MACHINE LEARNING

**Table of Contents**

1. **Unit I**
2. Introduction
3. A Gentle Start
4. A Formal Learning Model
5. Learning via Uniform Convergence
6. The Bias-Complexity
7. The VC-Dimension
8. Non-uniform Learnability
9. The Runtime of Learning
10. **Unit II**
11. Linear Predictors
12. Boosting
13. Model Selection and Validation
14. Convex Learning Problems
15. Regularization and Stability
16. Stochastic Gradient Descent
17. Support Vector Machines
18. Kernel Methods
19. **Unit III**
20. Multiclass, Ranking, and Complex Prediction Problems
21. Decision Trees
22. Nearest Neighbor
23. Neural Networks
24. Online Learning
25. Clustering
26. Dimensionality Reduction
27. Generative Models
28. **Unit IV**
29. Feature Selection and Generation
30. Rademacher Complexities
31. Covering Numbers
32. Proof of the Fundamental Theorem of Learning Theory
33. Python For Machine Learning
34. Multiclass Learnability
35. Compression Bounds
36. PAC-Bayes
37. Introduction

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 ani-mal 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 rst eat very small amounts, and subsequent feeding will depend on the avor of the food and its physiological e ect. If the food produces an ill e ect, 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 e ect 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 lter 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

1. Introduction

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 use-ful, 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 ltering 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 accord-ingly. 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 rst delivered, it found each pigeon engaged in some activity (pecking, turning the head, etc.). The arrival of food reinforced each bird's speci c 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 nd 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 delivered. They subsequently continue to perform these same actions diligently.

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 out random meaningless learning conclusions, once we export the task of learning to a machine, we must provide well de ned 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 rst 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

|  |  |
| --- | --- |
| 1.2 When Do We Need Machine Learning? | 21 |
|  |

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 a ect 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page60)). The development of tools for expressing domain expertise, translating it into a learning bias, and quantifying the e ect 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 exible the learning is { it is bound, a priori, by the commitment to these assumptions. We shall discuss these issues explicitly in Chapter [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page60).

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 concern-ing how we do them is not su ciently elaborate to extract a well

1. Introduction

de ned 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 su ciently many training examples.

Tasks beyond Human Capabilities: Another wide family of tasks that bene t from machine learning techniques are related to the analy-sis of very large and complex data sets: astronomical data, turning medical archives into medical knowledge, weather prediction, anal-ysis 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 combi-nation 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 { o er 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 xed program can adapt to variations between the handwriting of di erent 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 eld of machine learning has branched into several sub elds dealing with di erent types of learn-ing 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 eld of machine learning.

We describe four parameters along which learning paradigms can be classi ed.

Supervised versus Unsupervised Since learning involves an interaction be-tween the learner and the environment, one can divide learning tasks according to the nature of that interaction. The rst distinction to note is the di erence between supervised and unsupervised learning. As an

|  |  |
| --- | --- |
| 1.3 Types of Learning | 23 |
|  |  |

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 gure out a rule for labeling a newly arriving e-mail mes-sage. 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 signi cant 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 ac-quired 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 in uencing or directing it. Note that the learner of a spam lter 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 speci c 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-

1. Introduction

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 sce-narios 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 ltering example (if the spammer makes an e ort to mislead the spam ltering 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 brie y discuss online learning and batch unsupervised learning (in particular, clustering).

1.4 Relations to Other Fields

As an interdisciplinary eld, machine learning shares common threads with the mathematical elds of statistics, information theory, game theory, and optimiza-tion. It is naturally a sub eld 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 (Arti cial 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 perform-ing 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 en-vironment 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 signi cant di erences 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 gure 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 e ciency. Another di erence is that while statistics is often interested in asymptotic behavior (like the convergence of sample-based statistical estimates as the sample sizes grow to in nity), the theory of machine learning focuses on nite sample bounds. Namely, given the size of available samples, machine learning theory aims to gure out the degree of accuracy that a learner can expect on the basis of such samples.

1. A Gentle Start

Let us begin our mathematical analysis by showing how successful learning can be achieved in a relatively simpli ed setting. Imagine you have just arrived in some small Paci c island. You soon nd out that papayas are a signi cant 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 guring 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 rst 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 men-tioned 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 f0; 1g or f 1; +1g. Let Y denote our set of possible labels. For our papayas example, let Y be f0; 1g, where 1 represents being tasty and 0 stands for being not-tasty.

{ Training data: S = ((x1; y1) : : : (xm; ym)) is a nite sequence of pairs in

X 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

1. A Gentle Start

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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page34)

The learner's output: The learner is requested to output a prediction rule, h : X ! Y. This function is also called a predictor, a hypothesis, or a clas-si er. 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 ! Y, and that yi = f(xi) 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 gure out. In summary, each pair in the training data S is generated by rst sampling a point xi according

to D and then labeling it by f.

Measures of success: We de ne the error of a classi er 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page34) A X , the probability distribution, D, assigns a number, D(A), which determines how likely it is to observe a point x 2 A. In many cases, we refer to A as an event and express it using a function : X ! f0; 1g, namely, A = fx 2 X : (x) = 1g. In that case, we also use the notation Px D[ (x)] to express D(A).

We de ne the error of a prediction rule, h : X ! Y, to be

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| def |  | def | D(fx : h(x) 6= f(x)g): | (2.1) |
| LD;f (h) = | x | P [h(x) 6= f(x)] = |
|  |  | D |  |  |

That is, the error of such h is the probability of randomly choosing an example x for which h(x) 6= 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.

1. Strictly speaking, we should be more careful and require that A is a member of some -algebra of subsets of X, over which D is de ned. We will formally de ne our measurability assumptions in the next chapter.

|  |  |
| --- | --- |
| 2.2 Empirical Risk Minimization | 35 |
|  |  |

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 general-ization 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 hS : X ! Y (the subscript S emphasizes the fact that the output predictor depends on S). The goal of the algorithm is to nd hS 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 classi er incurs over the training sample:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| LS(h) = | jf |  | 2 | i 6 | igj | ; | (2.2) |
| def |  | i |  | [m] : h(x ) = y |  |  |  |

m

where [m] = f1; : : : ; mg.

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 LS(h) { is called Empirical Risk Minimization or ERM for short.

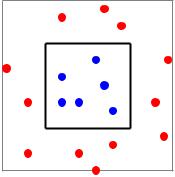
2.2.1 Something May Go Wrong { Over tting

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

1. A Gentle Start

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:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| hS(x) = | ( | yi | if 9i 2 [m] s:t: xi = x | (2.3) |
| 0 | otherwise: |

While this predictor might seem rather arti cial, in Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page41) we show a natural representation of it using polynomials. Clearly, no matter what the sample is, LS(hS) = 0, and therefore this predictor may be chosen by an ERM algorithm (it is one of the empirical-minimum-cost hypotheses; no classi er can have smaller error). On the other hand, the true error of any classi er that predicts the label 1 only on a nite number of instances is, in this case, 1=2. Thus, LD(hS) = 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 over tting. Intuitively, over tting occurs when our hypothesis ts 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 over tting. 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 over t, 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 H. Each h 2 H is a function mapping from X to Y. For a given class H, and a training sample, S, the ERMH learner uses the ERM rule to choose a predictor h 2 H,

|  |  |
| --- | --- |
| 2.3 Empirical Risk Minimization with Inductive Bias | 37 |
|  |  |

with the lowest possible error over S. Formally,

ERMH(S) 2 argmin LS(h);

h2H

where argmin stands for the set of hypotheses in H that achieve the minimum value of LS(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 ERMH over this class is guaranteed not to over t. On the other hand, the example of over tting 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 nite set of domain points does not su ce to guarantee that ERMH will not over t.

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

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

2.3.1 Finite Hypothesis Classes

The simplest type of restriction on a class is imposing an upper bound on its size (that is, the number of predictors h in H). In this section, we show that if H is a nite class then ERMH will not over t, provided it is based on a su ciently large training sample (this size requirement will depend on the size of H).

Limiting the learner to prediction rules within some nite 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 in nite class, if we discretize the repre-sentation of real numbers, say, by using a 64 bits oating-point representation, the hypothesis class becomes a nite class.

Let us now analyze the performance of the ERMH learning rule assuming that H is a nite class. For a training sample, S, labeled according to some f : X ! Y, let hS denote a result of applying ERMH to S, namely,

|  |  |
| --- | --- |
| hS 2 argmin LS(h): | (2.4) |
| h2H |  |

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

1. A Gentle Start

definition 2.1 (The Realizability Assumption) There exists h? 2 H s.t. L(D;f)(h?) = 0. Note that this assumption implies that with probability 1 over random samples, S, where the instances of S are sampled according to D and are labeled by f, we have LS(h?) = 0.

The realizability assumption implies that for every ERM hypothesis we have that[3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page38) LS(hS) = 0. However, we are interested in the true risk of hS, L(D;f)(hS), rather than its empirical risk.

Clearly, any guarantee on the error with respect to the underlying distribution, 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 re ect 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 su ce to direct the learner toward a good classi er (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, ERMH(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 , and call (1 ) the con dence parameter of our prediction.

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

1. Mathematically speaking, this holds with probability 1. To simplify the presentation, we sometimes omit the \with probability 1" speci er.

|  |  |
| --- | --- |
| 2.3 Empirical Risk Minimization with Inductive Bias | 39 |
|  |  |

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

Dm(fSjx : L(D;f)(hS) > g):

Let HB be the set of \bad" hypotheses, that is,

HB = fh 2 H : L(D;f)(h) > g:

In addition, let

1. = fSjx : 9h 2 HB; LS(h) = 0g

be the set of misleading samples: Namely, for every Sjx 2 M, there is a \bad" hypothesis, h 2 HB, that looks like a \good" hypothesis on Sjx. Now, recall that we would like to bound the probability of the event L(D;f)(hS) > . But, since the realizability assumption implies that LS(hS) = 0, it follows that the event L(D;f)(hS) > can only happen if for some h 2 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

fSjx : L(D;f)(hS) > g M :

Note that we can rewrite M as

|  |  |  |
| --- | --- | --- |
|  | [ |  |
| M = | fSjx : LS(h) = 0g: | (2.5) |

h2HB

Hence,

Dm(fSjx : L(D;f)(hS) > g) Dm(M) = Dm([h2HB fSjx : LS(h) = 0g):

(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 [ B) D(A) + D(B):

Applying the union bound to the right-hand side of Equation ([2.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page39)) yields

|  |  |  |
| --- | --- | --- |
| Dm(fSjx : L(D;f)(hS) > g) | X |  |
| Dm(fSjx : LS(h) = 0g): | (2.7) |

h2HB

Next, let us bound each summand of the right-hand side of the preceding in-equality. Fix some \bad" hypothesis h 2 HB. The event LS(h) = 0 is equivalent

1. A Gentle Start

to the event 8i; h(xi) = f(xi). Since the examples in the training set are sampled i.i.d. we get that

Dm(fSjx : LS(h) = 0g) = Dm(fSjx : 8i; h(xi) = f(xi)g)

|  |  |
| --- | --- |
| m |  |
| iY |  |
| = D(fxi : h(xi) = f(xi)g): | (2.8) |

=1

For each individual sampling of an element of the training set we have D(fxi : h(xi) = yig) = 1 L(D;f)(h) 1 ;

where the last inequality follows from the fact that h 2 HB. Combining the previous equation with Equation ([2.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page40)) and using the inequality 1 e we obtain that for every h 2 HB,

|  |  |
| --- | --- |
| Dm(fSjx : LS(h) = 0g) (1 )m e m: | (2.9) |

Combining this equation with Equation ([2.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page39)) we conclude that

Dm(fSjx : L(D;f)(hS) > g) jHBj e m jHj e m:

A graphical illustration which explains how we used the union bound is given in Figure [2.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page40).

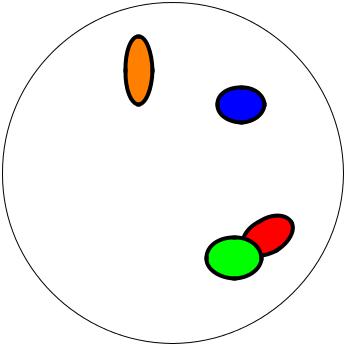


Figure 2.1 Each point in the large circle represents a possible m-tuple of instances. Each colored oval represents the set of \misleading" m-tuple of instances for some \bad" predictor h 2 HB. The ERM can potentially over t whenever it gets a misleading training set S. That is, for some h 2 HB we have LS(h) = 0.

Equation ([2.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page40)) guarantees that for each individual bad hypothesis, h 2 HB, at most (1 )m-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 2 HB (that is, the training sets in M) is at most the sum of the areas of the colored ovals. Therefore, it is bounded by jHBj times the maximum size of a colored oval. Any sample S outside the colored ovals cannot cause the ERM rule to over t.

corollary 2.3 Let H be a nite hypothesis class. Let 2 (0; 1) and > 0

|  |  |
| --- | --- |
| 2.4 Exercises | 41 |
|  |

and let m be an integer that satis es

1. log(jHj= ) :

Then, for any labeling function, f, and for any distribution, D, for which the realizability assumption holds (that is, for some h 2 H, L(D;f)(h) = 0), with probability of at least 1 over the choice of an i.i.d. sample S of size m, we have that for every ERM hypothesis, hS, it holds that

L(D;f)(hS) :

The preceeding corollary tells us that for a su ciently large m, the ERMH rule over a nite hypothesis class will be probably (with con dence 1 ) approximately (up to an error of ) correct. In the next chapter we formally de ne the model of Probably Approximately Correct (PAC) learning.

2.4 Exercises

1. Over tting of polynomial matching: We have shown that the predictor de ned in Equation ([2.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page36)) leads to over tting. 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 = f(xi; f(xi))gmi=1 (Rd f0; 1g)m, there exists a polynomial pS such that hS(x) = 1 if and only if pS(x) 0, where hS is as de ned in Equation ([2.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page36)). It follows that learning the class of all thresholded polynomials using the ERM rule may lead to over tting.

1. Let H be a class of binary classi ers over a domain X . Let D be an unknown distribution over X , and let f be the target hypothesis in H. Fix some h 2 H. Show that the expected value of LS(h) over the choice of Sjx equals L(D;f)(h), namely,

E [LS(h)] = L(D;f)(h):

Sjx Dm

1. Axis aligned rectangles: An axis aligned rectangle classi er in the plane

is a classi er that assigns the value 1 to a point if and only if it is inside a certain rectangle. Formally, given real numbers a1 b1; a2 b2, de ne the

classi er h(a1;b1;a2;b2) by

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| h(a1;b1;a2;b2)(x1; x2) = ( | 1 | if a1 x1 b1 | and a2 | x2 | b2 | : | (2.10) |
| 0 | otherwise |  |  |  |

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

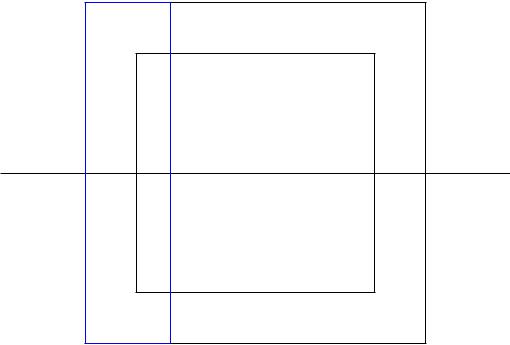
Hrec2 = fh(a1;b1;a2;b2) : a1 b1; and a2 b2g:

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

1. A Gentle Start
   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 | | | | | 4 log(4= ) | | | | then, with proba- |
|  |  | |  |
| bility of at least 1it returns a hypothesis with error of at most . | | | | | | | | | |
| Hint: Fix some distribution | D | over | X | , let R = R(a | | | ; b | ; a ; b ) be the rect- | |
|  |  |  | 1 | | 1 |  | 2 2 |

angle that generates the labels, and let f be the corresponding hypothesis. Let a1 a1 be a number such that the probability mass (with respect to D) of the rectangle R1 = R(a1; a1; a2; b2) is exactly =4. Similarly, let b1; a2; b2 be numbers such that the probability masses of the rectangles R2 = R(b1; b1; a2; b2); R3 = R(a1; b1; a2; a2); R4 = R(a1; b1; b2; b2) are all exactly =4. Let R(S) be the rectangle returned by A. See illustration in Figure [2.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page42).



R

- +

1. R(S)
   1. -

+

+

R1

Figure 2.2 Axis aligned rectangles.

Show that R(S) R .

Show that if S contains (positive) examples in all of the rectangles R1; R2; R3; R4, then the hypothesis returned by A has error of at most .

For each i 2 f1; : : : ; 4g, upper bound the probability that S does not contain an example from Ri.

Use the union bound to conclude the argument.

1. Repeat the previous question for the class of axis aligned rectangles in Rd.
2. Show that the runtime of applying the algorithm A mentioned earlier is polynomial in d; 1= ; and in log(1= ).

1. A Formal Learning Model

In this chapter we de ne our main formal learning model { the PAC learning model and its extensions. We will consider other notions of learnability in Chap-ter [7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page83).

3.1 PAC Learning

In the previous chapter we have shown that for a nite hypothesis class, if the ERM rule with respect to that class is applied on a su ciently 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 de ne Probably Approximately Correct (PAC) learning.

definition 3.1 (PAC Learnability) A hypothesis class H is PAC learnable if there exist a function mH : (0; 1)2 ! N and a learning algorithm with the following property: For every ; 2 (0; 1), for every distribution D over X , and for every labeling function f : X ! f0; 1g, if the realizable assumption holds with respect to H; D; f, then when running the learning algorithm on m mH( ; ) 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 (over the choice of the examples), L(D;f)(h) .

