithm or the training set used to construct h and is tighter than the usual bounds that we have seen so far. The reason for the tightness of this bound is that it is in terms of an estimate on a fresh validation set that is independent of the way h was generated. To illustrate this point, suppose that h was obtained by applying an ERM predictor

with respect to a hypothesis class of VC-dimension d, over a training set of m examples. Then, from the fundamental theorem of learning (Theorem 6.8) we obtain the bound

$$L_D(h) \leq L_S(h) + C\sqrt{\frac{d + \log(1/\delta)}{m}}$$

where C is the constant appearing in Theorem 6.8. In contrast, from Theorem 11.1 we obtain the bound

$$L_D(h) \leq L_V(h) + \sqrt{\frac{\log(2/\delta)}{2m_v}}$$

Therefore, taking m_v to be order of m, we obtain an estimate that is more accurate by a factor that depends on the VC-dimension. On the other hand, the price we pay for using such an estimate is that it requires an additional sample on top of the sample used for training the learner.

Sampling a training set and then sampling an independent validation set is equivalent to randomly partitioning our random set of examples into two parts, using one part for training and the other one for validation. For this reason, the validation set is often referred to as a hold out set.

11.2.2 Validation for Model Selection

Validation can be naturally used for model selection as follows. We first train different algorithms (or the same algorithm with different parameters) on the given training set. Let $H = \{h_1, \ldots, h_r\}$ be the set of all output predictors of the different algorithms. For example, in the case of training polynomial regressors, we would have each h_r be the output of polynomial regression of degree r. Now, to choose a single predictor from H we sample a fresh validation set and choose the predictor that minimizes the error over the validation set. In other words, we apply ERM H over the validation set.

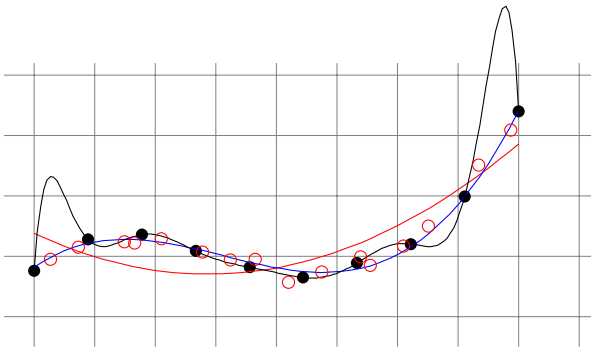
This process is very similar to learning a finite hypothesis class. The only difference is that H is not fixed ahead of time but rather depends on the training set. However, since the validation set is independent of the training set we get that it is also independent of H and therefore the same technique we used to derive bounds for finite hypothesis classes holds here as well. In particular, combining Theorem 11.1 with the union bound we obtain:

Let $H = \{h_1, \ldots, h_r\}$ be an arbitrary set of predictors and assume that the loss function is in $[0,1]$. Assume that a validation set of size m_v is sampled independent of H. Then, with probability of at least $1-\delta$ over the choice of we have

$$\forall h \in H, \quad |L_V(h) - L_D(h)| \leq \sqrt{\frac{\log(2|H|/\delta)}{2m_v}}.$$

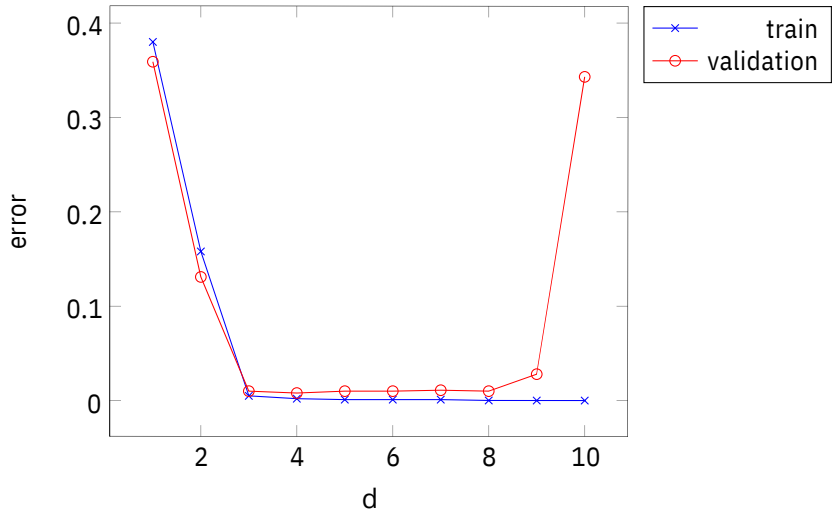
This theorem tells us that the error on the validation set approximates the true error as long as H is not too large. However, if we try too many methods (resulting in |H| that is large relative to the size of the validation set) then we're in danger of overfitting.

To illustrate how validation is useful for model selection, consider again the example of fitting a one dimensional polynomial as described in the beginning of this chapter. In the following we depict the same training set, with ERM polynomials of degree 2, 3, and 10, but this time we also depict an additional validation set (marked as red, unfilled circles). The polynomial of degree 10 has minimal training error, yet the polynomial of degree 3 has the minimal validation error, and hence it will be chosen as the best model.



11.2.3 The Model-Selection Curve

The model selection curve shows the training error and validation error as a function of the complexity of the model considered. For example, for the polynomial fitting problem mentioned previously, the curve will look like:



As can be shown, the training error is monotonically decreasing as we increase the polynomial degree (which is the complexity of the model in our case). On the other hand, the validation error first decreases but then starts to increase, which indicates that we are starting to suffer from overfitting.

Plotting such curves can help us understand whether we are searching the correct regime of our parameter space. Often, there may be more than a single parameter to tune, and the possible number of values each parameter can take might be quite large. For example, in Chapter 13 we describe the concept of regularization, in which the parameter of the learning algorithm is a real number. In such cases, we start with a rough grid of values for the parameter(s) and plot the corresponding model-selection curve. On the basis of the curve we will zoom in to the correct regime and employ a finer grid to search over. It is important to verify that we are in the relevant regime. For example, in the polynomial fitting problem described, if we start searching degrees from the set of values $\{1, 10, 20\}$ and do not employ a finer grid based on the resulting curve, we will end up with a rather poor model.

11.2.4 k-Fold Cross Validation

The validation procedure described so far assumes that data is plentiful and that we have the ability to sample a fresh validation set. But in some applications, data is scarce and we do not want to "waste" data on validation. The k-fold cross validation technique is designed to give an accurate estimate of the true error without wasting too much data.