The de nition of Probably Approximately Correct learnability contains two approximation parameters. The accuracy parameter determines how far the output classi er can be from the optimal one (this corresponds to the \approx-imately correct"), and a con dence parameter indicating how likely the clas-si er is to meet that accuracy requirement (corresponds to the \probably" part of \PAC"). Under the data access model that we are investigating, these ap-proximations are inevitable. Since the training set is randomly generated, there may always be a small chance that it will happen to be noninformative (for ex-ample, 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 D, because it is just a nite sample, there may always be some ne details of D that it fails

1. A Formal Learning Model

to re ect. Our accuracy parameter, , allows \forgiving" the learner's classi er for making minor errors.

Sample Complexity

The function mH : (0; 1)2 ! 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 ( ) and con dence ( ) parameters. It also depends on properties of the hypothesis class

H { for example, for a nite 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 de nition of PAC learnability. Therefore, to be precise,

we will de ne the sample complexity of learning H to be the \minimal function," in the sense that for any ; , mH( ; ) is the minimal integer that satis es the requirements of PAC learning with accuracy and con dence .

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

corollary 3.2 Every nite hypothesis class is PAC learnable with sample complexity

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | H |  |  |  |  |  |
| m |  | ( ; ) |  |  | log(jHj= ) | : |
|  |  |  |  |

There are in nite classes that are learnable as well (see, for example, Exer-cise [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page41)). Later on we will show that what determines the PAC learnability of a class is not its niteness 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 distri-bution 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.

|  |  |
| --- | --- |
| 3.2 A More General Learning Model | 45 |
|  |  |

Learning Problems beyond Binary Classi cation

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 di erent 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 nite 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page47) 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? 2 H such that Px 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 di erent taste). In the following, we relax the realizability assumption by replacing the \target labeling function" with a more exible 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

1. = f0; 1g). 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)jx). 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 di erent 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 rede ne the true error (or risk) of a prediction rule h to be

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| def | (x;yP) | def | D(f(x; y) : h(x) 6= yg): | (3.1) |
| LD(h) = | [h(x) 6= y] = |
|  | D | |  |  |

We would like to nd 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 de nition of the empirical risk

1. A Formal Learning Model

remains the same as before, namely,

def jfi 2 [m] : h(xi) 6= yigj

LS(h) = :

m

Given S, a learner can compute LS(h) for any function h : X ! f0; 1g. Note

that LS(h) = LD(uniform over S)(h).

The Goal

We wish to nd some hypothesis, h : X ! Y, that (probably approximately)

minimizes the true risk, LD(h).

The Bayes Optimal Predictor.

Given any probability distribution D over X f0; 1g, the best label predicting function from X to f0; 1g will be

(

1 if P[y = 1jx] 1=2

fD(x) =

0 otherwise

It is easy to verify (see Exercise [7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page52)) that for every probability distribution D, the Bayes optimal predictor fD is optimal, in the sense that no other classi er,

1. : X ! f0; 1g has a lower error. That is, for every classi er g, LD(fD) LD(g). Unfortunately, since we do not know D, we cannot utilize this optimal predictor

fD. What the learner does have access to is the training sample. We can now present the formal de nition of agnostic PAC learnability, which is a natural extension of the de nition of PAC learnability to the more realistic, nonrealizable, learning setup we have just discussed.

Clearly, we cannot hope that the learning algorithm will nd a hypothesis whose error is smaller than the minimal possible error, that of the Bayes predic-tor.

Furthermore, as we shall prove later, once we make no prior assumptions about the data-generating distribution, no algorithm can be guaranteed to nd a predictor that is as good as the Bayes optimal one. Instead, we require that the learning algorithm will nd 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 mH : (0; 1)2 ! N and a learning algorithm with the following property: For every ; 2 (0; 1) and for every distribution D over X Y, when running the learning algorithm on m mH( ; ) i.i.d. examples generated by D, the algorithm returns a hypothesis h such that, with probability of at least 1 (over the choice of the m training examples),

LD(h) min LD(h0) + :

h02H

|  |  |
| --- | --- |
| 3.2 A More General Learning Model | 47 |
|  |  |

Clearly, if the realizability assumption holds, agnostic PAC learning provides the same guarantee as PAC learning. In that sense, agnostic PAC learning gener-alizes the de nition of PAC learning. When the realizability assumption does not hold, no learner can guarantee an arbitrarily small error. Nevertheless, under the de nition 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 di erent learning tasks.

Multiclass Classi cation Our classi cation does not have to be binary. Take, for example, the task of document classi cation: 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 classi ed documents and, on the basis of these examples, should output a program that can take as input a new document and output a topic classi cation 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 di erent 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 nite 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 nite sequence of (feature vector; label) pairs, the learner's output will be a function from the domain set to the label set, and, nally, for our measure of success, we can use the probability, over (document, topic) pairs, of the event that our predictor suggests a wrong label.

Regression In this task, one wishes to nd some simple pattern in the data { a functional relationship between the X and Y components of the data. For example, one wishes to nd 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 nite sequence of (x; y) pairs, and a function from X to Y respectively). However, our measure of success is

1. A Formal Learning Model

di erent. We may evaluate the quality of a hypothesis function, h : X ! Y, by the expected square di erence between the true labels and their predicted values, namely,

|  |  |  |  |
| --- | --- | --- | --- |
| def | | (h(x) y)2: | (3.2) |
| LD(h) = | (x;yE) |
|  | D |  |  |

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 ` be any function from H Z to the set of nonnegative real numbers,

1. : H Z ! R+.

We call such functions loss functions.

Note that for prediction problems, we have that Z = X 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page307), Z is not a product of an instance domain and a label domain).

We now de ne the risk function to be the expected loss of a classi er, h 2 H, with respect to a probability distribution D over Z, namely,

|  |  |  |  |
| --- | --- | --- | --- |
| def | | [`(h; z)]: | (3.3) |
| LD(h) = | E |
|  | z D |  |  |

That is, we consider the expectation of the loss of h over objects z picked ran-domly according to D. Similarly, we de ne the empirical risk to be the expected loss over a given sample S = (z1; : : : ; zm) 2 Zm, namely,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| def | 1 |  | m |  |
|  |  |  | Xi |  |
| LS(h) = | m | | `(h; zi): | (3.4) |
|  |  |  | =1 |  |

The loss functions used in the preceding examples of classi cation and regres-sion tasks are as follows:

0{1 loss: Here, our random variable z ranges over the set of pairs X Y and the loss function is

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| `0 1 | (h; (x; y)) = | (1 | if | h(x) = y |
|  | def | 0 | if | h(x) = y |
|  |  |  |  | 6 |

This loss function is used in binary or multiclass classi cation problems.

One should note that, for a random variable, , taking the values f0; 1g,

E D[ ] = P D[ = 1]. Consequently, for this loss function, the de ni-

tions of LD(h) given in Equation ([3.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page48)) and Equation ([3.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page45)) coincide.

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

def 2

`sq(h; (x; y)) = (h(x) y) :

|  |  |
| --- | --- |
| 3.3 Summary | 49 |
|  |  |

This loss function is used in regression problems.

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

To summarize, we formally de ne 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+, if there exist a function mH : | | (0; 1)2 ! N |

and a learning algorithm with the following property: For every ; 2 (0; 1) and for every distribution D over Z, when running the learning algorithm on

1. mH( ; ) i.i.d. examples generated by D, the algorithm returns h 2 H such that, with probability of at least 1 (over the choice of the m training examples),

LD(h) min LD(h0) + ;

h02H

where LD(h) = Ez D[`(h; z)].

Remark 3.1 (A Note About Measurability\*) In the aforementioned de nition, for every h 2 H, we view the function `(h; ) : Z ! R+ as a random variable and de ne LD(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 -algebra of subsets of Z, over which the probability D is de ned, and that the preimage of every initial segment in R+ is in this -algebra. In the speci c case of binary classi cation with the 0 1 loss, the -algebra is over X f0; 1g and our assumption on ` is equivalent to the assumption that for every h, the set f(x; h(x)) : x 2 X g is in the -algebra.

Remark 3.2 (Proper versus Representation-Independent Learning\*) In the pre-ceding de nition, we required that the algorithm will return a hypothesis from H. In some situations, H is a subset of a set H0, and the loss function can be naturally extended to be a function from H0 Z to the reals. In this case, we may allow the algorithm to return a hypothesis h0 2 H0, as long as it satis es the requirement LD(h0) minh2H LD(h) + . Allowing the algorithm to output a hypothesis from H0 is called representation independent learning, while proper learning occurs when the algorithm must output a hypothesis from H. Represen-tation independent learning is sometimes called \improper learning," although there is nothing improper in representation independent learning.

3.3 Summary

In this chapter we de ned our main formal learning model { PAC learning. The basic model relies on the realizability assumption, while the agnostic variant does

1. A Formal Learning Model

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" pre x 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page83) we will discuss other notions of learnability.

3.4 Bibliographic Remarks

Our most general de nition 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 de nition requires that the sample complexity will be polynomial in 1= and in 1= , as well as in the representation size of hypotheses in the class (see also Kearns & Vazirani (1994)). As we will see in Chapter [6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page67), if a problem is at all PAC learnable then the sample complexity depends polynomially on 1= and log(1= ). Valiant's de nition 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page100), 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 classi cation task. Suppose that H is PAC learnable and its sample complexity is given by mH( ; ). Show that mH is monotonically nonincreasing in each of its parameters. That is, show that given 2 (0; 1), and given 0 < 1 2 < 1, we have that mH( 1; ) mH( 2; ). Similarly, show that given 2 (0; 1), and given 0 < 1 2 < 1, we have that mH( ; 1) mH( ; 2).
2. Let X be a discrete domain, and let HSingleton = fhz : z 2 X g [ fh g, where for each z 2 X , hz is the function de ned by hz(x) = 1 if x = z and hz(x) = 0 if x 6= z. h is simply the all-negative hypothesis, namely, 8x 2 X, h (x) = 0. The realizability assumption here implies that the true hypothesis f labels negatively all examples in the domain, perhaps except one.

|  |  |
| --- | --- |
| 3.5 Exercises | 51 |
|  |  |

* 1. Describe an algorithm that implements the ERM rule for learning HSingleton in the realizable setup.
  2. Show that HSingleton is PAC learnable. Provide an upper bound on the sample complexity.

1. Let X = R2, Y = f0; 1g, and let H be the class of concentric circles in the plane, that is, H = fhr : r 2 R+g, where hr(x) = 1[kxk r]. Prove that H is PAC learnable (assume realizability), and its sample complexity is bounded by

|  |  |  |
| --- | --- | --- |
| mH( ; ) |  | : |
|  | log(1= ) |  |
|  |  |  |

1. In this question, we study the hypothesis class of Boolean conjunctions de ned

as follows. The instance space is X = f0; 1gd and the label set is Y = f0; 1g. A literal over the variables x1; : : : ; xd is a simple Boolean function that takes the form f(x) = xi, for some i 2 [d], or f(x) = 1 xi for some i 2 [d]. We use the notation xi as a shorthand for 1 xi. A conjunction is any product of literals. In Boolean logic, the product is denoted using the ^ sign. For example, the function h(x) = x1 (1 x2) is written as x1 ^ x2.

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 x1 ^x1 (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) 2 X Y consists of an assignment to the d Boolean variables x1; : : : ; xd, and its truth value (0 for false and 1 for true).

For instance, let d = 3 and suppose that the true conjunction is x1 ^ x2. 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 m.

1. Let X be a domain and let D1; D2; : : : ; Dm be a sequence of distributions over X . Let H be a nite class of binary classi ers over X and let f 2 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 Di and then yi is set to be f(xi). Let Dm denote the average, that is, | |
|  | + + Dm)=m. |
| Dm = (D1 |

Fix an accuracy parameter 2 (0; 1). Show that

P 9h 2 H s.t. L (h) > and L(S;f)(h) = 0 jHje m:

(Dm;f)

1. A Formal Learning Model

Hint: Use the geometric-arithmetic mean inequality.

1. Let H be a hypothesis class of binary classi ers. Show that if H is agnostic PAC learnable, then H is PAC learnable as well. Furthermore, if A is a suc-cessful agnostic PAC learner for H, then A is also a successful PAC learner for H.
2. (\*) The Bayes optimal predictor: Show that for every probability distri-bution D, the Bayes optimal predictor fD is optimal, in the sense that for every classi er g from X to f0; 1g, LD(fD) LD(g).
3. (\*) We say that a learning algorithm A is better than B with respect to some probability distribution, D, if

LD(A(S)) LD(B(S))

for all samples S 2 (X f0; 1g)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 f0; 1g.

1. A probabilistic label predictor is a function that assigns to every domain

point x a probability value, h(x) 2 [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 i the coin comes up Heads. Formally, we de ne a probabilistic label

predictor as a function, h : X ! [0; 1]. The loss of such h on an example (x; y) is de ned to be jh(x) yj, 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 f0; 1g, then jh(x) yj = 1[h(x)6=y].

Prove that for every data-generating distribution D over X f0; 1g, the Bayes optimal predictor has the smallest risk (w.r.t. the loss function `(h; (x; y)) = jh(x) yj, among all possible label predictors, including prob-abilistic ones).

* 1. Let X be a domain and f0; 1g be a set of labels. Prove that for every distribution D over X f0; 1g, there exist a learning algorithm AD that is better than any other learning algorithm with respect to D.
  2. Prove that for every learning algorithm A there exist a probability distri-

bution, D, and a learning algorithm B such that A is not better than B w.r.t. D.

1. Consider a variant of the PAC model in which there are two example ora-cles: 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 ! f0; 1g, let D+ be the distribution over

X + = fx 2 X : f(x) = 1g de ned by D+(A) = D(A)=D(X +), for every A X +. Similarly, D is the distribution over X induced by D.

The de nition of PAC learnability in the two-oracle model is the same as the standard de nition of PAC learnability except that here the learner has access to m+H( ; ) i.i.d. examples from D+ and m ( ; ) i.i.d. examples from D . The learner's goal is to output h s.t. with probability at least 1 (over the choice

|  |  |
| --- | --- |
| 3.5 Exercises | 53 |
|  |  |

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

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. (\*\*) De ne h+ to be the always-plus hypothesis and h to be the always-minus hypothesis. Assume that h+; h 2 H. Show that if H is PAC learn-able in the two-oracle model, then H is PAC learnable in the standard one-oracle model.

1. Learning via Uniform Convergence

The rst formal learning model that we have discussed was the PAC model. In Chapter [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page33) we have shown that under the realizability assumption, any nite hypothesis class is PAC learnable. In this chapter we will develop a general tool, uniform convergence, and apply it to show that any nite 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 Su cient 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 su ces 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 ( -representative sample) A training set S is called -representative (w.r.t. domain Z, hypothesis class H, loss function `, and distribution D) if

8h 2 H; jLS(h) LD(h)j :

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

lemma 4.2 Assume that a training set S is 2 -representative (w.r.t. domain Z, hypothesis class H, loss function `, and distribution D). Then, any output of ERMH(S), namely, any hS 2 argminh2H LS(h), satis es

LD(hS) min LD(h) + :

h2H

|  |  |
| --- | --- |
| 4.2 Finite Classes Are Agnostic PAC Learnable | 55 |
|  |  |

Proof For every h 2 H,

LD(hS) LS(hS) + 2 LS(h) + 2 LD(h) + 2 + 2 = LD(h) + ;

where the rst and third inequalities are due to the assumption that S is 2 - representative (De nition [4.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54)) and the second inequality holds since hS is an ERM predictor. 

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

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

Similar to the de nition of sample complexity for PAC learning, the function mUCH measures the (minimal) sample complexity of obtaining the uniform con-vergence property, namely, how many examples we need to ensure that with probability of at least 1 the sample would be -representative.

The term uniform here refers to having a xed 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54) and the de nition of uniform convergence.

corollary 4.4 If a class H has the uniform convergence property with a function mUCH then the class is agnostically PAC learnable with the sample com-plexity mH( ; ) mUCH( =2; ). Furthermore, in that case, the ERMH paradigm is a successful agnostic PAC learner for H.

4.2 Finite Classes Are Agnostic PAC Learnable

In view of Corollary [4.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page55), the claim that every nite hypothesis class is agnostic PAC learnable will follow once we establish that uniform convergence holds for a nite hypothesis class.

To show that uniform convergence holds we follow a two step argument, similar to the derivation in Chapter [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page33). The rst step applies the union bound while the second step employs a measure concentration inequality. We now explain these two steps in detail.

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

1. Learning via Uniform Convergence

i.i.d. from D we have that for all h 2 H, jLS(h) LD(h)j . That is,

Dm(fS : 8h 2 H; jLS(h) LD(h)j g) 1 :

Equivalently, we need to show that

Dm(fS : 9h 2 H; jLS(h) LD(h)j > g) < :

Writing

fS : 9h 2 H; jLS(h) LD(h)j > g = [h2HfS : jLS(h) LD(h)j > g; and applying the union bound (Lemma [2.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page39)) we obtain

X

Dm(fS : 9h 2 H; jLS(h) LD(h)j > g) Dm(fS : jLS(h) LD(h)j > g):

h2H

(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 su ciently large m). That is, we will

show that for any xed hypothesis, h, (which is chosen in advance prior to the

sampling of the training set), the gap between the true and empirical risks,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| jLS(h) LD(h)j, is likely to be small. | | | 1 | m |  |
| Recall that L (h) = Ez | D | [`(h; z)] and that LS(h) = |  | `(h; zi). Since |
| m | i=1 |
| D |  |  |
| each zi is sampled i.i.d. from D, the expected value of | | | the random variable | | |
|  | P |  |

`(h; zi) is LD(h). By the linearity of expectation, it follows that LD(h) is also the expected value of LS(h). Hence, the quantity jLD(h) LS(h)j 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 in nity, 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, nite, sample size.