In k-fold cross validation the original training set is partitioned into k subsets (folds) of size m/k (for simplicity, assume that m/k is an integer). For each fold, the algorithm is trained on the union of the other folds and then the error of its output is estimated using the fold. Finally, the average of all these errors is the

estimate of the true error. The special case k = m, where m is the number of examples, is called leave-one-out (LOO).

k-Fold cross validation is often used for model selection (or parameter tuning), and once the best parameter is chosen, the algorithm is retrained using this parameter on the entire training set. A pseudocode of k-fold cross validation for model selection is given in the following. The procedure receives as input a training set, S, a set of possible parameter values, Θ, an integer, k, representing the number of folds, and a learning algorithm, A, which receives as input a training set as well as a parameter θ ∈ Θ. It outputs the best parameter as well as the hypothesis trained by this parameter on the entire training set.

k-Fold Cross Validation for Model Selection

input:
training set $S = (x_1, y_1), \ldots, (x_m, y_m)$
set of parameter values Θ
learning algorithm A
integer k
partition S into $S_1, S_2, \ldots, S_k$
foreach $\theta \in \Theta$
for $i = 1 \ldots k$
$h_{i,\theta} = A(S \setminus S_i; \theta)$
$error(\theta) = \frac{1}{k} \sum_{i=1}^{k} L_{S_i}(h_{i,\theta})$
output
$\theta^? = \operatorname{argmin}_\theta [error(\theta)]$
$h_{?\theta?} = A(S; \theta)$

The cross validation method often works very well in practice. However, it might sometime fail, as the artificial example given in Exercise 1 shows. Rigorously understanding the exact behavior of cross validation is still an open problem. Rogers and Wagner (Rogers & Wagner 1978) have shown that for k local rules (e.g., k Nearest Neighbor; see Chapter 19) the cross validation procedure gives a very good estimate of the true error. Other papers show that cross validation works for stable algorithms (we will study stability and its relation to learnability in Chapter 13).

11.2.5 Train-Validation-TestSplit

In most practical applications, we split the available examples into three sets. The first set is used for training our algorithm and the second is used as a validation set for model selection. After we select the best model, we test the performance of the output predictor on the third set, which is often called the "test set." The number obtained is used as an estimator of the true error of the learned predictor.

11.3 What to Do If Learning Fails

Consider the following scenario: You were given a learning task and have approached it with a choice of a hypothesis class, a learning algorithm, and parameters. You used a validation set to tune the parameters and tested the learned predictor on a test set. The test results, unfortunately, turn out to be unsatisfactory. What went wrong then, and what should you do next?

There are many elements that can be "fixed." The main approaches are listed in the following:

- Getalargersample
- Change the hypothesis class by:
  - Enlarging it
  - Reducing it
  - Completely changing it
  - Changing the parameters you consider
- Change the feature representation of the data
- Change the optimization algorithm used to apply your learning rule

In order to find the best remedy, it is essential first to understand the cause of the bad performance. Recall that in Chapter 5 we decomposed the true error of the learned predictor into approximation error and estimation error. The approximationerrorisdefinedtobeL? ? D(h) for some h ∈ argminh∈HLD(h), whiletheestimationerrorisdefinedtobeL ? D(hS) – LD(h), where hS is the learned predictor (which is based on the training set S).

The approximation error of the class does not depend on the sample size or on the algorithm being used. It only depends on the distribution D and on the hypothesis class H. Therefore, if the approximation error is large, it will not help us to enlarge the training set size, and it also does not make sense to reduce the hypothesis class. What can be beneficial in this case is to enlarge the hypothesis class or completely change it (if we have some alternative prior knowledge in the form of a different hypothesis class). We can also consider applying the same hypothesis class but on a different feature representation of the data (see Chapter 25).

The estimation error of the class does depend on the sample size. Therefore, if we have a large estimation error we can make an effort to obtain more training examples. We can also consider reducing the hypothesis class. However, it doesn't make sense to enlarge the hypothesis class in that case.

Error Decomposition Using Validation

We see that understanding whether our problem is due to approximation error or estimation error is very useful for finding the best remedy. In the previous section we saw how to estimate L D(hS) using the empirical risk on a validation set. However, it is more difficult to estimate the approximation error of the class.

Instead, we give a different error decomposition, one that can be estimated from the train and validation sets.

L
D(hS) = (LD(hS) − LV (hS)) + (LV (hS) − LS(hS)) + LS(hS).
The first term, (L
D(hS) − LV (hS)), can be bounded quite tightly using Theorem 11.1. Intuitively, when the second term, (LV (hS)
− LS(hS)), is large we say that our algorithm suffers from "overfitting" while when the empirical risk term, LS(hS), is large we say that our algorithm suffers from "underfitting." Note that these two terms are not necessarily good estimates of the estimation and approximation errors. To illustrate this, consider the case in which
H is a class of VC-dimension d, and
D is a distribution such that the approximation error of H with respect to
D is 1/4. As long as the size of our training set is smaller than d we will have LS(hS) = 0 for every ERM hypothesis. Therefore, the training risk,L(htheapproximatione?SS),andrror,L
D(h), can be significantly different. Nevertheless, as we show later, the values of LS(hS) and (LV (hS)
− LS (hS)) still provide us useful information.
Consider first the case in which LS(hS) is large. We can write

L(h)=(()
−(?)?−??SSLShSLSh)+(LS(h)LD(h))+LD(h).
Whenh isanERM hypothesisweh
−?SHavethatLS(hS)LS(h)≤0.Inaddition, since h? does not depend on S, the term (L ?
− ?S(h)LD(h))canbeboundedquite tightly (as in Theorem 11.1). The last term is the approximation error. It follows that if LS(hS) is large then so is the approximation error, and the remedy to the failure of our algorithm should be tailored accordingly (as discussed previously).

Remark 11.1 It is possible that the approximation error of our class is small, yet the value of LS(hS) is large. For example, maybe we had a bug in our ERM implementation, and the algorithm returns a hypothesis hS that is not an ERM. It may also be the case that finding an ERM hypothesis is computationally hard, and our algorithm applies some heuristic trying to find an approximate ERM. In some cases, it is hard to know how good hS is relative to an ERM hypothesis. But, sometimes it is possible at least to know whether there are better hypotheses. For example, in the next chapter we will study convex learning problems in which there are optimality conditions that can be checked to verify whether our optimization algorithm converged to an ERM solution. In other cases, the solution may depend on randomness in initializing the algorithm, so we can try different randomly selected initial points to see whether better solutions pop out.