Instead, we will use a measure concentration inequality due to Hoe ding, which quanti es the gap between empirical averages and their expected value.

lemma 4.5 (Hoe ding's Inequality) Let 1; : : : ; m be a sequence of i.i.d. ran-dom variables and assume that for all i, E[ i] = and P[a i b] = 1. Then, for any > 0

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P " m1 | | | m | i> # | 2 exp 2 m 2=(b a)2 | : |
|  | | |  |  |  |
| i=1 |
|  | | | X |  |
|  |  |  |
|  |  |  |  |  |  |  |
|  | | |  |  |  |  |

The proof can be found in Appendix [B](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page422).

Getting back to our problem, let i be the random variable `(h; zi). Since h is xed and z1; : : : ; zm are sampled i.i.d., it follows that 1; : : : ; m are also i.i.d.

random variables. Furthermore, LS(h) = 1 Pm i and LD(h) = . Let us

m i=1

|  |  |
| --- | --- |
| 4.2 Finite Classes Are Agnostic PAC Learnable | 57 |
|  |  |

further assume that the range of ` is [0; 1] and therefore i 2 [0; 1]. We therefore obtain that

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Dm(fS : jLS(h) LD(h)j > g) = P " m1 | | m | i> # | 2 exp | 2 m 2 : | |
|  | |  |  |  |  |
| i=1 |  |
|  |  | X |  |  |
|  |  |  |  |  |
|  | |  |  |  | (4.2) | |
| Combining this with Equation ([4.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page56)) yields | | |  |  |  |  |
|  |  |  | X |  |
| Dm(fS : 9h 2 H; jLS(h) LD(h)j > g) | | | 2 exp 2 m 2 | |
|  | | | h2H | |
| = | | | 2 jHj exp 2 m 2 : | |

Finally, if we choose

1. log(2jHj= ) 2 2

then

Dm(fS : 9h 2 H; jLS(h) LD(h)j > g) :

corollary 4.6 Let H be a nite hypothesis class, let Z be a domain, and let

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

|  |  |  |  |
| --- | --- | --- | --- |
| H |  | 2 2 |  |
| mUC( ; ) |  | log(2jHj= ) | : |
|  |  |

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | 2 log(2 | = ) | |
| mH( ; ) mHUC( =2; ) | jHj |  | : |
| 2 |  |

Remark 4.1 (The \Discretization Trick") While the preceding corollary only applies to nite hypothesis classes, there is a simple trick that allows us to get a very good estimate of the practical sample complexity of in nite hypothesis classes. Consider a hypothesis class that is parameterized by d parameters. For example, let X = R, Y = f 1g, and the hypothesis class, H, be all functions of the form h (x) = sign(x ). That is, each hypothesis is parameterized by one parameter, 2 R, and the hypothesis outputs 1 for all instances larger than

and outputs 1 for instances smaller than . This is a hypothesis class of an in nite size. However, if we are going to learn this hypothesis class in practice, using a computer, we will probably maintain real numbers using oating 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

oating 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page57) we obtain that the sample complexity of such

1. Learning via Uniform Convergence

128d+2 log(2= )

classes is bounded by 2 . This upper bound on the sample complex-ity has the de ciency of being dependent on the speci c representation of real numbers used by our machine. In Chapter [6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page67) we will introduce a rigorous way to analyze the sample complexity of in nite size hypothesis classes. Neverthe-less, 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 su ces for agnostic PAC learnability using the ERM rule. We have shown that nite 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 rst uniform convergence result in the 1930s. See (Dudley, Gine & Zinn 1991). The relation between uniform con-vergence and learnability was thoroughly studied by Vapnik { see (Vapnik 1992, Vapnik 1995, Vapnik 1998). In fact, as we will see later in Chapter [6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page67), the funda-mental theorem of learning theory states that in binary classi cation problems, uniform convergence is not only a su cient 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 ( ; ) requirement on the convergence of errors in our de nitions of PAC learning, is, in fact, quite close to a sim-pler 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 ; > 0, there exists m( ; ) such that 8m m( ; )

P [LD(A(S)) > ] <

S Dm

2.

|  |  |  |  |
| --- | --- | --- | --- |
| lim E | [L | D | (A(S))] = 0 |
| m!1 S Dm |  |  |

|  |  |
| --- | --- |
| 4.5 Exercises | 59 |
|  |  |

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

1. Bounded loss functions: In Corollary [4.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page57) 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 satis es

mH( ; ) mUC( =2; ) 2 log(2jHj= )(b a)2 :

H 2

1. The Bias-Complexity Tradeo

In Chapter [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page33) we saw that unless one is careful, the training data can mislead the learner, and result in over tting. To overcome this problem, we restricted the search space to some hypothesis class H. Such a hypothesis class can be viewed as re ecting some prior knowledge that the learner has about the task { a belief that one of the members of the class 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 speci c learning task is de ned by an unknown distribution D over X Y, where the goal of the learner is to nd a predictor h : X ! Y, whose risk, LD(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 D, if A receives m i.i.d. examples from D, there is a high chance it outputs a predictor h that has a low risk.

The rst 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 classi cation 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, de ned by some distribution D, we should have some prior knowledge on D. One type of such prior knowledge is that D comes from some speci c parametric family of distributions. We will study learning under such assumptions later on in Chapter [24](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page342). Another type of prior knowledge on D, which we assumed when de ning the PAC learning model, is that there exists h in some prede ned hypothesis class H, such that LD(h) = 0. A softer type of prior knowledge on D is assuming that minh2H LD(h) is small. In a sense, this weaker assumption on D is a prerequisite for using the

|  |  |
| --- | --- |
| 5.1 The No-Free-Lunch Theorem | 61 |
|  |  |

agnostic PAC model, in which we require that the risk of the output hypothesis will not be much larger than minh2H LD(h).

In the second part of this chapter we study the bene ts 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 rst component re ects the quality of our prior knowledge, measured by the minimal risk of a hypothesis in our hypothesis class, minh2H LD(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 over tting, which depends on the size or the complexity of the class H and is called the estimation error. These two terms imply a tradeo between choosing a more complex H (which can decrease the bias but increases the risk of over tting) or a less complex H (which might increase the bias but decreases the potential over tting).

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 classi cation with respect to the 0 1 loss over a domain X . Let m be any number smaller than jX j=2, representing a training set size. Then, there exists a distribution D over X f0; 1g such that:

1. There exists a function f : X ! f0; 1g with LD(f) = 0.
2. With probability of at least 1=7 over the choice of S Dm we have that LD(A(S)) 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 hypoth-esis class H = ffg, or more generally, ERM with respect to any nite hypothesis class that contains f and whose size satis es the equation m 8 log(7jHj=6) (see Corollary [2.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page40)).

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 = 22m possible functions from C to f0; 1g. Denote these functions by f1; : : : ; fT . For each such function, let Di be a distribution over

1. The Bias-Complexity Tradeo C f0; 1g de ned by

(

1=jCj if y = fi(x)

Di(f(x; y)g) =

0 otherwise:

That is, the probability to choose a pair (x; y) is 1=jC j if the label y is indeed the true label according to fi, and the probability is 0 if y 6= fi(x). Clearly,

LDi (fi) = 0.

We will show that for every algorithm, A, that receives a training set of m examples from C f0; 1g and returns a function A(S) : C ! f0; 1g, it holds that

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| max | | | S | E | m[LDi (A(S))] 1=4: | (5.1) |
| i | 2 | [T ] |  |
|  |  |  | Di | |  |

Clearly, this means that for every algorithm, A0, that receives a training set of m examples from X f0; 1g there exist a function f : X ! f0; 1g and a distribution

1. over X f0; 1g, such that LD(f) = 0 and

|  |  |  |  |
| --- | --- | --- | --- |
| S | E | m[LD(A0(S))] 1=4: | (5.2) |
|  | D |  |  |

It is easy to verify that the preceding su ces for showing that P[LD(A0(S)) 1=8] 1=7, which is what we need to prove (see Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page66)).

We now turn to proving that Equation ([5.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page62)) holds. There are k = (2m)m possible sequences of m examples from C. Denote these sequences by S1; : : : ; Sk. Also, if Sj = (x1; : : : ; xm) we denote by Sji the sequence containing the instances in Sj labeled by the function fi, namely, Sji = ((x1; fi(x1)); : : : ; (xm; fi(xm))). If the distribution is Di then the possible training sets A can receive are S1i; : : : ; Ski, and all these training sets have the same probability of being sampled. Therefore,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| E | [L i (A(S))] = | 1 | | k | L i (A(Si)): | | (5.3) |
|  |  | Xj |
| S Dim | D | k | | D | j |  |
| =1 |  |

Using the facts that \maximum" is larger than \average" and that \average" is larger than \minimum," we have

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 1 | k |  | 1 | | | T |
|  | X | i |  |  |  | Xi |
| max |  |  |  |  |  |  |
| k |  | LDi (A(Sj)) T | | | |  |
| i [T ] | j=1 | =1 |
|  |  |  |  |  |  |
|  |  |  |  | 1 | |  | k |
|  |  |  |  | = |  |  |  |
|  |  |  |  | k | |  |

X

j=1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | | | k |  |
|  |  |  | Xj | (A(Si)) |
|  | k | | L |
|  | Di | j |
|  |  |  | =1 |  |
| 1 | |  | T |  |
|  |  |  | Xi | (A(Si)) |
| T | | | L |
| Di | j |
|  |  |  | =1 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 2 | 1 | | T |  |
|  |  |  | Xi |  |
|  |  |  |  | i |  |
| min |  |  | LDi (A(Sj)): | (5.4) |
| T | |
| j [k] |  |
|  |  |  |  | =1 |  |

Next, x some j 2 [k]. Denote Sj = (x1; : : : ; xm) and let v1; : : : ; vp be the examples in C that do not appear in Sj. Clearly, p m. Therefore, for every

|  |  |
| --- | --- |
| 5.1 The No-Free-Lunch Theorem | 63 |
|  |  |

function h : C ! f0; 1g and every i we have

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 1 | | | |  | X | | |  |  |  |
|  |  |  |  |  |  |  |  |  | 6 |  |
|  |  |  |  | 2m x2C | | | | | | |  |  |
|  |  |  | LDi (h) = |  |  |  |  | p |  |  | 1[h(x)=fi(x)] | |  |
|  |  |  | 1 | | | |  |  |  |  |  |  |
|  |  |  |  | Xr | | | 1[h(vr)6=fi(vr)] | |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  | 2m | | | | =1 | |  |  |
|  |  |  | 1 | | | |  | p |  |  |  |  |  |
|  |  |  |  |  |  |  | Xr | | | |  |  |  |
|  |  |  | 2p | | | |  |  |  |
|  |  |  | =1 | | 1[h(vr)6=fi(vr)]: | | | | (5.5) |
| Hence, | |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | | T | 1 | | |  |  | T |  |  | 1 | p |  |
|  |  | X | LDi (A(Sji)) | |  |  | Xi | |  |  |  | X |  |
|  |  |  |  |  |  |  |  |  | | 1[A(Sji)(vr)6=fi(vr)] |  |
|  | T | i=1 | T | =1 | |  | 2p | |  |
|  |  |  |  |  |  |  |  |  | r=1 |  |

* + 1. T

1 X 1 X

1. 2p r=1 T i=1 1[A(Sji)(vr)6=fi(vr)]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 |  | 2 | 1 | Xi |  |  |  |
|  |  | min | T | 1 | i | (5.6) |
| 2 | T |  |
| r [p] | =1 | [A(Sj)(vr)6=fi(vr)]: | |  |
|  |  |  |  |  |  |  |  |

Next, x some r 2 [p]. We can partition all the functions in f1; : : : ; fT into T =2 disjoint pairs, where for a pair (fi; fi0) we have that for every c 2 C, fi(c) 6= fi0(c) if and only if c = vr. Since for such a pair we must have Sji = Sji0, it follows that

1[A(Sji)(vr)6=fi(vr)] + 1[A(Sji0)(vr)6=fi0(vr)] = 1;

which yields

T

1 X

T

i=1

1

1[A(Sji)(vr)6=fi(vr)] = 2 :

Combining this with Equation ([5.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page63)), Equation ([5.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page62)), and Equation ([5.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page62)), we obtain that Equation ([5.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page62)) 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

1. to f0; 1g. 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 in nite domain set and let H be the set of all functions from X to f0; 1g. Then, H is not PAC learnable.

1. The Bias-Complexity Tradeo

Proof Assume, by way of contradiction, that the class is learnable. Choose some < 1=8 and < 1=7. By the de nition of PAC learnability, there must be some learning algorithm A and an integer m = m( ; ), such that for any

data-generating distribution over X f0; 1g, if for some function f : X ! f0; 1g, LD(f) = 0, then with probability greater than 1 when A is applied to samples S of size m, generated i.i.d. by D, LD(A(S)) . However, applying the No-Free-Lunch theorem, since jX j > 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 > , LD(A(S)) > 1=8 > , 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 speci c 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 tradeo is discussed in the following section.

5.2 Error Decomposition

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| LD(hS) = app + est | where : | app = | min L | D | (h); | est = LD(hS) app: (5.7) |
|  | h2H |  |

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 speci c 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page64)

1 In fact, it always includes the error of the Bayes optimal predictor (see Chapter [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page43)), 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

minh2H LD(h), but rather to the excess error over that of the Bayes optimal predictor, namely, minh2H LD(h) Bayes.

|  |  |
| --- | --- |
| 5.3 Summary | 65 |
|  |  |

The Estimation Error { the di erence 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 nite hypothesis class, est increases (logarithmically) with jHj and de-creases with m. We can think of the size of H as a measure of its complexity. In future chapters we will de ne other complexity measures of hypothesis classes.

Since our goal is to minimize the total risk, we face a tradeo , called the bias-complexity tradeo . 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 over tting. 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 under tting. Of course, a great choice for H is the class that contains only one classi er { the Bayes optimal classi er. But the Bayes optimal classi er 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 rea-sonable 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 clas-si er, we still have some prior knowledge of the speci c 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 speci ed 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 tradeo , 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

k 1 = 1

1 .

2k

2 2k

1. The Bias-Complexity Tradeo

be small. In the next chapter we will study in more detail the behavior of the estimation error. In Chapter [7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page83) we will discuss alternative ways to express prior knowledge.

5.4 Bibliographic Remarks

(Wolpert & Macready 1997) proved several no-free-lunch theorems for optimiza-tion, but these are rather di erent 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page62)) su ces for showing that P[LD(A(S)) 1=8] 1=7. Hint: Let be a random variable that receives values in [0; 1] and whose

expectation satis es E[ ] 1=4. Use Lemma [B.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page422) to show that P[ 1=8] 1=7.

1. Assume you are asked to design a learning algorithm to predict whether pa-tients are going to su er a heart attack. Relevant patient features the al-gorithm 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 rst 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 ve 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 a ect your choice.

3. Prove that if jX j km for a positive integer k 2, then we can replace

the lower bound of 1=4 in the No-Free-Lunch theorem with

Namely, let A be a learning algorithm for the task of binary classi cation. Let m be any number smaller than jX j=k, representing a training set size. Then, there exists a distribution D over X f0; 1g such that:

There exists a function f : X ! f0; 1g with LD(f) = 0. ES Dm [LD(A(S))] 12 21k .

1. The VC-Dimension

In the previous chapter, we decomposed the error of the ERMH rule into ap-proximation error and estimation error. The approximation error depends on the t of our prior knowledge (as re ected by the choice of the hypothesis class H) to the underlying unknown distribution. In contrast, the de nition of PAC learnability requires that the estimation error would be bounded uniformly over all distributions.

Our current goal is to gure 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 nite classes are learnable, but that the class of all functions (over an in nite size domain) is not. What makes one class learnable and the other unlearnable? Can in nite-size classes be learnable, and, if so, what determines their sample complexity?

We begin the chapter by showing that in nite classes can indeed be learn-able, and thus, niteness 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 classi cation with the zero-one loss. This characterization was rst discovered by Vladimir Vapnik and Alexey Chervonenkis in 1970 and relies on a combinatorial notion called the Vapnik-Chervonenkis dimension (VC-dimension). We formally de ne 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 In nite-Size Classes Can Be Learnable

In Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54) we saw that nite 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 rst present a simple example of an in nite-size hypothesis class that is learnable.

Example 6.1 Let H be the set of threshold functions over the real line, namely,

H = fha : a 2 Rg, where ha : R ! f0; 1g is a function such that ha(x) = 1[x<a].

To remind the reader, 1[x<a] is 1 if x < a and 0 otherwise. Clearly, H is of in nite

1. The VC-Dimension

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 de ned earlier. Then, H is PAC learnable, using the ERM rule, with sample complexity of mH( ; ) dlog(2= )= e.