Next consider the case in which LS(hS) is small. As we argued before, this does not necessarily imply that the approximation error is small. Indeed, consider
two scenarios, in both of which we are trying to learn a hypothesis class of VC-dimension d using the ERM learning rule. In the first scenario, we have a training set of m < d examples and the approximation error of the class is high. In the second scenario, we have a training set of m > 2d examples and the

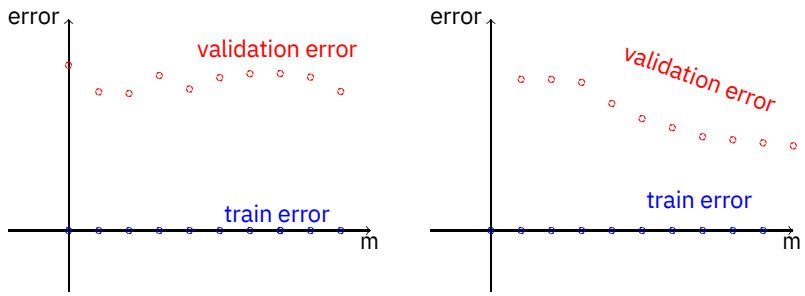


Figure 11.1 Examples of learning curves. Left: This learning curve corresponds to the scenario in which the number of examples is always smaller than the VC dimension of the class. Right: This learning curve corresponds to the scenario in which the approximation error is zero and the number of examples is larger than the VC dimension of the class.

approximation error of the class is zero. In both cases $LS(hS) = 0$. How can we distinguish between the two cases?

Learning Curves

One possible way to distinguish between the two cases is by plotting learning curves. To produce a learning curve we train the algorithm on prefixes of the data of increasing sizes. For example, we can first train the algorithm on the first 10% of the examples, then on 20% of them, and so on. For each prefix we calculate the training error (on the prefix the algorithm is being trained on) and the validation error (on a predefined validation set). Such learning curves can help us distinguish between the two aforementioned scenarios. In the first scenario we expect the validation error to be approximately 1/2 for all prefixes, as we didn't really learn anything. In the second scenario the validation error will start as a constant but then should start decreasing (it must start decreasing once the training set size is larger than the VC-dimension). An illustration of the two cases is given in Figure 11.1.

In general, as long as the approximation error is greater than zero we expect the training error to grow with the sample size, as a larger amount of data points makes it harder to provide an explanation for all of them. On the other hand, the validation error tends to decrease with the increase in sample size. If the VC-dimension is finite, when the sample size goes to infinity, the validation and train errors converge to the approximation error. Therefore, by extrapolating the training and validation curves we can try to guess the value of the approximation error, or at least to get a rough estimate on an interval in which the approximation error resides.

Getting back to the problem of finding the best remedy for the failure of our algorithm, if we observe that $LS(hS)$ is small while the validation error is large, then in any case we know that the size of our training set is not sufficient for learning the class

H. We can then plot a learning curve. If we see that the

validation error is starting to decrease then the best solution is to increase the number of examples (if we can afford to enlarge the data). Another reasonable solution is to decrease the complexity of the hypothesis class. On the other hand, if we see that the validation error is kept around 1/2 then we have no evidence that the approximation error of H is good. It may be the case that increasing the training set size will not help us at all. Obtaining more data can still help us, as at some point we can see whether the validation error starts to decrease or whether the training error starts to increase. But, if more data is expensive, it may be better first to try to reduce the complexity of the hypothesis class. To summarize the discussion, the following steps should be applied:

1. If learning involves parameter tuning, plot the model-selection curve to make sure that you tuned the parameters appropriately (see Section 11.2.3).
2. Ifthetrainingerrorisexcessivelylargeconsiderenlargingthehypothesisclass, completely change it, or change the feature representation of the data.
3. If the training error is small, plot learning curves and try to deduce from them whether the problem is estimation error or approximation error.
4. If the approximation error seems to be small enough, try to obtain more data. If this is not possible, consider reducing the complexity of the hypothesis class.
5. If the approximation error seems to be large as well, try to change the hypothesis class or the feature representation of the data completely.

11.4 Summary

Model selection is the task of selecting an appropriate model for the learning task based on the data itself. We have shown how this can be done using the SRM learning paradigm or using the more practical approach of validation. If our learning algorithm fails, a decomposition of the algorithm's error should be performed using learning curves, so as to find the best remedy.

11.5 Exercises

1. Failure of k-fold cross validation Consider a case in that the label is chosen at random according to P[y = 1] = P[y = 0] = 1/2. Consider a learning algorithm that outputs the constant predictor h(x) = 1 if the parity of the labels on the training set is 1 and otherwise the algorithm outputs the constant predictor h(x) = 0. Prove that the difference between the leave-one-out estimate and the true error in such a case is always 1/2.
2. Let H1, . . . , Hk be k hypothesis classes. Suppose you are given m i.i.d. training examples and you would like to learn the class H = ∪ki=1Hi. Consider two alternative approaches:
• Learn H on the m examples using the ERM rule

• Dividethemexamplesintoatrainingsetofsize(1−α)mandavalidation set of size αm, for some α ∈ (0, 1). Then, apply the approach of model selection using validation. That is, first train each class Hi on the (1 − α)m training examples using the ERM rule with respect to Hi, and let ĥ^1,...,ĥ^kbetheresultinghypotheses.Second,applytheERMrulewith respect to the finite class {ĥ^1,...,ĥ^k}ontheαmvalidationexamples. Describe scenarios in which the first method is better than the second and vice versa.

12 Convex Learning Problems

In this chapter we introduce convex learning problems. Convex learning comprises an important family of learning problems, mainly because most of what we can learn efficiently falls into it. We have already encountered linear regression with the squared loss and logistic regression, which are convex problems, and indeed they can be learned efficiently. We have also seen nonconvex problems, such as halfspaces with the 0-1 loss, which is known to be computationally hard to learn in the unrealizable case.

In general, a convex learning problem is a problem whose hypothesis class is a convex set, and whose loss function is a convex function for each example. We be- gin the chapter with some required definitions of convexity. Besides convexity, we will define Lipschitzness and smoothness, which are additional properties of the loss function that facilitate successful learning. We next turn to defining convex learning problems and demonstrate the necessity for further constraints such as Boundedness and Lipschitzness or Smoothness. We define these more restricted families of learning problems and claim that Convex-Smooth/Lipschitz-Bounded problems are learnable. These claims will be proven in the next two chapters, in which we will present two learning paradigms that successfully learn all problems that are either convex-Lipschitz-bounded or convex-smooth-bounded.