Proof Let a? be a threshold such that the hypothesis h?(x) = 1[x<a?] achieves LD(h?) = 0. Let Dx be the marginal distribution over the domain X and let a0 < a? < a1 be such that

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | P | x[x 2 (a0; a?)] = x | | P | x[x 2 (a?; a1)] = : |
|  | D |  |  | D |  |
|  |  | mass |  |  | mass |
|  | |  |  |  |  |
| a0 | | a | ? |  | a1 |
|  |  |  |  |  |



(If Dx( 1; a?) we set a0 = 1 and similarly for a1). Given a training set S, let b0 = maxfx : (x; 1) 2 Sg and b1 = minfx : (x; 0) 2 Sg (if no example in S is positive we set b0 = 1 and if no example in S is negative we set b1 = 1). Let bS be a threshold corresponding to an ERM hypothesis, hS, which implies that bS 2 (b0; b1). Therefore, a su cient condition for LD(hS) is that both b0 a0 and b1 a1. In other words,

P

S Dm

[LD(hS) > ]

P

S Dm

[b0

< a0 \_ b1 > a1];

and using the union bound we can bound the preceding by

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | P | m[LD(hS) > ] | S | P | m[b0 < a0] + | S | P | m[b1 > a1]: | (6.1) |
|  | D |  |  | D |  |  | D |  |  |

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | P | [b | 0 | < a ] = | S | P | [ (x; y) | 2 | S; x (a | ; a?)] = (1 |  | )m |  | e m: |
| m | 0 | m 8 | 62 0 |  |  |  |
|  | D |  |  |  |  | D |  |  |  |  |  |  |  |  |

Since we assume m > log(2= )= it follows that the equation is at most =2. In the same way it is easy to see that PS Dm [b1 > a1] =2. Combining with Equation ([6.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page68)) we conclude our proof. 

6.2 The VC-Dimension

We see, therefore, that while niteness of H is a su cient condition for learn-ability, 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 learn-ability. To motivate the de nition of the VC-dimension, let us recall the No-Free-Lunch theorem (Theorem [5.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page61)) and its proof. There, we have shown that without

|  |  |
| --- | --- |
| 6.2 The VC-Dimension | 69 |
|  |  |

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 distri-bution. To do so, the adversary used a nite set C 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 f0; 1g. 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 f0; 1g.

When considering PAC learnability of a hypothesis class H, the adversary is restricted to constructing distributions for which some hypothesis h 2 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 de nition.

definition 6.2 (Restriction of H to C) Let H be a class of functions from X to f0; 1g and let C = fc1; : : : ; cmg X . The restriction of H to C is the set of functions from C to f0; 1g that can be derived from H. That is,

HC = f(h(c1); : : : ; h(cm)) : h 2 Hg;

where we represent each function from C to f0; 1g as a vector in f0; 1gjCj.

If the restriction of H to C is the set of all functions from C to f0; 1g, then we say that H shatters the set C. Formally:

definition 6.3 (Shattering) A hypothesis class H shatters a nite set C X if the restriction of H to C is the set of all functions from C to f0; 1g. That is, jHC j = 2jCj.

Example 6.2 Let H be the class of threshold functions over R. Take a set

1. = fc1g. Now, if we take a = c1 + 1, then we have ha(c1) = 1, and if we take a = c1 1, then we have ha(c1) = 0. Therefore, HC is the set of all functions from C to f0; 1g, and H shatters C. Now take a set C = fc1; c2g, where c1 c2. No h 2 H can account for the labeling (0; 1), because any threshold that assigns the label 0 to c1 must assign the label 0 to c2 as well. Therefore not all functions from C to f0; 1g are included in HC ; hence C is not shattered by H.

Getting back to the construction of an adversarial distribution as in the proof of the No-Free-Lunch theorem (Theorem [5.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page61)), 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 f0; 1g, while still maintaining the realizability assumption. This immediately yields:

corollary 6.4 Let H be a hypothesis class of functions from X to f0; 1g. Let m be a training set size. Assume that there exists a set C X of size 2m that is shattered by H. Then, for any learning algorithm, A, there exist a distribution D over X f0; 1g and a predictor h 2 H such that LD(h) = 0 but with probability of at least 1=7 over the choice of S Dm we have that LD(A(S)) 1=8.

1. The VC-Dimension

Corollary [6.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page69) 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 de nition 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 in nite VC-dimension.

A direct consequence of Corollary [6.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page69) is therefore:

theorem 6.6 Let H be a class of in nite VC-dimension. Then, H is not PAC learnable.

Proof Since H has an in nite VC-dimension, for any training set size m, there exists a shattered set of size 2m, and the claim follows by Corollary [6.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page69). 

We shall see later in this chapter that the converse is also true: A nite VC-dimension guarantees learnability. Hence, the VC-dimension characterizes PAC learnability. But before delving into more theory, we rst 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 be the class of threshold functions over R. Recall Example [6.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page69), where we have shown that for an arbitrary set C = fc1g, H shatters C; therefore VCdim(H) 1. We have also shown that for an arbitrary set C = fc1; c2g where c1 c2, H does not shatter C. We therefore conclude that VCdim(H) = 1.

|  |  |
| --- | --- |
| 6.3 Examples | 71 |
|  |  |

6.3.2 Intervals

Let H be the class of intervals over R, namely, H = fha;b : a; b 2 R; a < bg,

where ha;b : R ! f0; 1g is a function such that ha;b(x) = 1[x2(a;b)]. Take the set C = f1; 2g. Then, H shatters C (make sure you understand why) and therefore VCdim(H) 2. Now take an arbitrary set C = fc1; c2; c3g 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 = fh(a1;a2;b1;b2) : a1 | | | a2 | and b1 b2g |  |  |
| where |  |  |  |  |  |  |
| h(a1;a2;b1;b2)(x1; x2) = ( | 1 | if a1 | x1 | a2 and b1 x2 | b2 | (6.2) |
| 0 | otherwise | |  |  |

We shall show in the following that VCdim(H) = 4. To prove this we need to nd 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page71)). Now, consider any set C R2 of 5 points. In C, take a leftmost point (whose rst coordinate is the smallest in C), a rightmost point ( rst 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 = fc1; : : : ; c5g and let c5 be the point that was not selected. Now, de ne 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 de ned by the selected points. So, C is not shattered by H, and therefore VCdim(H) = 4.

c1



c4 c5 c2



c3



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.

1. The VC-Dimension

6.3.4 Finite Classes

Let H be a nite class. Then, clearly, for any set C we have jHC j jHj and thus C cannot be shattered if jHj < 2jCj. This implies that VCdim(H) log2(jHj). This shows that the PAC learnability of nite classes follows from the more general statement of PAC learnability of classes with nite VC-dimension, which we shall see in the next section. Note, however, that the VC-dimension of a nite class

1. can be signi cantly smaller than log2(jHj). For example, let X = f1; : : : ; kg, for some integer k, and consider the class of threshold functions (as de ned in

Example [6.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page69)). Then, jHj = k but VCdim(H) = 1. Since k can be arbitrarily large, the gap between log2(jHj) and VCdim(H) 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 de ning 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 = fh : 2 Rg where h : X ! f0; 1g is de ned by h (x) = d0:5 sin( x)e. It is possible to prove that VCdim(H) = 1, namely, for every d, one can nd d points that are shattered by H (see Exercise [8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page80)).

6.4 The Fundamental Theorem of PAC learning

We have already shown that a class of in nite 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 H be a hypothesis class of functions from a domain X to f0; 1g and let the loss function be the 0 1 loss. Then, the following are equivalent:

1. H has the uniform convergence property.
2. Any ERM rule is a successful agnostic PAC learner for H.
3. H is agnostic PAC learnable.
4. H is PAC learnable.
5. Any ERM rule is a successful PAC learner for H.
6. H has a nite VC-dimension.

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

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

theorem 6.8 (The Fundamental Theorem of Statistical Learning { Quantita-tive Version) Let H be a hypothesis class of functions from a domain X to f0; 1g and let the loss function be the 0 1 loss. Assume that VCdim(H) = d < 1. Then, there are absolute constants C1; C2 such that:

|  |  |
| --- | --- |
| 6.5 Proof of Theorem [6.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72) | 73 |
|  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | H has the uniform convergence property with sample complexity | | | | | | | | | | | | |
|  |  | C1 |  | d + log(1= ) | | | mHUC( ; ) | C2 | | d + log(1= ) | | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 2 |  |  |  | 2 | | |
| 2. | H is agnostic PAC learnable with sample complexity | | | | | | | | | | | | |
|  |  | C1 | d + log(1= ) | | | | mH( ; ) | C2 | |  | d + log(1= ) | | |
|  |  |  |  |  |  |  |  | |  |
|  |  | 2 |  |  |  | 2 | | |
| 3. | H is PAC learnable with sample complexity | | | | | | |  |  |  |  |  |  |
|  | C1 | d + log(1= ) | | | mH( ; ) C2 | | | | d log(1= ) + log(1= ) | | | | |
|  |  |  |  |  |  |  | | |
|  |  |  |  |  |  |  | | |

The proof of this theorem is given in Chapter [28](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page392).

Remark 6.3 We stated the fundamental theorem for binary classi cation 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page171), Exercise [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page181)). Furthermore, in some situations, the ERM rule fails but learnability is possible with other learning rules.

6.5 Proof of Theorem [6.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72)

We have already seen that 1 ! 2 in Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54). 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 di cult 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 in nite, when restricting it to a nite set C X , its \e ective" size, jHC j, is only O(jCjd). That is, the size of HC grows polynomially rather than exponentially with jCj. 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page73) later.

In Section [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54) we have shown that nite hypothesis classes enjoy the uniform convergence property. In Section [6.5.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page75) later we generalize this result and show that uniform convergence holds whenever the hypothesis class has a

\small e ective size." By \small e ective size" we mean classes for which jHC j grows polynomially with jCj.

6.5.1 Sauer's Lemma and the Growth Function

We de ned the notion of shattering, by considering the restriction of H to a nite set of instances. The growth function measures the maximal \e ective" size of H on a set of m examples. Formally:

1. The VC-Dimension

definition 6.9 (Growth Function) Let H be a hypothesis class. Then the growth function of H, denoted H : N ! N, is de ned as

|  |  |  |
| --- | --- | --- |
| H(m) = C X:jCj=m | | HC : |
| max |  |  |
|  |

In words, H (m) is the number of di erent functions from a set C of size m to f0; 1g that can be obtained by restricting H to C.

Obviously, if VCdim(H) = d then for any m d we have H(m) = 2m. In such cases, H induces all possible functions from C to f0; 1g. The following beau-tiful 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.

lemma 6.10 (Sauer-Shelah-Perles)

d < 1. Then, for all m, H(m) H(m) (em=d)d.

Let H be a hypothesis class with VCdim(H) Pd m . In particular, if m > d + 1 then

i=0 i

Proof of Sauer's Lemma \*

To prove the lemma it su ces to prove the following stronger claim: For any

1. = fc1; : : : ; cmg we have

|  |  |
| --- | --- |
| 8 H; jHC j jfB C : H shatters Bgj: | (6.3) |

The reason why Equation ([6.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page74)) is su cient to prove the lemma is that if VCdim(H) d then no set whose size is larger than d is shattered by H and therefore

d

X m

i=0

When m > d + 1 the right-hand side of the preceding is at most (em=d)d (see Lemma [A.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page420) in Appendix [A](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419)).

We are left with proving Equation ([6.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page74)) and we do it using an inductive argu-ment. For m = 1, no matter what H is, either both sides of Equation ([6.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page74)) equal 1 or both sides equal 2 (the empty set is always considered to be shattered by H). Assume Equation ([6.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page74)) holds for sets of size k < m and let us prove it for sets of size m. Fix H and C = fc1; : : : ; cmg. Denote C0 = fc2; : : : ; cmg and in addition, de ne the following two sets:

Y0 = f(y2; : : : ; ym) : (0; y2; : : : ; ym) 2 HC \_ (1; y2; : : : ; ym) 2 HC g;

and

Y1 = f(y2; : : : ; ym) : (0; y2; : : : ; ym) 2 HC ^ (1; y2; : : : ; ym) 2 HC g:

It is easy to verify that jHC j = jY0j + jY1j. Additionally, since Y0 = HC0, using the induction assumption (applied on H and C0) we have that

jY0j = jHC0j jfB C0 : H shatters Bgj = jfB C : c1 62B ^ H shatters Bgj:

|  |  |
| --- | --- |
| 6.5 Proof of Theorem [6.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72) | 75 |
|  |  |

Next, de ne H0 H to be

H0 = fh 2 H : 9h0 2 H s.t. (1 h0(c1); h0(c2); : : : ; h0(cm))

1. (h(c1); h(c2); : : : ; h(cm)g;

namely, H0 contains pairs of hypotheses that agree on C0 and di er on c1. Using this de nition, it is clear that if H0 shatters a set B C0 then it also shatters the set B [ fc1g and vice versa. Combining this with the fact that Y1 = HC00 and using the inductive assumption (now applied on H0 and C0) we obtain that

jY1j = jHC00j jfB C0 : H0 shatters Bgj = jfB C0 : H0 shatters B [ fc1ggj

= jfB C : c1 2 B ^ H0 shatters Bgj jfB C : c1 2 B ^ H shatters Bgj:

Overall, we have shown that

jHC j = jY0j + jY1j

jfB C : c1 62B ^ H shatters Bgj + jfB C : c1 2 B ^ H shatters Bgj

= jfB C : H shatters Bgj;

which concludes our proof.

6.5.2 Uniform Convergence for Classes of Small E ective Size

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

theorem 6.11 Let H be a class and let H be its growth function. Then, for every D and every 2 (0; 1), with probability of at least 1 over the choice of

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| S Dm we have | | p |  |  |  |  |  |
| jLD(h) LS(h)j | | p2m | | |  | : |
| 4 + | |  | log( H(2m)) | | | |  |
|  |  |  |  |  |  |  |  |

Before proving the theorem, let us rst conclude the proof of Theorem [6.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72).

Proof of Theorem [6.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72) It su ces to prove that if the VC-dimension is nite then

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| the uniform convergence property holds. We will prove that | | | | | | | | | |  |
| mHUC( ; ) 4 ( )2 | | log | ( )2 | |  | + |  | ( )2 | | : |
|  | 16d |  |  | 16d |  |  | 16 d log(2e=d) | | |  |
|  |  |  |  |  |  |  |  |  |  |  |

From Sauer's lemma we have that for m > d, H(2m) (2em=d)d. Combining this with Theorem [6.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page75) we obtain that with probability of at least 1 ,

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| jLS(h) LD(h)j 4 + p | | | | | | | | p2m | | |  | : |
|  |  |  |  |  |  |  |  | d log(2em=d) | | | |  |
| For simplicity assume that p |  |  |  | |  | |  | |  |  |  |  |
|  | |  | |  | |  | |  |  |  |  |
|  | |  | | 4; hence, | | | | | | |  |
| d log(2em=d) | | | |  |
| jLS(h) LD(h)j | | | | r | |  |  |  | m | | | : |
| 1 | | | |  |  | 2d log(2em=d) | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

1. The VC-Dimension

To ensure that the preceding is at most we need that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 2d log(m) | | | + | 2 d log(2e=d) | | | : |
|  |  |  |  |  |  |
|  | ( )2 | |  | ( )2 | |

Standard algebraic manipulations (see Lemma [A.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419) in Appendix [A](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419)) show that a su cient condition for the preceding to hold is that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m 4 ( )2 | | log | ( )2 | |  | + | ( )2 | : |
|  | 2d |  |  | 2d |  |  | 4 d log(2e=d) | |
|  |  |  |  |  |  |  |  |  |

Remark 6.4 The upper bound on mUCH we derived in the proof Theorem [6.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72) is not the tightest possible. A tighter analysis that yields the bounds given in Theorem [6.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72) can be found in Chapter [28](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page392).

Proof of Theorem [6.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page75) \*

We will start by showing that

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S Dm | h |  | j D |  |  | j | p |  |  |  |  |  |
| 2H |  | p2m | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| E | sup | | L (h) |  | LS(h) | 4 + | log( H(2m)) | | | | : | (6.4) |
|  |  |
|  |  |  |  |  |  |

Since the random variable suph2H jLD(h) LS(h)j is nonnegative, the proof of the theorem follows directly from the preceding using Markov's inequality (see Section [B.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page422)).

To bound the left-hand side of Equation ([6.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page76)) we rst note that for every

1. 2 H, we can rewrite LD(h) = ES0 Dm [LS0(h)], where S0 = z10; : : : ; zm0 is an additional i.i.d. sample. Therefore,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S Dm h2H j D |  | j | S Dm h2H | S0 | Dm | 0 |  |  |  |
| E sup L | (h) | LS(h) = | E sup |  | E | LS (h) | LS(h) | : |
|  |  |
|  |  |  |  |  |  |  |  |  |  |

A generalization of the triangle inequality yields

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S0 Dm | | 0 | (h) |  | LS(h)] | S0 Dm j | | 0 | (h) |  | j | ; |
|  | E | [LS |  |  | E | LS |  | LS(h) |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

and the fact that supermum of expectation is smaller than expectation of supre-mum yields

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sup E | j | LS0(h) |  | LS(h) | j | E sup | j | LS0 | (h) |  | LS(h) | : |
| h2H S0 Dm |  |  | S0 Dm h2H |  |  | j |  |

Formally, the previous two inequalities follow from Jensen's inequality. Combin-ing all we obtain

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S Dm | h2H | j |  | D |  | LS(h) | j | S;S0 Dm | | h2H | j | |  | 0 | |  |  | LS(h) | | j | |  |
| E | sup |  | L | (h) |  |  | E |  | sup |  |  | LS | |  | (h) |  |  |  | # |
|  |  |  |  |  |  |  |  | S;S0E | m "h | |  | m | | |  | m |  | i0 |  | | i |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | = |  | sup |  |  |  | (`(h; z ) | | |  |  | `(h; z )) | : |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | D |  | 2H |  |  |  |  | i=1 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

(6.5)

|  |  |
| --- | --- |
| 6.5 Proof of Theorem [6.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72) | 77 |
|  |  |

The expectation on the right-hand side is over a choice of two i.i.d. samples

1. = z1; : : : ; zm and S0 = z10; : : : ; zm0. Since all of these 2m vectors are chosen i.i.d., nothing will change if we replace the name of the random vector zi with the name of the random vector zi0. If we do it, instead of the term (`(h; zi0) `(h; zi)) in Equation ([6.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page76)) we will have the term (`(h; zi0) `(h; zi)). It follows that for every 2 f 1gm we have that Equation ([6.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page76)) equals

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S;S0E | m "h | m | |  | i | i0 | i | # |
| D | 2H |  |  | m |  |  |  |  |
|  |  | i=1 |  |  |  |
|  | sup | 1 | | X | (`(h; z ) | | `(h; z )) |  |
|  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Since this holds for every 2 f 1gm, it also holds if we sample each component of uniformly at random from the uniform distribution over f 1g, denoted U . Hence, Equation ([6.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page76)) also equals

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Um S;S0 m | | "h |  | m | |  | i | # |
|  | D |  | 2H |  |  | m |  |  |
|  |  |  | i=1 |  |
| E | E | sup | | 1 |  | X | i(`(h; z0) | `(h; zi)) ; |
|  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

and by the linearity of expectation it also equals

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S;S0E | m Um | "h |  | m |  |
| D |  |  | 2H |  | m |
|  |  | i=1 |
|  | E | sup | | 1 | X |

#

i(`(h; zi0) `(h; zi)) :

Next, x S and S0, and let C be the instances appearing in S and S0. Then, we can take the supremum only over h 2 HC . Therefore,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Um "h | | |  | m | | |  |  |  |  |  |  |  | i |  |  |  |  | # | |  |  |  |
|  |  |  | |  |  | 2H |  |  |  | m | | | | |  |  |  |  |  |  |  |  | |  |  |  |
|  |  |  |  |  |  |  | i=1 | | | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | 1 | | X | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | E | | sup | | |  |  |  | i(`(h; z0) | | | | | | `(h; zi)) | | | |  | |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 1 | | |  |  |  | m |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |
|  |  | = | | E | | |  | max | | | |  |  |  |  | | i(`(h; z0) | |  | | `(h; zi)) | | | : |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  | Um " | | | h | 2H | | C m | | | | i=1 | i |  |  |  | # |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | X |  |  |  |  |  |  |  |  |  |
| Fix some h |  | and denote | | | | | | | | | = | | 1 | |  | | m (`(h; z | | | ) |  |  | `(h; z | )). Since | E | [ ] = 0 |
|  | C | |  |  |  |  |  |  |  | h |  | m | | | | | i=1 i |  | i0 |  |  | | i | | h |
|  | 2 H |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | P |  |  |  |  |  |  |  |

and h is an average of independent variables, each of which takes values in [ 1; 1], we have by Hoe ding's inequality that for every > 0,

P[j hj > ] 2 exp 2 m 2 :

Applying the union bound over h 2 HC , we obtain that for any > 0,

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | h2HC | j | hj |  |  |  | jHC j |  |  | 2 m 2 |  |
|  | max |  |  | > |  | 2 |  | exp |  | : |

Finally, Lemma [A.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419) in Appendix [A](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419) tells us that the preceding implies

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2H |  | j |  |  | j |  | p |  |  |  |  |
|  |  |  |  | 2m | |  |
| E | max | |  |  |  |  |  | 4 + | log(jHC j) | | | : |
|  |  |  |  |  |
|  | h |  |  | p | |  |  |
| h | C |  |  |  |  |  |

Combining all with the de nition of H, we have shown that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S Dm | h2H | j |  | D |  |  |  | j |  | p |  |  |  |  |
|  |  |  | p2m | | |
| E | sup |  | L |  | (h) |  | LS(h) |  | 4 + |  | log( H(2m)) | | | : |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

1. The VC-Dimension

6.6 Summary

The fundamental theorem of learning theory characterizes PAC learnability of classes of binary classi ers 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 learn-able if and only if its VC-dimension is nite and speci es 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 de nition 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 de nition 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page78) 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 H0 H then VCdim(H0) VCdim(H):
2. Given some nite domain set, X , and a number k jX j, gure out the VC-dimension of each of the following classes (and prove your claims):
   1. H=Xk = fh 2 f0; 1gX : jfx : h(x) = 1gj = kg. That is, the set of all functions that assign the value 1 to exactly k elements of X .
3. [http://gilkalai.wordpress.com/2008/09/28/](http://gilkalai.wordpress.com/2008/09/28/extremal-combinatorics-iii-some-basic-theorems) [extremal-combinatorics-iii-some-basic-theorems](http://gilkalai.wordpress.com/2008/09/28/extremal-combinatorics-iii-some-basic-theorems)

|  |  |
| --- | --- |
| 6.8 Exercises | 79 |
|  |

2. Hat most k = fh 2 f0; 1gX : jfx : h(x) = 1gj k or jfx : h(x) = 0gj kg.

1. Let X be the Boolean hypercube f0; 1gn. For a set I f1; 2; : : : ; ng we de ne

a parity function hI as follows. On a binary vector x = (x1; x2; : : : ; xn) 2 f0; 1gn,

!

X

hI (x) =

xi

mod 2 :

i2I

(That is, hI computes parity of bits in I.) What is the VC-dimension of the class of all such parity functions, Hn-parity = fhI : I f1; 2; : : : ; ngg?

1. We proved Sauer's lemma by proving that for every class H of nite VC-dimension d, and every subset A of the domain,

d

X jAj

jHAj jfB A : H shatters Bgj :

i=0

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

1. VC-dimension of axis aligned rectangles in Rd: Let Hrecd be the class of axis aligned rectangles in Rd. We have already seen that VCdim(Hrec2) = 4. Prove that in general, VCdim(Hrecd) = 2d.
2. VC-dimension of Boolean conjunctions: Let Hcond be the class of Boolean

conjunctions over the variables x1; : : : ; xd (d 2). We already know that this class is nite and thus (agnostic) PAC learnable. In this question we calculate VCdim(Hcond).

* 1. Show that jHcondj 3d + 1.
  2. Conclude that VCdim(H) d log 3.
  3. Show that Hcond shatters the set of unit vectors fei : i dg.
  4. (\*\*) Show that VCdim(Hcond) d.

Hint: Assume by contradiction that there exists a set C = fc1; : : : ; cd+1g

that is shattered by Hcond. Let h1; : : : ; hd+1 be hypotheses in Hcond that satisfy

(

0 i = j

8i; j 2 [d + 1]; hi(cj) =

1 otherwise

For each i 2 [d + 1], hi (or more accurately, the conjunction that corre-sponds to hi) contains some literal `i which is false on ci and true on cj for each j 6= i. Use the Pigeonhole principle to show that there must be a pair i < j d + 1 such that `i and `j use the same xk and use that fact to derive a contradiction to the requirements from the conjunctions hi; hj.

1. Consider the class Hmcond of monotone Boolean conjunctions over f0; 1gd. Monotonicity here means that the conjunctions do not contain negations.

1. The VC-Dimension

As in Hcond, the empty conjunction is interpreted as the all-positive hy-pothesis. We augment Hmcond with the all-negative hypothesis h . Show that VCdim(Hmcond) = d.

1. We have shown that for a nite hypothesis class H, VCdim(H) blog(jHj)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 in nite while VCdim(H) = 1.
   2. Give an example of a nite hypothesis class H over the domain X = [0; 1], where VCdim(H) = blog2(jHj)c.
2. (\*) 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 de ne 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 de ne 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 = fx 7! sin(d x)e : 2 Rg

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

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 2 (0; 1), then for any natural number m, dsin(2m x)e = (1 xm), provided that 9k m s.t. xk = 1.

1. Let H be the class of signed intervals, that is, H = fha;b;s : a b; s 2 f 1; 1gg where

(

s if x 2 [a; b]

ha;b;s(x) =

s if x 2= [a; b]

1. Let H be a class of functions from X to f0; 1g.
   1. Prove that if VCdim(H) d, for any d, then for some probability distri-bution D over X f0; 1g, for every sample size, m,
   2. Prove that for every H that is PAC learnable, VCdim(H) < 1. (Note that this is the implication 3 ! 6 in Theorem [6.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72).)
2. VC of union: Let H1; : : : ; Hr be hypothesis classes over some xed domain set X . Let d = maxi VCdim(Hi) and assume for simplicity that d 3.

|  |  |
| --- | --- |
| 6.8 Exercises | 81 |
|  |

1. Prove that

VCdim ([ri=1Hi) 4d log(2d) + 2 log(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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419).

2. (\*) Prove that for r = 2 it holds that

VCdim (H1 [ H2) 2d + 1:

1. Dudley classes: In this question we discuss an algebraic framework for

de ning 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 de ne the corresponding function, POS (f)(x) = 1[f(x)>0]. For a class F of real valued functions we de ne a corresponding class of functions POS (F) = fPOS (f) : f 2 Fg. We say that a family, F, of real valued func-tions is linearly closed if for all f; g 2 F and r 2 R, (f + rg) 2 F (where addition and scalar multiplication of functions are de ned point wise, namely, for all x 2 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

n def

function g : R ! R and a family of functions F, let F +g = ff +g : f 2 Fg. 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 de ned 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. Consider the mapping x 7!(f1(x); : : : ; fd(x)) (from Rn to Rd). 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117)).
3. Show that each of the following classes can be represented as a Dudley class:
   1. The class HSn of halfspaces over Rn (see Chapter [9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117)).
   2. The class HHSn of all homogeneous halfspaces over Rn (see Chapter [9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117)).
   3. The class Bd of all functions de ned by (open) balls in Rd. Use the Dudley representation to gure out the VC-dimension of this class.
   4. Let Pnd denote the class of functions de ned by polynomial inequalities of degree d, namely,

Pnd = fhp : p is a polynomial of degree d in the variables x1; : : : ; xng;

1. The VC-Dimension

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 of its terms. For example, the degree of p(x) = 3x31x22 + 4x3x27 is 5).

1. Use the Dudley representation to gure out the VC-dimension of the class P1d { the class of all d-degree polynomials over R.
2. Prove that the class of all polynomial classi ers over R has in nite VC-dimension.
3. Use the Dudley representation to gure out the VC-dimension of the class Pnd (as a function of d and n).

1. Nonuniform Learnability

The notions of PAC learnability discussed so far in the book allow the sample sizes to depend on the accuracy and con dence parameters, but they are uniform with respect to the labeling rule and the underlying data distribution. Conse-quently, classes that are learnable in that respect are limited (they must have a nite VC-dimension, as stated by Theorem [6.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72)). 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 de nitions.

We begin this discussion by de ning a notion of \nonuniform learnability" that allows the sample size to depend on the hypothesis to which the learner is com-pared. 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 su cient condition for nonuniform learnability is that H is a countable union of hypothesis classes, each of which enjoys the uniform con-vergence property. These results will be proved in Section [7.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page85) by introducing a new learning paradigm, which is called Structural Risk Minimization (SRM). In Section [7.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page89) we specify the SRM paradigm for countable hypothesis classes, which yields the Minimum Description Length (MDL) paradigm. The MDL paradigm gives a formal justi cation to a philosophical principle of induction called Oc-cam's razor. Next, in Section [7.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page92) we introduce consistency as an even weaker notion of learnability. Finally, we discuss the signi cance and usefulness of the di erent notions of learnability.

7.1 Nonuniform Learnability

\Nonuniform learnability" allows the sample size to be nonuniform with respect to the di erent hypotheses with which the learner is competing. We say that a hypothesis h is ( ; )-competitive with another hypothesis h0 if, with probability higher than (1 ),

LD(h) LD(h0) + :

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

1. Nonuniform Learnability

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 con dence parameters. In nonuniform learnability, however, we allow the sample size to be of the form mH( ; ; 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 mNULH : (0; 1)2 H ! N such that, for every ; 2 (0; 1) and for every h 2 H, if m mNULH( ; ; h) then for every distribution D, with probability of at least 1 over the choice of S Dm, it holds that

LD(A(S)) LD(h) + :

At this point it might be useful to recall the de nition of agnostic PAC learn-ability (De nition [3.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page46)):

A hypothesis class H is agnostically PAC learnable if there exist a learning algo-rithm, A, and a function mH : (0; 1)2 ! N such that, for every ; 2 (0; 1) and for every distribution D, if m mH( ; ), then with probability of at least 1 over the choice of S Dm it holds that

LD(A(S)) min LD(h0) + :

h02H

Note that this implies that for every h 2 H

LD(A(S)) LD(h) + :

In both types of learnability, we require that the output hypothesis will be ( ; )-competitive with every other hypothesis in the class. But the di erence 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 learn-ability. 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 classi ers is agnostic PAC learnable if and only if its VC-dimension is nite. In the following theorem we nd a di erent characterization for nonuniform learnable classes for the task of binary classi cation.

theorem 7.2 A hypothesis class H of binary classi ers is nonuniformly learn-able if and only if it is a countable union of agnostic PAC learnable hypothesis classes.

The proof of Theorem [7.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page84) relies on the following result of independent interest:

|  |  |
| --- | --- |
| 7.2 Structural Risk Minimization | 85 |
|  |  |

theorem 7.3 Let H be a hypothesis class that can be written as a countable

S

union of hypothesis classes, H = n2N Hn, where each Hn enjoys the uniform convergence property. Then, H is nonuniformly learnable.

Recall that in Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54) we have shown that uniform convergence is su cient for agnostic PAC learnability. Theorem [7.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page84) generalizes this result to nonuni-form learnability. The proof of this theorem will be given in the next section by introducing a new learning paradigm. We now turn to proving Theorem [7.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page84).

|  |  |  |
| --- | --- | --- |
| Proof of Theorem [7.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page84) First assume that H = | | n2N Hn where each Hn is ag- |
| nostic PAC learnable. Using the fundamental | theorem of statistical learning, it | |
|  | S |

follows that each Hn has the uniform convergence property. Therefore, using Theorem [7.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page84) we obtain that H is nonuniform learnable.

For the other direction, assume that H is nonuniform learnable using some

algorithm A. For every n 2 N, let Hn = fh 2 H : mNULH(1=8; 1=7; h) ng. Clearly, H = [n2NHn. In addition, using the de nition of mNULH we know that

for any distribution D that satis es the realizability assumption with respect to Hn, with probability of at least 6=7 over S Dn we have that LD(A(S)) 1=8. Using the fundamental theorem of statistical learning, this implies that the VC-dimension of Hn must be nite, and therefore Hn is agnostic PAC learnable. 

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

Example 7.1 Consider a binary classi cation problem with the instance domain being X = R. For every n 2 N let Hn be the class of polynomial classi ers of degree n; namely, Hn is the set of all classi ers of the form h(x) = sign(p(x))

S

where p : R ! R is a polynomial of degree n. Let H = n2N Hn. Therefore, H is the class of all polynomial classi ers over R. It is easy to verify that VCdim(H) = 1 while VCdim(Hn) = n + 1 (see Exercise [12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page81)). Hence, H is not PAC learnable, while on the basis of Theorem [7.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page84), 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 (SRM) paradigm,

S

we do so by rst assuming that H can be written as H = n2N Hn and then specifying a weight function, w : N ! [0; 1], which assigns a weight to each hypothesis class, Hn, such that a higher weight re ects 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.

1. Nonuniform Learnability

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Concretely, let H be a hypothesis class that can be written as H = | |  | n2N Hn. | | |
| For example, H may be the class of all polynomial classi ers where | each | | | Hn | is |
|  | S |  |  |

the class of polynomial classi ers of degree n (see Example [7.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page85)). Assume that for each n, the class Hn enjoys the uniform convergence property (see De nition [4.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page55)

|  |  |  |  |
| --- | --- | --- | --- |
|  | UC | ( ; ). Let us also de ne | |
| in Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54)) with a sample complexity function mHn | |
| the function n : N (0; 1) ! (0; 1) by |  |  |  |
| UC | ( ; ) | mg: | (7.1) |
| n(m; ) = minf 2 (0; 1) : mHn |

In words, we have a xed 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 de nitions of uniform convergence and n, it follows that for every m and , with probability of at least 1 over the choice of S Dm we have that

|  |  |
| --- | --- |
| 8h 2 Hn; jLD(h) LS(h)j n(m; ): | (7.2) |

P1

Let w : N ! [0; 1] be a function such that n=1 w(n) 1. We refer to w as

a weight function over the hypothesis classes H1; H2; : : :. Such a weight function

can re ect the importance that the learner attributes to each hypothesis class, or some measure of the complexity of di erent hypothesis classes. If H is a nite 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, re ecting this prior knowledge. When H is a (countable) in nite union of hypothesis classes, a uniform weighting is not possible but many other weighting schemes may work. For example, one can choose w(n) = 26n2 or w(n) = 2 n. Later in this chapter we will provide another convenient way to de ne weighting functions using description languages.