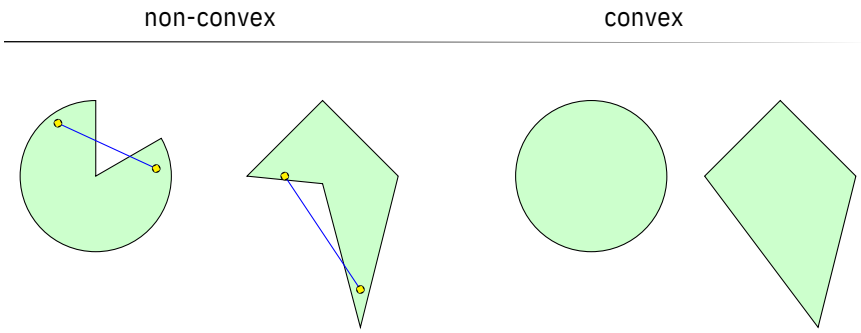
Finally, in Section 12.3, we show how one can handle some nonconvex problems by minimizing "surrogate" loss functions that are convex (instead of the original nonconvex loss function). Surrogate convex loss functions give rise to efficient solutions but might increase the risk of the learned predictor.

12.1 Convexity, Lipschitzness, and Smoothness

12.1.1 Convexity

definition 12.1 (Convex Set) A set C in a vector space is convex if for any two vectors u, v in C, the line segment between u and v is contained in C. That is, for any $\alpha \in [0, 1]$ we have that $\alpha u + (1 - \alpha)v \in C$.

Examples of convex and nonconvex sets in R^2 are given in the following. For the nonconvex sets, we depict two points in the set such that the line between the two points is not contained in the set.

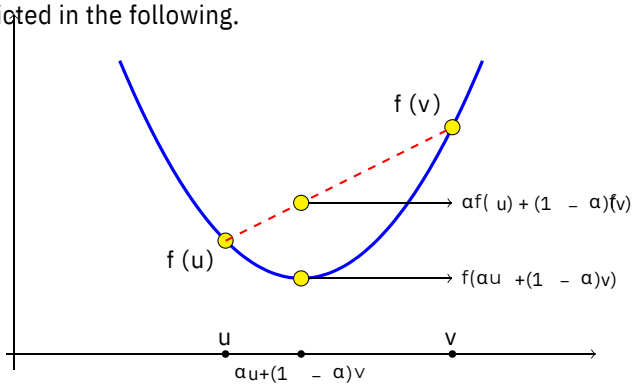


Given $\alpha \in [0, 1]$, the combination, $\alpha u + (1 - \alpha)v$ of the points u, v is called a convex combination.

definition 12.2 (Convex Function) Let C be a convex set. A function $f : C \subseteq \rightarrow R$ is convex if for every $u, v \in C$ and $\alpha \in [0, 1]$,

$f(\alpha u + (1 - \alpha)v) \leq \alpha f(u) + (1-\alpha)f(v)$.

In words, f is convex if for any u, v, the graph of f between u and v lies below the line segment joining f(u) and f(v). An illustration of a convex function, $f : R \rightarrow R$, is depicted in the following.

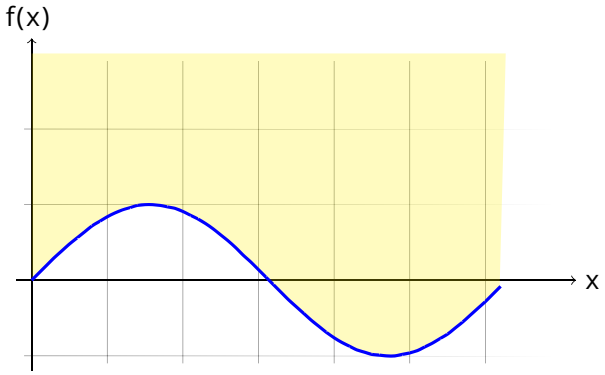


The epigraph of a function f is the set

epigraph(f) =

$$\{(x,\beta): f(x) \leq \beta\}. \quad (12.1)$$

It is easy to verify that a function f is convex if and only if its epigraph is a convex set. An illustration of a nonconvex function $f : R \rightarrow R$, along with its epigraph, is given in the following.



An important property of convex functions is that every local minimum of the function is also a global minimum. Formally, let $B(u, r) = \{ v : \| v - u \| \leq r \}$ be a ball of radius r centered around u. We say that f(u) is a local minimum of f at u if there exists some $r > 0$ such that for all $v \in B(u, r)$ we have $f(v) \geq f(u)$. It follows that for any v (not necessarily in B), there is a small enough $\alpha > 0$ such that $u + \alpha(v - u) \in B(u, r)$ and therefore

$$f(u) \leq f(u + \alpha(v - u)) \ . \tag{12.2}$$

If f is convex, we also have that

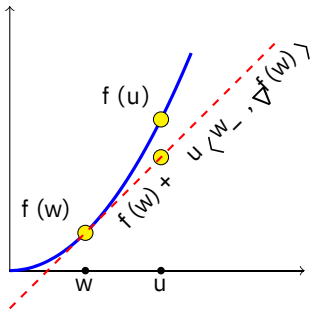
$$f(u + \alpha(v - u)) = f(\alpha v + (1 - \alpha)u) \leq (1 - \alpha) f(u) + \alpha f(v) \ . \tag{12.3}$$

Combining these two equations and rearranging terms, we conclude that $f(u) \leq f(v)$. Since this holds for every v, it follows that f(u) is also a global minimum of f.

Another important property of convex functions is that for every w we can construct a tangent to f at w that lies below f everywhere. If f is differentiable, this tangent is the linear function $l(u) = f(w) + \langle \nabla f(w), u - w \rangle$, where $\nabla f(w)$ is the gradient of f at w, namely, the vector whose ith coordinate is the derivative of $u \mapsto f(u)$ with respect to the ith coordinate of u. The following lemma shows that this property holds also for nondifferentiable functions.

$$\forall u, \quad f(u) \geq f(w) + \langle \nabla f(w), u - w \rangle. \tag{12.4}$$

In Chapter 14 we will generalize this inequality to nondifferentiable functions. An illustration of Equation (12.4) is given in the following.



If f is a scalar differentiable function, there is an easy way to check if it is convex.

lemma12.3 f:R Let beascalartwicedifferentialfunction,andlet f→,f′Rbe its first and second derivatives, respectively. Then, the following are equivalent:

1. f is convex
2. f′ is monotonically nondecreasing
3. f′′ is nonnegative

Example 12.1

• The scalar function f(x) = x2 is convex. To see this, note that f′(x) = 2x and f′′(x) = 2 > 0.