The SRM rule follows a \bound minimization" approach. This means that the goal of the paradigm is to nd 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| H |  | H |  |  | 1 |
|  |  | n2N HP | |
| theorem 7.4 Let w : N ! [0; 1] be a function such that | | | | | n=1 w(n) 1. Let |
|  | be a hypothesis class that can be written as |  | = |  | n, where for each n, |

S

Hn satis es the uniform convergence property with a sample complexity function

mUC . Let n be as de ned in Equation ([7.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page86)). Then, for every 2 (0; 1) and

Hn

distribution D, with probability of at least 1 over the choice of S Dm, the following bound holds (simultaneously) for every n 2 N and h 2 Hn.

jLD(h) LS(h)j n(m; w(n) ):

Therefore, for every 2 (0; 1) and distribution D, with probability of at least

|  |  |
| --- | --- |
| 7.2 Structural Risk Minimization | 87 |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1it holds that |  |  |  |  |
| 8h 2 H; LD(h) LS(h) + | min | n | n(m; w(n) ): | (7.3) |
| n:h |  |
|  | 2H |  |  |  |

Proof For each n de ne n = w(n) . Applying the assumption that uniform convergence holds for all n with the rate given in Equation ([7.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page86)), we obtain that if we x n in advance, then with probability of at least 1 n over the choice of

1. Dm,

8h 2 Hn; jLD(h) LS(h)j n(m; n):

Applying the union bound over n = 1; 2; : : :, we obtain that with probability of

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| at least 1 | | n n = 1 | n w(n) 1 , the preceding holds for all n, which | |
| concludes | our proof. | | P |  |
|  | P |  |
| Denote |  |  |  |  |
|  |  |  | n(h) = minfn : h 2 Hng; | (7.4) |

and then Equation ([7.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page87)) implies that

LD(h) LS(h) + n(h)(m; w(n(h)) ):

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

Structural Risk Minimization (SRM)

|  |  |  |  |
| --- | --- | --- | --- |
| prior knowledge: | |  | n w(n) 1 |
| w : NS! [0; 1] where | |  |
|  |  |  | UC |
| H = n Hn where Hn has uniform convergence with mHn | | | |
| de ne: n as in | Equation ([7.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page86)) ; n(h) as in Equation ([7.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page87)) | | |
|  | P | |
| input: training set S Dm, con dence | | | |
| output: h 2 argminh2H | | | LS(h) + n(h)(m; w(n(h)) ) |

Unlike the ERM paradigm discussed in previous chapters, we no longer just care about the empirical risk, LS(h), but we are willing to trade some of our bias toward low empirical risk with a bias toward classes for which n(h)(m; w(n(h)) ) 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 be a hypothesis class such that H = | | n | 2 | N Hn, where | | |
| each Hn has the uniform convergence property with sample |  |  |  | UC | . Let |
| complexity m | | | | Hn |
|  | S |  |  |  |

w : N ! [0; 1] be such that w(n) = n26 2 . Then, H is nonuniformly learnable using the SRM rule with rate

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| mH | ( ; ; h)mHn(h) | =2 ; | ( n(h))2 | | | : |
| NUL | UC |  |  | 6 |  |  |

1. Nonuniform Learnability

Proof Let A be the SRM algorithm with respect to the weighting function w. For every h 2 H, , and , let m mUC ( ; w(n(h)) ). Using the fact that

Hn(h)

P

n w(n) = 1, we can apply Theorem [7.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page86) to get that, with probability of at least 1 over the choice of S Dm, we have that for every h0 2 H,

LD(h0) LS(h0) + n(h0)(m; w(n(h0)) ):

The preceding holds in particular for the hypothesis A(S) returned by the SRM rule. By the de nition of SRM we obtain that

LD(A(S)) min LS(h0) + n(h0)(m; w(n(h0)) ) LS(h) + n(h)(m; w(n(h)) ):

h0

Finally, if m mHn(h) ( =2; w(n(h)) ) then clearly n(h)(m; w(n(h)) ) =2. In addition, from the uniform convergence property of each Hn we have that with

UC

probability of more than 1 ,

LS(h) LD(h) + =2:

Combining all the preceding we obtain that LD(A(S)) LD(h) + , which con-cludes our proof. 

Note that the previous theorem also proves Theorem [7.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page84).

Remark 7.2 (No-Free-Lunch for Nonuniform Learnability) We have shown that any countable union of classes of nite VC-dimension is nonuniformly learnable. It turns out that, for any in nite domain set, X , the class of all binary valued functions over X is not a countable union of classes of nite VC-dimension. We leave the proof of this claim as a (nontrivial) exercise (see Exercise [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page98)). It follows that, in some sense, the no free lunch theorem holds for nonuniform learning as well: namely, whenever the domain is not nite, there exists no nonuniform learner with respect to the class of all deterministic binary classi ers (although for each such classi er there exists a trivial algorithm that learns it { ERM with respect to the hypothesis class that contains only this classi er).

It is interesting to compare the nonuniform learnability result given in The-orem [7.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page87) to the task of agnostic PAC learning any speci c 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 spe-ci c Hn. The cost of this weakening of prior knowledge is the increase in sample complexity needed to compete with any speci c h 2 Hn. For a concrete evalua-tion of this gap, consider the task of binary classi cation with the zero-one loss.

Assume that for all n, VCdim(Hn) = n. Since mUC ( ; ) = C n+log(1= ) (where

Hn 2

C is the contant appearing in Theorem [6.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72)), a straightforward calculation shows that

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NUL | UC |  | 2 log(2n) |  |
| mH | ( ; ; h) mHn | ( =2; ) 4C |  | : |
| 2 |

That is, the cost of relaxing the learner's prior knowledge from a speci c Hn that contains the target h to a countable union of classes depends on the log of

|  |  |
| --- | --- |
| 7.3 Minimum Description Length and Occam's Razor | 89 |
|  |  |

the index of the rst class in which h resides. That cost increases with the index of the class, which can be interpreted as re ecting 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. Then, we can write H as a countable

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| union of singleton classes, namely, H = | | | | | | | | | | | | | | | n2Nfhng. By Hoe ding's inequality | | |
| (Lemma [4.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page56)), each singleton class has | | | | | | | | | | | | | | the uniform convergence property with | | | |
|  | S | |  |
| rate m | UC | ( ; ) | = |  | log(2= ) | | | | . Therefore, the function n given in Equation ([7.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page86)) | | | | | | | | |
|  |  | 2 2 | | |  |
| becomes n(m; ) = q | | | | | |  | log(2= ) | | |  | and the SRM rule becomes | | | | | |  |
|  | 2m | | |  |
|  |  |  |  | hn2H | | | | "LS(h) + r | | | |  |  |  |  |  | # : |
|  |  |  |  |  | |  | 2m | |
|  |  |  | argmin | | | | |  |  |  |  |  |  | log(w(n)) + log(2= ) | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

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

" #

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| h2H | LS(h) + r |  | | 2m | | : |
| argmin |  |  |  | log(w(h)) + log(2= ) | |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

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 de ne a weight func-tion 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 x some description language. This can be English, or a programming language, or some set of mathematical formu-las. In any of these languages, a description consists of nite strings of symbols (or characters) drawn from some xed alphabet. We shall now formalize these notions.

Let H be the hypothesis class we wish to describe. Fix some nite set of symbols (or \characters"), which we call the alphabet. For concreteness, we let = f0; 1g. A string is a nite sequence of symbols from ; for example,

= (0; 1; 1; 1; 0) is a string of length 5. We denote by j j the length of a string. The set of all nite length strings is denoted . A description language for H is a function d : H ! , 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 jhj.

We shall require that description languages be pre x-free; namely, for every

distinct h; h0, d(h) is not a pre x of d(h0). That is, we do not allow that any string d(h) is exactly the rst jhj symbols of any longer string d(h0). Pre x-free collections of strings enjoy the following combinatorial property:

1. Nonuniform Learnability

lemma 7.6 (Kraft Inequality) If S f0; 1g is a pre x-free set of strings, then

X 1

2j j 1:

2S

Proof De ne a probability distribution over the members of S as follows: Re-peatedly 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 2 S, let P ( ) be the probability that this process generates the string . Note that since S is pre x-free, for every 2 S, if the coin toss outcomes follow the bits of then we will stop only once the sequence of outcomes equals . We therefore get that, for every 2 S, P ( ) = 2j1 j . Since probabilities add up to at most 1, our proof is concluded. 

In light of Kraft's inequality, any pre x-free description language of a hypoth-esis 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: |
| 2jhj |
|  |  |

theorem 7.7 Let H be a hypothesis class and let d : H ! f0; 1g be a pre x-free description language for H. Then, for every sample size, m, every con dence parameter, > 0, and every probability distribution, D, with probability greater than 1 over the choice of S Dm we have that,

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8h 2 H; LD(h) LS(h) + r | | | | ln(2= ) | | | | |  |  |  |
| jhj + | |  | ; |  |  |  |  |
| 2m | |  |  |  |  |
| where jhj is the length of d(h). | | |  |  |  |  | q |  |  |  |  |
| note that ln(2jhj) = | h ln(2)j | j< h . | |  | n |  |  | 2m | | |
| Proof Choose w(h) = 1=2 h , apply Theorem [7.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page86) with | | | | |  | (m; ) = | | | ln(2= ) | , and | |
|  |  |
| j | j |  | j j |  |  |  |  |  |  |  |  |

As was the case with Theorem [7.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page86), this result suggests a learning paradigm for H { given a training set, S, search for a hypothesis h 2 H that minimizes

|  |  |  |
| --- | --- | --- |
| the bound, LS(h) + qjhj | +ln(2= ) | . In particular, it suggests trading o empirical |
| 2m |

risk for saving description length. This yields the Minimum Description Length learning paradigm.

Minimum Description Length (MDL)

prior knowledge:

H is a countable hypothesis class

H is described by a pre x-free language over f0; 1g

For every h 2 H, jhj is the length of the representation of h input: A training set S Dm, con dence

output: h 2 argminh2H LS(h)

q

1. jhj+ln(2= )

2m

Example 7.3 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

|  |  |
| --- | --- |
| 7.3 Minimum Description Length and Occam's Razor | 91 |
|  |  |

binary string obtained by running the gzip command on the program (this yields a pre x-free description language over the alphabet f0; 1g). Then, jhj 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page90) 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 rst to phrase it explicitly. Here, we provide one possible justi cation to this principle. The inequality of Theorem [7.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page90) 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 t to guarantee that it has a small true risk, LD(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. As-sume that we have two hypotheses such that jh 0j is much smaller than jhj. 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 h0, so one should prefer h0 over h. However, we could have chosen a di erent description language, say, one that assigns a string of length 3 to h and a string of length 100000 to h0. Suddenly it looks as if one should prefer h over h0. But these are the same h and h0 for which we argued two sentences ago that h0 should be preferable. Where is the catch here?

Indeed, there is no inherent generalizability di erence 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 Hoe ding's bound (Equation ([4.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page57))), if we commit to any hypothesis be-fore seeing the data, then we are guaranteed a rather small estimation error term

q

|  |  |  |
| --- | --- | --- |
| LD(h) LS(h) + | ln(2= ) | . Choosing a description language (or, equivalently, |
| 2m |

some weighting of hypotheses) is a weak form of committing to a hypothesis. 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 gen-eralization 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.

1. Nonuniform Learnability

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 ; , 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page92) 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 learn-ing rule A is consistent with respect to H and P if there exists a function mCONH : (0; 1)2 H P ! N such that, for every ; 2 (0; 1), every h 2 H, and every D 2 P, if m mNULH( ; ; h; D) then with probability of at least 1 over the choice of S Dm it holds that

LD(A(S)) LD(h) + :

If P is the set of all distributions,[2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page92) 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 de ned in Example [7.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page92) later is universally consistent for the class of all binary classi ers over N. However, as we have argued before, this class is not nonuniformly learnable.

Example 7.4 Consider the classi cation prediction algorithm Memorize de ned 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 xed default label if no instance of x appears in the training set). It is possible to show (see Exercise [6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page98)) that the Memorize algorithm is universally consistent for every countable domain X and a nite 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page33),

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

to strong consistency).

1. Formally, we assume that Z is endowed with some sigma algebra of subsets , and by \all distributions" we mean all probability distributions that have contained in their associated family of measurable subsets.

|  |  |
| --- | --- |
| 7.5 Discussing the Di erent Notions of Learnability | 93 |
|  |  |

which led to over tting, is in fact the Memorize algorithm. In the next section we discuss the signi cance of the di erent notions of learnability and revisit the No-Free-Lunch theorem in light of the di erent de nitions of learnability.

7.5 Discussing the Di erent Notions of Learnability

We have given three de nitions of learnability and we now discuss their useful-ness. As is usually the case, the usefulness of a mathematical de nition depends on what we need it for. We therefore list several possible goals that we aim to achieve by de ning learnability and discuss the usefulness of the di erent de ni-tions in light of these goals.

What Is the Risk of the Learned Hypothesis?

The rst possible goal of deriving performance guarantees on a learning algo-rithm 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page144)).

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

When approaching a learning problem, a natural question is how many exam-ples 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 H. In nonuniform learning this number depends on the best hypothesis in H, and in consistency it also depends on the underlying distribution. In this sense, PAC learning is the only useful de nition of learnability. On the ip 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 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 re ects 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 di erent hypothesis class. Similarly, if a nonuniform algorithm fails, we can consider a di erent 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

1. Nonuniform Learnability

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?

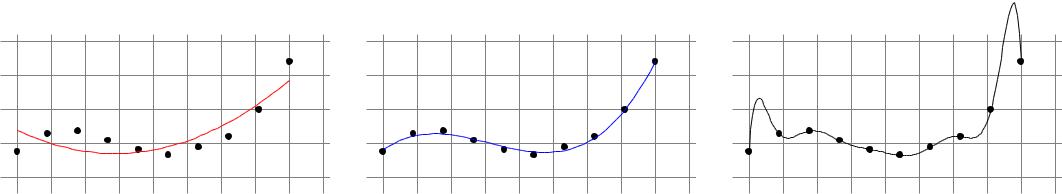
Maybe the most useful aspect of the theory of learning is in providing an answer to the question of \how to learn." The de nition of PAC learning yields the limitation of learning (via the No-Free-Lunch theorem) and the necessity of prior knowledge. It gives us a crisp way to encode prior knowledge by choosing a hypothesis class, and once this choice is made, we have a generic learning rule { ERM. The de nition of nonuniform learnability also yields a crisp way to encode

prior knowledge by specifying weights over (subsets of) hypotheses of 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page144) and here we give a brief example.

Consider the problem of tting a one dimensional polynomial to data; namely,

our goal is to learn a function, h : R ! 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 t the data well (i.e., it will have a large approximation error), whereas a high degree might lead to over tting (i.e., it will have a large estimation error). In the following we depict the result of tting a polynomial of degrees 2, 3, and 10 to the same training set.

degree 2 degree 3 degree 10



It is easy to see that the empirical risk decreases as we enlarge the degree. Therefore, if we choose 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 over t. On the other hand, if we choose too small a hypothesis class, say, polynomials up to degree 2, then the ERM would su er from under tting (i.e., a large approximation error). In contrast, we can use the SRM rule on the set of all polynomials, while ordering subsets of 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 exibility (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

|  |  |
| --- | --- |
| 7.5 Discussing the Di erent Notions of Learnability | 95 |
|  |  |

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

Unlike the notions of PAC learnability and nonuniform learnability, the de ni-tion 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 de ned over a countable domain and a nite 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

1. to Y. Concretely, consider a countable domain, X , a nite 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 classi ers from X to Y using the following simple trick: Upon receiving a training set, we will rst 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 modi cation

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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page61) from Chapter [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page60)) implies that no algorithm can learn the class of all classi ers over an in nite domain. In contrast, in this chapter we saw that the Memorize algorithm is consistent with respect to the class of all classi ers over a countable in nite domain. To understand why these two statements do not contradict each other, let us rst recall the formal statement of the No-Free-Lunch theorem.

Let X be a countable in nite domain and let Y = f 1g. The No-Free-Lunch theorem implies the following: For any algorithm, A, and a training set size, m, there exist a distribution over X and a function h? : X ! Y, such that if A

1. Nonuniform Learnability

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

The consistency of Memorize implies the following: For every distribution over

1. and a labeling function h? : X ! 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 classi er with a small error.

We see that in the No-Free-Lunch theorem, we rst x the training set size, and then nd a distribution and a labeling function that are bad for this training set size. In contrast, in consistency guarantees, we rst x the distribution and the labeling function, and only then do we nd a training set size that su ces 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 in nite VC-dimension can be learnable, in some weaker sense of learn-ability. We discussed the usefulness of the di erent de nitions of learnability.

For hypothesis classes that are countable, we can apply the Minimum Descrip-tion Length scheme, where hypotheses with shorter descriptions are preferred, following the principle of Occam's razor. An interesting example is the hypothe-sis 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 1000+log(2= ) , the runtime is 21000. This is a huge

2

number { much larger than the number of atoms in the visible universe. In the next chapter we formally de ne 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 e ciently.

|  |  |
| --- | --- |
| 7.7 Bibliographic Remarks | 97 |
|  |

7.7 Bibliographic Remarks

Our de nition of nonuniform learnability is related to the de nition 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 pre-sentation of consistency follows Steinwart & Christmann (2008), who also derived several no-free-lunch theorems.

7.8 Exercises

1. Prove that for any nite class H, and any description language d : H !

f0; 1g , the VC-dimension of H is at most 2 supfjd(h)j : h 2 Hg { the maxi-mum description length of a predictor in H. Furthermore, if d is a pre x-free description then VCdim(H) supfjd(h)j : h 2 Hg.

1. Let H = fhn : n 2 Ng be an in nite countable hypothesis class for binary classi cation. Show that it is impossible to assign weights to the hypotheses

in H such that

H could be learnt nonuniformly using these weights. That is, the weighting

P

function w : H ! [0; 1] should satisfy the condition h2H w(h) 1.

The weights would be monotonically nondecreasing. That is, if i < j, then w(hi) w(hj).

1. Consider a hypothesis class H = S1n=1 Hn, where for every n 2 N, Hn is

|  |  |  |  |
| --- | --- | --- | --- |
| nite. Find a weighting function w : H ! [0; 1] such that |  | h2H w(h) | |
| 1 and so that for all h 2 H, w(h) is determined by n(h) | = min n : h | | 2 |
| P | f |

(\*) De ne such a function w when for all n Hn is countable (possibly in nite).

1. Let H be some hypothesis class. For any h 2 H, let jhj denote the description length of h, according to some xed description language. Consider the MDL learning paradigm in which the algorithm returns:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| hS 2 arg h2H | " | S |  | r |  |  |  | # |
|  | j j 2m | | |
| min |  | L | (h) + |  |  | h + ln(2= ) | | ; |
|  |  |  |  |  |

where S is a sample of size m. For any B > 0, let HB = fh 2 H : jhj Bg, and de ne

hB = arg min LD(h):

h2HB

1. Nonuniform Learnability

Prove a bound on LD(hS) LD(hB) in terms of B, the con dence parameter

, 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 classi er (or \oracle") hB.

1. In this question we wish to show a No-Free-Lunch result for nonuniform learn-ability: namely, that, over any in nite 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 H if there exists a function mNULH : (0; 1)2 H ! N such that, for every ; 2 (0; 1) and for every h 2 H, if m mNULH( ; ; h) then for every distribution D, with probability of at least 1 over the choice of S Dm, it holds that

LD(A(S)) LD(h) + :

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 2 N de ne HnA = fh 2 H : mNUL(0:1; 0:1; h) ng. Prove that each such class Hn has a nite VC-dimension.
2. Prove that if a class H is nonuniformly learnable then there are classes Hn

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| so that H = | n2N Hn and, for every n 2 N, VCdim(Hn) is nite. | | | |
| class that shatters an in nite set. Then, for every sequence | | | | |
| 3. Let H be a S |  | S | 2 | N Hn, there exists some n for |
| which VCdim(Hn) = 1. | |
| of classes (Hn | : n 2 N) such that H = | n |  |

Hint: Given a class H that shatters some in nite set K, and a sequence of

classes (Hn : n 2 N), each having a nite VC-dimension, start by de ning subsets Kn K such that, for all n, jKnj > VCdim(Hn) and for any n 6= m, Kn \ Km = ;. Now, pick for each such Kn a function fn : Kn ! f0; 1g so that no h 2 Hn agrees with fn on the domain Kn. Finally, de ne

S

f : X ! f0; 1g by combining these fn's and prove that f 2 H n n2N Hn .

* 1. Construct a class H1 of functions from the unit interval [0; 1] to f0; 1g that is nonuniformly learnable but not PAC learnable.
  2. Construct a class H2 of functions from the unit interval [0; 1] to f0; 1g that is not nonuniformly learnable.

1. 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 fxi : i 2 Ng be an enumeration of the elements of X so that for all
2. j, D(fxig) D(fxjg). Prove that

X

lim D(fxig) = 0:

n!1

i n

1. Given any > 0 prove that there exists D > 0 such that D(fx 2 X : D(fxg) < Dg) < :

|  |  |
| --- | --- |
| 7.8 Exercises | 99 |
|  |  |

1. Prove that for every > 0, if n is such that D(fxig) < for all i > n, then for every m 2 N,

P [9xi : (D(fxig) > and xi 2= S)] ne m:

S Dm

1. Conclude that if X is countable then for every probability distribution D over X there exists a function mD : (0; 1) (0; 1) ! N such that for every ; > 0 if m > mD( ; ) then
2. [D(fx : x 2= Sg) > ] < :

S Dm

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

1. 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 com-plexity of a task: that is, how much computation is involved in carrying out a learning task. Once a su cient training sample is available to the learner, there is some computation to be done to extract a hypothesis or gure 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 con-text 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 speci c ma-chine the algorithm is being implemented on (e.g., what the clock rate of the machine's CPU is). To avoid dependence on the speci c 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(n log(n)). This implies that we can implement the algo-rithm on any machine that satis es the requirements of some accepted abstract model of computation, and the actual runtime in seconds will satisfy the follow-ing: 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 c n log(n). It is common to use the term feasible or e ciently 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 de ning what is the input size n of any instance to which the algorithm is expected to be applied. For \purely algorithmic" tasks, as dis-cussed in the common computational complexity literature, this input size is clearly de ned; the algorithm gets an input instance, say, a list to be sorted, or an arithmetic operation to be calculated, which has a well de ned size (say, the

|  |  |
| --- | --- |
| 8.1 Computational Complexity of Learning | 101 |
|  |  |

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 de ne the computational complexity of learning. For advanced students, we also provide a detailed formal de nition. We then move on to consider the computational complexity of im-plementing the ERM rule. We rst give several examples of hypothesis classes where the ERM rule can be e ciently implemented, and then consider some cases where, although the class is indeed e ciently learnable, ERM implemen-tation is computationally hard. It follows that hardness of implementing ERM does not imply hardness of learning. Finally, we brie y discuss how one can show hardness of a given learning task, namely, that no learning algorithm can solve it e ciently.

8.1 Computational Complexity of Learning

Recall that a learning algorithm has access to a domain of examples, Z, a hy-pothesis 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 ; , the algorithm should output a hypothesis h such that with probability of at least 1 ,

LD(h) min LD(h0) + :

h02H

As mentioned before, the actual runtime of an algorithm in seconds depends on the speci c 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 speci c 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 de ne 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 learn-ing problem, the algorithm can simply ignore the extra examples. Therefore, a larger training set does not make the learning problem more di cult, and, con-sequently, 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 con dence of achieving that accuracy, the dimensionality of the

1. The Runtime of Learning

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 speci c problem of learning axis aligned rectangles is de-rived by specifying , , and the dimension of the instance space. We can de ne a sequence of problems of the type \rectangles learning" by xing ; and varying the dimension to be d = 2; 3; 4; : : :. We can also de ne another sequence of \rect-

angles learning" problems by xing d; and varying the target accuracy to be

= 12 ; 13 ; : : :. One can of course choose other sequences of such problems. Once a sequence of the problems is xed, one can analyze the asymptotic runtime as a function of variables of that sequence.

Before we introduce the formal de nition, 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 de ne 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 xed 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 classi er 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 classi er from the input training sample). In the next subsection the advanced reader may nd a formal de nition of the computational complexity of learning.

8.1.1 Formal De nition\*

The de nition 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 de ne the complexity of learning in two steps. First we consider the compu-tational complexity of a xed 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

|  |  |
| --- | --- |
| 8.2 Implementing the ERM Rule | 103 |
|  |  |

over Z, and input ; 2 (0; 1), when A has access to samples generated i.i.d. by D,

A terminates after performing at most cf( ; ) operations

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

The output of A is probably approximately correct; namely, with proba-

bility of at least 1 (over the random samples A receives), LD(hA) minh02H LD(h0) +

1. Consider a sequence of learning problems, (Zn; Hn; `n)1n=1, where problem n is de ned 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 (Zn; Hn; `n) in time O(fn), where fn : (0; 1)2 ! N is de ned by fn( ; ) = g(n; ; ).

We say that A is an e cient algorithm with respect to a sequence (Zn; Hn; `n) if its runtime is O(p(n; 1= ; 1= )) for some polynomial p.

From this de nition we see that the question whether a general learning prob-lem can be solved e ciently depends on how it can be broken into a sequence of speci c learning problems. For example, consider the problem of learning a nite hypothesis class. As we showed in previous chapters, the ERM rule over H is guaranteed to ( ; )-learn H if the number of training examples is order of mH( ; ) = log(jHj= )= 2. 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(jHj mH( ; )) by performing an exhaustive search over H with a training set of size mH( ; ). For any xed nite H, the exhaustive search algorithm runs in polynomial time. Furthermore, if we de ne a sequence of problems in which jHnj = n, then the exhaustive search is still considered to be e cient. However, if we de ne a sequence of problems for which jHnj = 2n, then the sample complex-ity is still polynomial in n but the computational complexity of the exhaustive search algorithm grows exponentially with n (thus, rendered ine cient).

8.2 Implementing the ERM Rule

Given a hypothesis class H, the ERMH rule is maybe the most natural learning paradigm. Furthermore, for binary classi cation 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 corre-sponding ERMH rule can be de ned as follows:

1. The Runtime of Learning

On a nite input sample S 2 Zm output some h 2 H that minimizes the empirical loss, LS(h) = jS1j Pz2S `(h; z).

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 nite class may be considered as a reason-ably 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 ex-amples of useful nite classes are any hypothesis class that can be parameterized by a nite number of parameters, where we are satis ed with a representation of each of the parameters using a nite number of bits, for example, the class of axis aligned rectangles in the Euclidean space, Rd, when the parameters de ning any given rectangle are speci ed up to some limited precision.

As we have shown in previous chapters, the sample complexity of learning a nite class is upper bounded by mH( ; ) = c log(cjHj= )= c, 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= ))= c.

A straightforward approach for implementing the ERM rule over a nite hy-pothesis class is to perform an exhaustive search. That is, for each h 2 H we 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 exam-ple takes a constant amount of time, k, the runtime of this exhaustive search becomes kjHjm, 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 kjHjc log(cjHj= )= c.

The linear dependence of the runtime on the size of H makes this approach ine cient (and unrealistic) for large classes. Formally, if we de ne a sequence of problems (Zn; Hn; `n)1n=1 such that log(jHnj) = 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 nite classes.

It is important to realize that the ine ciency of one algorithmic approach (such as the exhaustive search) does not yet imply that no e cient ERM imple-mentation exists. Indeed, we will show examples in which the ERM rule can be implemented e ciently.

|  |  |
| --- | --- |
| 8.2 Implementing the ERM Rule | 105 |
|  |  |

8.2.2 Axis Aligned Rectangles

Let Hn be the class of axis aligned rectangles in Rn, namely,

Hn = fh(a1;:::;an;b1;:::;bn) : 8i; ai big

where

|  |  |  |  |
| --- | --- | --- | --- |
| h(a1;:::;an;b1;:::;bn)(x; y) = ( | 1 | if 8i; xi 2 [ai; bi] | (8.1) |
| 0 | otherwise |

E ciently Learnable in the Realizable Case

Consider implementing the ERM rule in the realizable case. That is, we are given

1. training set S = (x1; y1); : : : ; (xm; ym) of examples, such that there exists an axis aligned rectangle, h 2 Hn, for which h(xi) = yi for all i. Our goal is to nd 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 2 [n], set ai = minfxi : (x; 1) 2 Sg and bi = maxfxi : (x; 1) 2 Sg. In words, we take ai to be the minimal value of the i'th coordinate of a positive example in S and bi 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 nding each ai and bi is O(m). Hence, the total runtime of this procedure is O(nm).

Not E ciently 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 nd h that minimizes the number of examples for which yi 6= h(xi). 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 nd some h 2 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 nd an ERM hypothesis for these problems (Ben-David, Eiron & Long 2003).

On the other hand, it is worthwhile noticing that, if we x one speci c hypoth-esis class, say, axis aligned rectangles in some xed dimension, n, then there exist e cient learning algorithms for this class. In other words, there are successful agnostic PAC learners that run in time polynomial in 1= and 1= (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 per-form 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

1. The Runtime of Learning

the rectangle with the minimal training error. This procedure is guaranteed to

nd an ERM hypothesis, and the runtime of the procedure is mO(n). It follows that if n is xed, 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 = f0; 1gn to Y = f0; 1g that can be expressed as a proposition formula of the form xi1 ^ : : : ^ xik ^ :xj1 ^ : : : ^ :xjr , for some indices i1; : : : ; ik; j1; : : : ; jr 2 [n]. The function that such a proposition formula de nes is

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| h(x) = | ( | 0 | otherwise |  | k |  |  |  |  |  |
|  |  | 1 | if xi1 = | = xi |  | = 1 and xj1 | = |  | = xjr | = 0 |

Let HCn be the class of all Boolean conjunctions over f0; 1gn. The size of HCn is 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 HCn using the ERM rule is at most n log(3= )= .

E ciently Learnable in the Realizable Case

Next, we show that it is possible to solve the ERM problem for HCn in time polynomial in n and m. The idea is to de ne 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 de ne, 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 satis ed 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 2 HCn , that is consistent with all the examples in 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).

|  |  |
| --- | --- |
| 8.3 E ciently Learnable, but Not by a Proper ERM | 107 |
|  |  |

Not E ciently 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 nd 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 = f0; 1gn and each hypothesis is represented by the Boolean formula of the form h(x) = A1(x) \_ A2(x) \_ A3(x), where each Ai(x) is a Boolean conjunction (as de ned 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 H3nDNF be the hypothesis class of all such 3-term DNF formulae. The size of H3nDNF is at most 33n. Hence, the sample complexity of learning H3nDNF using

the ERM rule is at most 3n log(3= )= .

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 ERMHn3DNF outputs a 3-term DNF 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page111). See also (Kearns & Vazirani 1994, Section 1.4).

8.3 E ciently Learnable, but Not by a Proper ERM

In the previous section we saw that it is impossible to implement the ERM rule e ciently for the class H3nDNF of 3-DNF formulae. In this section we show that it is possible to learn this class e ciently, 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 e ciently. 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

1. The Runtime of Learning

algorithm might return a hypothesis that does not belong to the original hypoth-esis 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 \_ distributes over ^, each 3-term DNF formula can be rewritten as

^

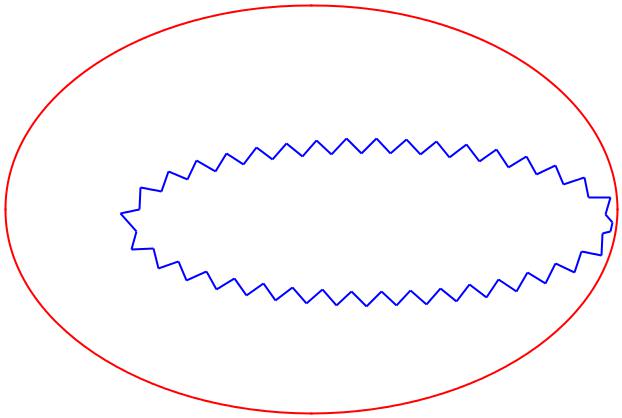
A1 \_ A2 \_ A3 =

(u \_ v \_ w)

u A1;v2A2;w2A3

Next, let us de ne: : f0; 1gn ! f0; 1g(2n)3 such that for each triplet of literals u; v; w there is a variable in the range of indicating if u \_ v \_ w is true or false. So, for each 3-DNF formula over f0; 1gn there is a conjunction over f0; 1g(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 f0; 1g(2n)3 . Furthermore, the sample complexity of learning the class of conjunctions in the higher dimensional space is at most n3 log(1= )= . 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 e cient ERM search. In the new representation, solving the ERM problem is easy.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | 3 |
|  |  |  |  |  |  |  | ) |
|  |  |  |  |  |  | n | |
|  |  |  |  |  | (2 | |  |
|  |  |  |  | g | |  |  |
|  |  |  | 1 | |  |  |  |
|  |  | ; | |  |  |  |  |
|  | 0 | |  |  |  |  |  |
| over | f |  |  |  |  |  |  |

conjunctions

3-term-DNF formulae over f0; 1gn

8.4 Hardness of Learning\*

We have just demonstrated that the computational hardness of implementing ERMH 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, cryp-tography 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

|  |  |
| --- | --- |
| 8.4 Hardness of Learning\* | 109 |
|  |  |

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 su ce 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 cryp-tographic 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 brie y describe the basic idea of how to deduce hardness of learnabil-ity 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 : f0; 1gn ! f0; 1gn (more formally, it is a sequence of functions, one for each dimension n) that is easy to compute but is hard to in-vert. 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)] < p(1n) ;

where the probability is taken over a random choice of x according to the uniform distribution over f0; 1gn and the randomness of A.

A one way function, f, is called trapdoor one way function if, for some poly-nomial 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 2 f0; 1gn, 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 f0; 1gn that can be calcu-lated by some polynomial time algorithm. That is, we x 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, HFn = ff 1 : f 2 Fn g. Since each function in this class can be inverted by some secret key sn of size polynomial in n, the class HFn can be parameter-ized 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 e cient learner for this class. If there were such a learner, L, then by sampling uniformly at random a polynomial number of strings in f0; 1gn, and computing f over them, we could generate a labeled training sample of pairs (f(x); x), which should su ce for our learner to gure out an ( ; ) 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

1. The Runtime of Learning

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

8.5 Summary

The runtime of learning algorithms is asymptotically analyzed as a function of di erent parameters of the learning problem, such as the size of the hypothe-sis class, our measure of accuracy, our measure of con dence, or the size of the domain set. We have demonstrated cases in which the ERM rule can be imple-mented e ciently. For example, we derived e cient 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 di erence between the realizable and agnostic cases (i.e., a class is learnable in both cases if and only if it has a nite VC-dimension). In contrast, as we saw, from the computational perspective the di erence is im-mense. 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 e ciently 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 e cient PAC learning model in which the runtime of the algorithm is required to be polynomial in 1= , 1= , 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 ERMH learning rule (in the agnostic case) that given a training set of size m, runs

in time O(m2).

Hint: Use dynamic programming.

1. Let H1; H2; : : : be a sequence of hypothesis classes for binary classi cation. 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 Hn only depends on O(n) examples out of the training set. Furthermore,

|  |  |
| --- | --- |
| 8.7 Exercises | 111 |
|  |

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 nd an ERM hypothesis in the realizable case that is de ned by at most 2n examples. Prove that in such cases, it is possible to nd an ERM hypothesis for Hn in the unrealizable case in time O(mn mO(n)).