• Thescalarfunctionf(x)=log(1+exp(x))isconvex.Toseethis,observethat f′(x)= exp(x) = 1 −x)+1. This is a monotonically increasing function since the exponent function is a monotonically increasing function.
1+exp(x) exp(

The following claim shows that the composition of a convex scalar function with a linear function yields a convex vector-valued function.

claim12.4 f:Rd Assume that R for some ,and R → ⟨w, x⟩ + y), convexity of f .
can be written as f(w)=g(∈Rd,y∈R g: R.Then,convexityofgimpliesthe

Proof. Let w1,w2 ∈Rd and α∈[0,1].Wehave

f(αw1 + (1 − α)w2) = g(⟨αw1 + (1 − α) 2, ⟩ + y)
=g(α ⟨w1,x⟩ + (1 − α) ⟨ 2, ⟩ + y))
= g(α(⟨w1, x⟩ + y) + (1 − α)(⟨ 2, ⟩ +y),
≤ αg(⟨w1, x⟩ + y) + (1 − α)g(⟨ 2, ⟩ +y),

where the last inequality follows from the convexity of g. □

Example 12.2

• Given some x ∈ Rd and y ∈ R, let f : Rd → R be defined as f(w) = (⟨w,x⟩ − y)2. Then, f is a composition of the function g(a) = a2 onto a linear function, and hence f is a convex function.
• Given some x ∈ Rd and y ∈ {±1}, let f : Rd → R be defined as f(w) = log(1 + exp(−y ⟨w,x⟩)). Then, f is a composition of the function g(a) = log(1 + exp(a)) onto a linear function, and hence f is a convex function.

Finally, the following lemma shows that the maximum of convex functions is convex and that a weighted sum of convex functions, with nonnegative weights, is also convex.

claim 12.5 *For i = 1,...,r let $f_i : \mathbb{R}^d \to \mathbb{R}$ be a convex function. Then, the following functions from $\mathbb{R}^d$ to R are also convex.*

- *$g(x) = \max_{i \in [r]} f_i(x)$*
- *$g(x) = \sum_{i=1}^{r} w_i f_i(x)$, where for all i, $w_i \geq 0$.*

Proof The first claim follows by

$$g(\alpha u + (1-\alpha)v) = \max_i f_i(\alpha u + (1-\alpha)v) \leq \max_i [\alpha f_i(u) + (1-\alpha) f_i(v)]$$
$$\leq \alpha \max_i f_i(u) + (1-\alpha) \max_i f_i(v) = \alpha g(u) + (1-\alpha) g(v).$$

For the second claim

$$g(\alpha u + (1-\alpha)v) = \sum_i w_i f_i(\alpha u + (1-\alpha)v) \leq \sum_i w_i [\alpha f_i(u) + (1-\alpha) f_i(v)]$$
$$= \sum_i [\alpha w_i f_i(u) + (1-\alpha) w_i f_i(v)] = \alpha g(u) + (1-\alpha) g(v).$$

□

Example 12.3 The function $g(x) = |x|$ is convex. To see this, note that $g(x) = \max\{x, -x\}$ and that both the function $f_1(x) = x$ and $f_2(x) = -x$ are convex.

12.1.2 Lipschitzness

The definition of Lipschitzness below is with respect to the Euclidean norm over Rd. However, it is possible to define Lipschitzness with respect to any norm.

definition 12.6 (Lipschitzness) *Let $C \subset \mathbb{R}^d$. A function $f : \mathbb{R}^d \to \mathbb{R}^k$ is ρ-Lipschitz over C if for every $w_1, w_2 \in C$ we have that $\|f(w_1) - f(w_2)\| \leq \rho \|w_1 - w_2\|$.*

Intuitively, a Lipschitz function cannot change too fast. Note that if $f : R \to R$ is differentiable, then by the mean value theorem we have

$$f(w)-f(w)=f'(u)(w_1-w_2),$$

where u is some point between w1 and w2. It follows that if the derivative of f is everywhere bounded (in absolute value) by ρ, then the function is ρ-Lipschitz.

Example 12.4

- The function $f(x) = |x|$ is 1-Lipschitz over R. This follows from the triangle inequality: For every x1, x2,

$$|x_1| - |x_2| = |x_1 - x_2 + x_2| - |x_2| \le |x_1 - x_2| + |x_2| - |x_2|$$

Since this holds for both x1,x2 and x2,x1, we obtain that

$$||x_1| - |x_2|| \le |x_1 - x_2|.$$

- The function $f(x)=\log(1+\exp(x))$ is 1-Lipschitz over R. To see this, observe that $|f'(x)| = |\frac{\exp(x)}{1+\exp(x)}| = |\frac{1}{1+\exp(-x)}| \le 1$.
- The function $f(x) = x^2$ is not ρ-Lipschitz over R for any ρ. To see this, take $x_1=0$ and $x_2=1+\rho$, then

$$f(x_2)-f(x_1)=(1+\rho)^2-1^2>\rho(1+\rho)=\rho|x_2-x_1|.$$

However, this function is ρ-Lipschitz over the set $C = \{x : |x| \le \rho/2\}$. Indeed, for any x1, x2 ∈ C we have

$$|x_2^2 - x_1^2| = |x_1 + x_2||x_1 - x_2| \le 2(\rho/2)|x_1 - x_2| = \rho|x_1 - x_2|.$$

- The linear function $f : R^d \to R$ defined by $f(w) = \langle v,w\rangle + b$ where $v \in R^d$ is $\|v\|$-Lipschitz. Indeed, using Cauchy-Schwartz inequality,

$$|f(w_1) - f(w_2)| = |\langle v, w_1 - w_2\rangle| \le \|v\| \|w_1 - w_2\|.$$

The following claim shows that composition of Lipschitz functions preserves Lipschitzness.

claim 12.7 Let $f(x) = g_1(g_2(x))$, where g_1 is ρ_1-Lipschitz and g_2 is ρ_2-Lipschitz. Then, f is $(\rho_1\rho_2)$-Lipschitz. In particular, if g_2 is the linear function, $g_2(w) = \langle v,w\rangle + b$ for some $v \in R^d, b \in R$, then f is $\|v\|\rho_1$-Lipschitz.

Proof

$$|f(w_1) - f(w_2)| = |g_1(g_2(w_1)) - g_1(g_2(w_2))| \le \rho_1|g_2(w_1) - g_2(w_2)| \le \rho_1\rho_2\|w_1 - w_2\|.$$

□

12.1.3 Smo othness

The definition of a smooth function relies on the notion of gradient. Recall that the gradient of a differentiable function f : Rd
→ R a
(tw,denoted∇f()w),isthevectorofpartialderivativesoff,namely,∇f(w)=∂f(w)∂f(w)∂
w,...,1∂w.d

definition 12.8 (Smoothness) A differentiable function f : Rd
→Risβ-
smooth if its gradient is β-Lipschitz; namely, for all v,w we have
|| ∇ f (v) − ∇ f (w) || ≤ β || v − w || .

It is possible to show that smoothness implies that for all v,w we have
β
f(v)
≤f(w)+ 〈∇f(w),v−w〉 +||v−w||2. (12.5)2
Recall that convexity of f implies that f(v)
≥ f(w)+ 〈∇f(w), v−w〉 . Therefore, when a function is both convex and smooth, we have both upper and lower bounds on the difference between the function and its first order approximation. Setting v = w
− 1β∇f(w) in the right-hand side of Equation (12.5) and rearranging terms, we obtain
1
||∇f(w)||2≤f(w)−f(v).2β
If we further assume that f(v)
≥ 0 for all v we conclude that smoothness implies the following:
|| ∇ f(w)||2≤2βf(w). (12.6)
A function that satisfies this property is also called a self-bounded function.

Example 12.5
- The function f(x) = x2 is 2-smooth. This follows directly from the fact that f′(x) = 2x. Note that for this particular function Equation (12.5) and Equation (12.6) hold with equality.
- Thefunctionf(x)=log(1+exp(x))is(1/4)-smooth.Indeed,sincef′(x)=1
1+exp(
−x) we have that
|′′|exp(−x)1f(x)==≤1/4.(1+exp(
−x))2 (1+exp(−x))(1+exp(x))
Hence, f′ is (1/4)-Lipschitz. Since this function is nonnegative, Equation (12.6) holds as well.

The following claim shows that a composition of a smooth scalar function over a linear function preserves smoothness.

claim 12.9 Let f(w) = g(〉
〈w, x||+b), where g : R → R is a β-smooth function, x
∈ Rd, and b ∈ R. Then, f is (β x 2)-smooth.

Proof By the chain rule we have that $\nabla f(w) = g'(\langle w, x\rangle + b)x$, where g' is the derivative of g. Using the smoothness of g and the Cauchy-Schwartz inequality we therefore obtain

$$f(v) = g(\langle v, x\rangle + b) \leq g(\langle w, x\rangle + b) + g'(\langle w, x\rangle + b)\langle v - w, x\rangle + \frac{\beta}{2}(\langle v - w, x\rangle)^2$$

$$\leq g(\langle w, x\rangle + b) + g'(\langle w, x\rangle + b)\langle v - w, x\rangle + \frac{\beta}{2}\|v - w\|^2 \|x\|^2$$

$$= f(w) + \langle \nabla f(w), v - w\rangle + \frac{\beta}{2}\|v - w\|^2 .$$

□

Example 12.6

- For any $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$, let $f(w) = \frac{1}{2}(\langle w,x\rangle - y)^2$. Then, f is $(\|x\|^2)$-smooth.
- For any $x \in \mathbb{R}^d$ and $y \in \{\pm 1\}$, let $f(w) = \log(1+\exp(-y\langle w,x\rangle))$. Then, f is $(\|x\|^2/4)$-smooth.

12.2 Convex Learning Problems

Recall that in our general definition of learning (Definition 3.4 in Chapter 3), we have a hypothesis class H, a set of examples Z, and a loss function $\ell : H \times Z \to \mathbb{R}_+$. So far in the book we have mainly thought of Z as being the product of an instance space and a target space, $Z = X \times Y$, and H being a set of functions from X to Y. However, H can be an arbitrary set. Indeed, throughout this chapter, we consider hypothesis classes H that are subsets of the Euclidean space $\mathbb{R}^d$. That is, every hypothesis is some real-valued vector. We shall, therefore, denote a hypothesis in H by w. Now we can finally define convex learning problems:

Definition 12.10 (Convex Learning Problem) A learning problem, (H, Z, ℓ), is called convex if the hypothesis class H is a convex set and for all $z \in Z$, the loss function, $\ell(\cdot, z)$, is a convex function (where, for any z, $\ell(\cdot, z)$ denotes the function $f : H \to \mathbb{R}$ defined by $f(w) = \ell(w, z)$).

Example 12.7 (Linear Regression with the Squared Loss) Recall that linear regression is a tool for modeling the relationship between some "explanatory" variables and some real valued outcome (see Chapter 9). The domain set X is a subset of $\mathbb{R}^d$, for some d, and the label set Y is the set of real numbers. We would like to learn a linear function $h : \mathbb{R}^d \to \mathbb{R}$ that best approximates the relationship between our variables. In Chapter 9 we defined the hypothesis class as the set of homogenous linear functions, $H = \{x \mapsto \langle w, x\rangle : w \in \mathbb{R}^d\}$, and used the squared loss function, $\ell(h,(x,y)) = (h(x) - y)^2$. However, we can equivalently model the learning problem as a convex learning problem as follows.

Each linear function is parameterized by a vector w ∈ Rd. Hence, we can define H to be the set of all such parameters, namely, H = Rd. The set of examples is Z = X ×Y = Rd×R = Rd+1, and the loss function is `(w, (x, y)) = (⟨w, x⟩ −y)2. Clearly, the set H is a convex set. The loss function is also convex with respect to its first argument (see Example 12.2).

If is a convex loss function and the class H is convex, then the ERM lemma 12.11 `

H problem, of minimizing the empirical loss over H, is a convex optimization problem (that is, a problem of minimizing a convex function over a convex set).

Recall that the ERM Proof problem is defined by

ERM

H(S) = argmin LS(w).

w ∈ H

∑mSince,forasampleS=z,.,L11..zm,foreveryw,S(w)=mi=1(w,zi),Claim12.5impliesthatLS(w)isaconvexfu □

is a problem of minimizing a convex function subject to the constraint that the solution should be in a convex set.

Under mild conditions, such problems can be solved efficiently using generic optimization algorithms. In particular, in Chapter 14 we will present a very simple algorithm for minimizing convex functions.

12.2.1 Learnability of Convex Learning Problems

We have argued that for many cases, implementing the ERM rule for convex learning problems can be done efficiently. But is convexity a sufficient condition for the learnability of a problem?

To make the quesion more specific: In VC theory, we saw that halfspaces in d-dimension are learnable (perhaps inefficiently). We also argued in Chapter 9 using the "discretization trick" that if the problem is of d parameters, it is learnable with a sample complexity being a function of d. That is, for a constant d, the problem should be learnable. So, maybe all convex learning problems over Rd, are learnable?

Example 12.8 later shows that the answer is negative, even when d is low. Not all convex learning problems over Rd are learnable. There is no contradiction to VC theory since VC theory only deals with binary classification while here we consider a wide family of problems. There is also no contradiction to the "discretization trick" as there we assumed that the loss function is bounded and also assumed that a representation of each parameter using a finite number of bits suffices. As we will show later, under some additional restricting conditions that hold in many practical scenarios, convex problems are learnable.

Example 12.8 (Nonlearnability of Linear Regression Even If d = 1) Let H=R, and the loss be the squared loss: `(w, (x, y)) = (wx − y)2 (we're referring to the

homogenous case). Let A be any deterministic algorithm.[1] Assume, by way of contradiction, that A is a successful PAC learner for this problem. That is, there exists a function m(·,·), such that for every distribution D and for every ε,δ if A receives a training set of size m ≥ m(ε, δ), it should output, with probability of at least 1 − δ, a hypothesis ŵ = A(S), such that $L_D(\hat{w}) - \min_w L_D(w) \le \varepsilon$. Choose ε = 1/100,δ = 1/2, let m ≥m(ε,δ),andsetμ=log(100/99)2m.Wewill define two distributions, and will show that A is likely to fail on at least one of them. The first distribution, D1, is supported on two examples, z1 = (1,0) and z2 = (μ, −1), where the probability mass of the first example is μ while the probability mass of the second example is 1 − μ. The second distribution, D2, is supported entirely on z2.

Observe that for both distributions, the probability that all examples of the training set will be of the second type is at least 99%. This is trivially true for D2, whereas for D1, the probability of this event is (1 − μ)m ≥ e−2μm = 0.99. However, since we assume that A is a deterministic algorithm, upon receiving a training set of m examples, each of which is (μ, −1), the algorithm will output some ŵ. It follows that $\min_w L_{D_1}(w) \le L_{D_1}(0) = (1 - \mu)$. Now, if ŵ < 1 −1/(2μ), we will set the distribution to be D1. Hence, $L_{D_1}(\hat{w}) - \min_w L_{D_1}(w) \ge -(1-\mu) > \varepsilon$.w4μ $L_{D_1}(\hat{w}) \ge \mu(\hat{w})2 \ge 1/(4\mu)$. Therefore, such algorithm A fails on D1. On the other hand, if ŵ ≥ −1/(2μ) then we'll set the distribution to be D2.ThenwehavethatLD(ŵ)≥1/4while2 minw L D2(w) = 0, so A fails on D2. In summary, we have shown that for every A there exists a distribution on which A fails, which implies that the problem is not PAC learnable.

A possible solution to this problem is to add another constraint on the hypothesis class. In addition to the convexity requirement, we require that H will be bounded; namely, we assume that for some predefined scalar B, every hypothesis w ∈ H satisfies ||w|| ≤ B.

Boundedness and convexity alone are still not sufficient for ensuring that the problem is learnable, as the following example demonstrates. | ≤ } ⊂

Example12.9 AsinExample 12.8,consideraregressionproblemwiththe squared loss. However, this time let H={w:|w 1 Rbeabounded wil also fail to learn the problem.

[1] Namely, given S the output of A is determined. This requirement is for the sake of simplicity. A slightly more involved argument will show that nondeterministic algorithms

hypothesis class. It is easy to verify that H is convex. The argument will be the same as in Example 12.8, except that now the two distributions, D1, D2 will be supported on z1 = (1/μ,0) and z2 = (1, −1). If the algorithm A returns ŵˆ< −1/2 upon receiving m examples of the second type, then we will set the distribution to be D1 and have that

L

D(ŵˆ)−minL(w)≥μ(ŵˆ/μ2L/μ1D)−1D(0)≥1(4μ)−(1−)>ε.1w

Similarly, if ŵˆ

≥ −1/2 we will set the distribution to be D2 and have that

L

D2(ŵˆ)−minLD2(w)≥(−1/2+1)2−0>ε.w

This example shows that we need additional assumptions on the learning problem, and this time the solution is in Lipschitzness or smoothness of the loss function. This motivates a definition of two families of learning problems, convex-Lipschitz-bounded and convex-smooth-bounded, which are defined later.

12.2.2 Convex-Lipschitz/Smooth-Bounded Learning Problems

definition 12.12 (Convex-Lipschitz-Bounded Learning Problem) A learning problem, (H,Z,`), is called Convex-Lipschitz-Bounded, with parameters ρ,B if the following holds:

• ThehypothesisclassHisaconvexsetandforallw∈Hwehave||w||≤B.

• Forallz∈Z,thelossfunction,`(·,z),isaconvexandρ-Lipschitzfunction.

Example12.10 Let X={x∈Rd:||x||≤ρ}andY=R.LetH={w∈Rd: ||w|| ≤ B} and let the loss function be `(w,(x,y)) = |⟨w,x⟩ − y|. This corresponds to a regression problem with the absolute-value loss, where we assume that the instances are in a ball of radius ρ and we restrict the hypotheses to be homogenous linear functions defined by a vector w whose norm is bounded by B. Then, the resulting problem is Convex-Lipschitz-Bounded with parameters ρ, B.

definition 12.13 (Convex-Smooth-Bounded Learning Problem) A learning problem, (H,Z,`), is called Convex-Smooth-Bounded, with parameters β,B if the following holds:

• ThehypothesisclassHisaconvexsetandforallw∈Hwehave||w||≤B.

• Forallz∈Z,thelossfunction,`(·,z),isaconvex,nonnegative,andβ-smooth function.

Note that we also required that the loss function is nonnegative. This is needed to ensure that the loss function is self-bounded, as described in the previous section.

Example12.11 Let X={x∈Rd:||x||≤β/2}andY=R.LetH={w∈Rd: ||w|| ≤ B} and let the loss function be `(w,(x,y)) = (⟨w,x⟩ − y)2. This corresponds to a regression problem with the squared loss, where we assume that the instances are in a ball of radius β/2 and we restrict the hypotheses to be homogenous linear functions defined by a vector w whose norm is bounded by B. Then, the resulting problem is Convex-Smooth-Bounded with parameters β,B.

We claim that these two families of learning problems are learnable. That is, the properties of convexity, boundedness, and Lipschitzness or smoothness of the loss function are sufficient for learnability. We will prove this claim in the next chapters by introducing algorithms that learn these problems successfully.

12.3 Surrogate Loss Functions

As mentioned, and as we will see in the next chapters, convex problems can be learned effficiently. However, in many cases, the natural loss function is not convex and, in particular, implementing the ERM rule is hard.

As an example, consider the problem of learning the hypothesis class of half-spaces with respect to the 0 − 1 loss. That is,

$$\ell_{0-1}(w, (x, y)) = 1[y=\text{sign}(\langle w,x\rangle)] = 1[y\langle w,x\rangle \le 0].$$

This loss function is not convex with respect to w and indeed, when trying to minimize the empirical risk with respect to this loss function we might encounter local minima (see Exercise 1). Furthermore, as discussed in Chapter 8, solving the ERM problem with respect to the 0 −1 loss in the unrealizable case is known to be NP-hard.

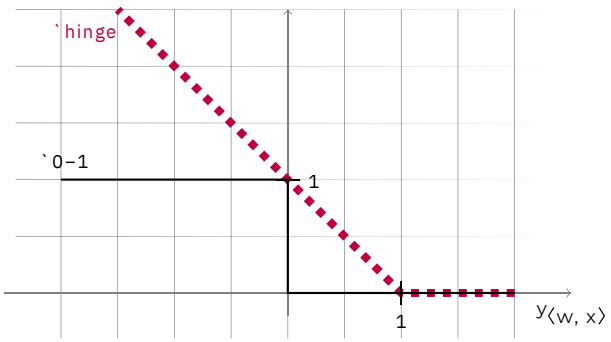
To circumvent the hardness result, one popular approach is to upper bound the nonconvex loss function by a convex surrogate loss function. As its name indicates, the requirements from a convex surrogate loss are as follows:

1. It should be convex.
2. It should upper bound the original loss.

For example, in the context of learning halfspaces, we can define the so-called hinge loss as a convex surrogate for the 0 − 1 loss, as follows.

$$\ell_{\text{hinge}}(w,(x,y)) = \max\{0, 1 - y\langle w, x\rangle\}.$$

Clearly, for all w and all (x,y), $\ell_{0-1}(w,(x,y)) \le \ell_{\text{hinge}}(w,(x,y))$. In addition, the convexity of the hinge loss follows directly from Claim 12.5. Hence, the hinge loss satisfies the requirements of a convex surrogate loss function for the zero-one loss. An illustration of the functions ℓ_{0-1} and ℓ_{hinge} is given in the following.



Once we have defined the surrogate convex loss, we can learn the problem with respect to it. The generalization requirement from a hinge loss learner will have the form

Lhinge
D(A(S))≤minLhinge(w)+ε,w
∈HD

where Lhinge

D(w)=Ex,y D[`hinge()(w,(x,y))].Usingthesurrogateproperty,we can lower bound the left-hand side by L0−1
D (A(S)),whichyields

L0−1

D(A(S))≤minLhinge(w)+ε.w
∈HD

We can further rewrite the upper bou

(ndasfollows:)L0−1(A(S))≤minL0−1(wLhingew−L0−1D)+min()min(w)+ε.w∈HDDw∈Hw∈HD

That is, the 0 −1 error of the learned predictor is upper bounded by three terms:

•Approximationerror:Thisisthetermmin L0−1w∈HD(w),whichmeasureshow well the hypothesis class performs on the distribution. We already elaborated on this error term in Chapter 5.

• Estimation error: This is the error that results from the fact that we only receive a training set and do not
elaborated on this error term

(observethedistributionD.WealreadyinChapter5.)•Optimizationerror:ThisisthetermminLhingew−L0−1 to the surrogate loss and the approximation error with respect to the original loss. The optimization error is a result of our inability to minimize the training loss with respect to the original loss. The size of this error depends on the specific distribution of the data and on the specific surrogate loss we are using.

12.4 Summary

We introduced two families of learning problems: convex-Lipschitz-bounded and convex-smooth-bounded. In the next two chapters we will describe two generic

learning algorithms for these families. We also introduced the notion of convex surrogate loss function, which enables us also to utilize the convex machinery for nonconvex problems.

12.5 Bibliographic Remarks

There are several excellent books on convex analysis and optimization (Boyd & Vandenberghe 2004, Borwein & Lewis 2006, Bertsekas 1999, Hiriart-Urruty & Lemaré ´chal1996).Regardinglearningproblems,thefamilyofconvex-Lipschitz-bounded problems was first studied by Zinkevich (2003) in the context of online learning and by Shalev-Shwartz, Shamir, Sridharan & Srebro (2009) in the context of PAC learning.

12.6 Exercises

1. Construct an example showing that the 0 −1 loss function may suffer from local minima; namely, construct a training sample S ∈ (X × {±1})m (say, for X = R2), for which there exist a vector w and some ε > 0 such that
1. For any w′ such that ||w−w′||≤εwehaveLS(w)≤L w′S( )(wherethe loss here is the 0
−1 loss). This means that w is a local minimum of LS.
2. There exists some w ⋆ such that L w ⋆ S( ) < LS(w). This means that w is not a global minimum of LS.
2. Consider the learning problem of logistic regression: Let H = X = { x ∈ Rd: ||x|| ≤ B}, for some scalar B > 0, let Y = {±1}, and let the loss function ` be defined as ` (w,(x,y)) = log(1 + exp(−y ⟨w,x⟩)). Show that the resulting learning problem is both convex-Lipschitz-bounded and convex-smooth-bounded. Specify the parameters of Lipschitzness and smoothness.
3. Consider the problem of learning halfspaces with the hinge loss. We limit our domain to the Euclidean ball with radius R. That is, X = { x : || x || 2 ≤ R } . The label set is Y = {±1} and the loss function ` is defined by ` (w, (x, y)) = max {0, 1 – y ⟨w, x⟩ }. We already know that the loss function is convex. Show that it is R-Lipschitz.
4. (*) Convex-Lipschitz-Boundedness Is Not Sufficient for Computational Efficiency: In the next chapter we show that from the statistical perspective, all convex-Lipschitz-bounded problems are learnable (in the agnostic PAC model). However, our main motivation to learn such problems resulted from the computational perspective – convex optimization is often efficiently solvable. Yet the goal of this exercise is to show that convexity alone is not sufficient for efficiency. We show that even for the case d = 1, there is a convex-Lipschitz-bounded problem which cannot be learned by any computable learner.
Let the hypothesis class be H = [0,1] and let the example domain, Z, be