1. In this exercise, we present several classes for which nding an ERM classi-

er is computationally hard. First, we introduce the class of n-dimensional halfspaces, HSn, for a domain X = Rn. This is the class of all functions of the form hw;b(x) = sign(hw; xi + b) where w; x 2 Rn, hw; xi is their inner product, and b 2 R. See a detailed description in Chapter [9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117).

* 1. Show that ERMH over the class H = HSn of linear predictors is compu-tationally 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.

Hint: You can prove the hardness by a reduction from the following prob-

lem:

Max FS: Given a system of linear inequalities, Ax > b with A 2 Rm n and b 2

Rm (that is, a system of m linear inequalities in n variables, x = (x1; : : : ; xn)), nd 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 nds an ERMHSn hypothesis for any training sample S 2 (Rn f+1; 1g)m can be used to solve the Max FS problem of size m; n. Hint: De ne 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 satis es an inequality q if and only if the labeled point that corresponds to q is classi ed 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).

* 1. Let X = Rn and let Hkn be the class of all intersections of k-many linear halfspaces in Rn. In this exercise, we wish to show that ERMHnk is com-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.

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

Given a graph G = (V; E), and a number k, determine whether there exists a

function f : V ! f1 : : : kg so that for every (u; v) 2 E, f(u) 6= f(v).

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

1972).

1. The Runtime of Learning

We wish to reduce the k-coloring problem to ERMHnk : that is, to prove that if there is an algorithm that solves the ERMHnk problem in time 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 fv1 : : : vng be the vertices in V . Construct

1. sample S(G) 2 (Rn f 1g)m, where m = jV j + jEj, as follows: For every vi 2 V , construct an instance ei with a negative label.

For every edge (vi; vj) 2 E, construct an instance (ei + ej)=2 with a positive label.

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

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | k | be an ERM classi er in Hkn | |  |  |  |  |  |
| Hint: Let h = | j=1 hj | over S. De ne a | | | | |
| setting f(v | | | ) to be the minimal j such that h | | (e | ) = |  | 1. |
| coloring of V by T |  | i |  | j | i |  |  |

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.

* + 1. Prove that if G is k-colorable then there exists some h 2 Hkn 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 classi er 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.

* + 1. Based on the above, prove that for any k 3, the ERMHnk problem is NP-hard.

1. In this exercise we show that hardness of solving the ERM problem is equiv-alent 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 rst need the following de nition.

definition 8.2 The complexity class Randomized Polynomial (RP) time is the class of all decision problems (that is, problems in which on any instance one has to nd out whether the answer is YES or NO) for which there exists a probabilistic algorithm (namely, the algorithm is allowed to ip random coins while it is running) with these properties:

On any input instance the algorithm runs in polynomial time in the input size.

If the correct answer is NO, the algorithm must return NO.

If the correct answer is YES, the algorithm returns YES with probability a 1=2 and returns NO with probability 1 a.[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page112)

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 de nition can be replaced by any constant in (0; 1).

|  |  |
| --- | --- |
| 8.7 Exercises | 113 |
|  |  |

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 ERMH problem is in the class RP. In particular, this implies that whenever the ERMH 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= and 1= . Your goal is to use that algorithm as a subroutine to contract an algorithm B for solving the ERMH problem in random polynomial time. Given a training set, S 2 (X f 1gm), and some h 2 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 nds a function h 2 H that has error less than = 1=jSj (with respect to that uniform distribution). Show that the algorithm just described satis es the requirements for being a RP solver for ERMH.

Part II

From Theory to Algorithms

1. 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, rst and foremost because of the ability to learn them e ciently in many cases. In addition, linear predictors are intuitive, are easy to interpret, and t 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 rele-vant 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 de ne the class of a ne functions as

Ld = fhw;b : w 2 Rd; b 2 Rg;

|  |  |  |
| --- | --- | --- |
| where |  | wixi! + b: |
| hw;b(x) = hw; xi + b = | d |
|  | Xi |  |
|  | =1 |  |

It will be convenient also to use the notation

Ld = fx 7!wh; xi + b : w 2 Rd; b 2 Rg;

which reads as follows: Ld is a set of functions, where each function is parame-terized by w 2 Rd and b 2 R, and each such function takes as input a vector x and returns as output the scalar hw; xi + b.

The di erent hypothesis classes of linear predictors are compositions of a func-tion : R ! Y on Ld. For example, in binary classi cation, we can choose to be the sign function, and for regression problems, where Y = R, 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 2 X ; namely, let w0 = (b; w1; w2; : : : wd) 2 Rd+1 and let x0 = (1; x1; x2; : : : ; xd) 2

1. Linear Predictors Rd+1. Therefore,

hw;b(x) = hw; xi + b = hw0; x0i:

It follows that each a ne 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 simpli es the presentation, we will omit the bias term and refer to Ld as the class of homogenous linear functions of the form hw(x) = hw; xi.

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

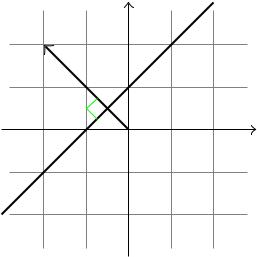
9.1 Halfspaces

The rst hypothesis class we consider is the class of halfspaces, designed for binary classi cation problems, namely, X = Rd and Y = f 1; +1g. The class of halfspaces is de ned as follows:

HSd = sign Ld = fx 7!sign(hw;b(x)) : hw;b 2 Ldg:

In other words, each halfspace hypothesis in HSd is parameterized by w 2 Rd and b 2 R and upon receiving a vector x the hypothesis returns the label sign(hw; xi + 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=w2). 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.



|  |  |
| --- | --- |
| w | + |
|  |  |
|  | + |
|  |  |

In Section [9.1.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page122) we will show that VCdim(HSd) = d + 1. It follows that we can learn halfspaces using the ERM paradigm, as long as the sample size is

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| for |  |  |  |  |
|  |  | d+log(1= ) | | . Therefore, we now discuss how to implement an ERM procedure |
|  |  |  |
|  |  |  |
|  | halfspaces. | | |  |

We introduce below two solutions to nding an ERM halfspace in the realiz-able 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

|  |  |
| --- | --- |
| 9.1 Halfspaces | 119 |
|  |  |

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 non-separable 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 the 0 1 loss, but rather with respect to a di ferent loss function. For example, in Section [9.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page126) we will describe the logistic regression approach, which can be implemented e ciently even in the nonseparable case. We will study surrogate loss functions in more detail later on in Chapter [12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page156).

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 hu; wi

w2Rd

subject to Aw v

where w 2 Rd is the vector of variables we wish to determine, A is an m d matrix, and v 2 Rm; u 2 Rd are vectors. Linear programs can be solved e ciently,[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page119) 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 = f(xi; yi)gmi=1 be a training set of size m. Since we assume the realizable case, an ERM predictor should have zero errors on the training set. That is, we are looking for some vector w 2 Rd for which

sign(hw; xii) = yi; 8i = 1; : : : ; m:

Equivalently, we are looking for some vector w for which

yihw; xii > 0; 8i = 1; : : : ; m:

Let w be a vector that satis es this condition (it must exist since we assume

realizability). De ne = mini(yihw ; xii) and let w = w . Therefore, for all i

we have

yihw; xii = 1 yihw ; xii 1:

We have thus shown that there exists a vector that satis es

|  |  |
| --- | --- |
| yihw; xii 1;8i = 1; : : : ; m: | (9.1) |

And clearly, such a vector is an ERM predictor.

To nd a vector that satis es Equation ([9.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page119)) we can rely on an LP solver as follows. Set A to be the m d matrix whose rows are the instances multiplied

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

1. Linear Predictors

by yi. That is, Ai;j = yi xi;j, where xi;j is the j'th element of the vector xi. Let v be the vector (1; : : : ; 1) 2 Rm. Then, Equation ([9.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page119)) can be rewritten as

Aw 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) 2 Rd.

9.1.2 Perceptron for Halfspaces

A di erent 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 nds an example i that is mis-labeled by w(t), namely, an example for which sign(hw(t); xii) 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 yihw; xii > 0 for all i and note that

yihw(t+1); xii = yihw(t) + yixi; xii = yihw(t); xii + kxik2:

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); : : : ; (xm; ym)

initialize: w(1) = (0; : : : ; 0)

for t = 1; 2; : : :

if (9 i s.t. yihw(t); xii 0) then

w(t+1) = w(t) + yixi

else

output w(t)

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

theorem 9.1 Assume that (x1; y1); : : : ; (xm; ym) is separable, let B = minfkwk : 8i 2 [m]; yi hw; xi i 1g, and let R = maxi kxik. Then, the Perceptron al-gorithm stops after at most (RB)2 iterations, and when it stops it holds that 8i 2 [m]; yihw(t); xii > 0.

Proof By the de nition of the stopping condition, if the Perceptron stops it must have separated all the examples. We will show that if the Perceptron runs for T iterations, then we must have T (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 de nition of B. That is,

|  |  |
| --- | --- |
| 9.1 Halfspaces | 121 |
|  |  |

yihw?; xii 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

p

of the angle between w? and w(T +1) is at least RBT . That is, we will show that

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| hw?; w(T +1)i |  | p |  |  |  |
| T | : | (9.2) |
| kw?k kw(T +1)k | RB | | |
|  |  |

By the Cauchy-Schwartz inequality, the left-hand side of Equation ([9.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page121)) is at most 1. Therefore, Equation ([9.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page121)) would imply that

p

1 RBT ) T (RB)2;

which will conclude our proof.

To show that Equation ([9.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page121)) holds, we rst show that hw?; w(T +1)i T . Indeed, at the rst iteration, w(1) = (0; : : : ; 0) and therefore hw?; w(1)i = 0, while on iteration t, if we update using example (xi; yi) we have that

hw?; w(t+1)i hw?; w(t)i = hw?; w(t+1) w(t)i

= hw?; yixii = yihw?; xii

1:

Therefore, after performing T iterations, we get:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T |  | hw?; w(t+1)i hw?; w(t)i | T; | (9.3) |
| hw?; w(T +1)i = t=1 |
| X | |  |  |  |

as required.

Next, we upper bound kw(T +1)k. For each iteration t we have that

kw(t+1)k2 = kw(t) + yixik2

1. kw(t)k2 + 2yihw(t); xii + yi2kxik2

|  |  |
| --- | --- |
| kw(t)k2 + R2 | (9.4) |

where the last inequality is due to the fact that example i is necessarily such that yihw(t); xii 0, and the norm of xi is at most R. Now, since kw(1)k2 = 0, if we use Equation ([9.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page121)) recursively for T iterations, we obtain that

p

kw(T +1)k2 TR2 ) kw(T +1)k T R: (9.5)

Combining Equation ([9.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page121)) with Equation ([9.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page121)), and using the fact that kw?k = B, we obtain that

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| hw(T +1); w?i |  |  | | |  | p |  |  |  |
|  | T | | | = | T | : |
|  |  |  |  |  |  |  |  |
| kw?k kw(T +1)k |  | B pT R | | | B R | | |
|  |  |

We have thus shown that Equation ([9.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page121)) holds, and this concludes our proof.

1. Linear Predictors

Remark 9.1 The Perceptron is simple to implement and is guaranteed to con-verge. 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 Rd is d.

Proof First, consider the set of vectors e1; : : : ; ed, where for every i the vector ei 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 y1; : : : ; yd, set

1. = (y1; : : : ; yd), and then hw; eii = yi for all i.

Next, let x1; : : : ; xd+1 be a set of d + 1 vectors in Rd. Then, there must exist

Pd+1

real numbers a1; : : : ; ad+1, not all of them are zero, such that i=1 aixi = 0.

Let I = fi : ai > 0g and J = fj : aj < 0g. Either I or J is nonempty. Let us rst assume that both of them are nonempty. Then,

X X

aixi = jajjxj:

i2I j2J

Now, suppose that x1; : : : ; xd+1 are shattered by the class of homogenous classes. Then, there must exist a vector w such that hw; xii > 0 for all i 2 I while hw; xji < 0 for every j 2 J. It follows that

X

\*

+ \*

X X

+

X

0 <

aihxi; wi =

aixi; w

=

jajjxj; w

=

jajjhxj; wi < 0;

i2I

i2I

j2J

j2J

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 Rd is d + 1.

Proof First, as in the proof of Theorem [9.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page122), it is easy to verify that the set of vectors 0; e1; : : : ; ed is shattered by the class of nonhomogenous halfspaces. Second, suppose that the vectors x1; : : : ; xd+2 are shattered by the class of non-homogenous halfspaces. But, using the reduction we have shown in the beginning of this chapter, it follows that there are d + 2 vectors in Rd+1 that are shattered by the class of homogenous halfspaces. But this contradicts Theorem [9.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page122). 

|  |  |
| --- | --- |
| 9.2 Linear Regression | 123 |
|  |  |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | r | r | r | r r | r r r r |
|  | r r | |  |  |  |
|  |  |  |  |  |  |

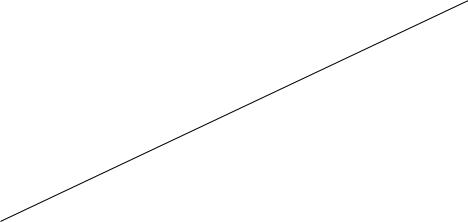


Figure 9.1 Linear regression for d = 1. For instance, the x-axis may denote the age of the baby, and the y-axis her weight.

9.2 Linear Regression

Linear regression is a common statistical tool for modeling the relationship be-tween 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 la-bel 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page123) 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,

Hreg = Ld = fx 7!wh; xi + b : w 2 Rd; b 2 Rg:

Next we need to de ne a loss function for regression. While in classi cation the de nition 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 de ne 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,

m

LS(h) = m1 X(h(xi) yi)2:

i=1

1. Linear Predictors

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, `(h; (x; y)) = jh(x) yj. The ERM rule for the absolute value loss function can be implemented using linear programming (see Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page128).)

Note that since linear regression is not a binary prediction task, we cannot an-alyze 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page57) in Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54)); namely, if we are happy with a representation of each element of the vector w and the bias b using a nite number of bits (say a 64 bits oating point representation), then the hypothesis class becomes nite and its size is at most 264(d+1). We can now rely on sample complexity bounds for nite hypothesis classes as described in Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54). Note, however, that to apply the sample complexity bounds from Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54) 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 hypoth-esis 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 Ld, is to nd

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| argmin L | (h | ) = argmin | 1 |  | m | ( w; x | y )2: |
|  |  |  |
| wS | w | w | m | | Xi | | iii |
| =1 | h |
|  |  |  |  |  |  |  |

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

m

2 X

(hw; xii yi)xi = 0:

m

i=1

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A = | m | xi xi>! | and b = | m | yixi: | (9.6) |
|  | X |  |  | Xi |  |  |
|  | i=1 |  |  | =1 |  |  |

|  |  |
| --- | --- |
| 9.2 Linear Regression | 125 |
|  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Or, in matrix form: |  |  |  |  |  |  |  |  |  |
| A = | 0 | x...1 : : : x...m | | 1 0 | x...1 : : : x...m | | | 1> | ; |
|  | B | .. | .. | C B | .. |  | .. | C |  |
|  | B | . | . | C B | . |  | . | C |  |
|  | @ |  |  | A @ |  |  |  | A |  |
| b = 0 x...1 : : : | | | x...m | 1 | y..1 | 1 | : |  |  |
|  | B |  |  | 0 | . |  |  |  |
|  |  |  | C B |  | C |  |  |  |
|  | B |  |  | C @ |  | A |  |  |  |
|  | @ | ... | ... | A | ym |  |  |  |  |

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

w = A 1 b:

(9.7)

(9.8)

The case in which A is not invertible requires a few standard tools from linear algebra, which are available in Appendix [C](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page430). 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 nd 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). De ne D+ to be the diagonal matrix such that Di;i+ = 0 if Di;i = 0 and otherwise Di;i+ = 1=Di;i. Now, de ne

A+ = V D+V > and w^ = A+b:

|  |  |
| --- | --- |
| Let vi denote the i'th column of V . Then, we have | X6 |
|  |
| Aw^ = AA+b = V DV >V D+V >b = V DD+V >b = | vivi>b: |
|  | i:Di;i=0 |

That is, Aw^ is the projection of b onto the span of those vectors vi for which Di;i 6= 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 Aw^ = b, which concludes our argument.

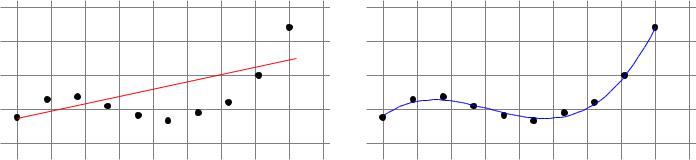
9.2.2 Linear Regression for Polynomial Regression Tasks

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) = a0 + a1x + a2x2 + + anxn

where (a0; : : : ; an) is a vector of coe cients of size n + 1. In the following we depict a training set that is better tted using a 3rd degree polynomial predictor than using a linear predictor.

1. Linear Predictors



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

Hpolyn = fx 7!p(x)g;

where p is a one dimensional polynomial of degree n, parameterized by a vector of coe cients (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 de ne the mapping : R ! Rn+1 such that (x) = (1; x; x2; : : : ; xn). Then we have that

p( (x)) = a0 + a1x + a2x2 + + anxn = ha; (x)i

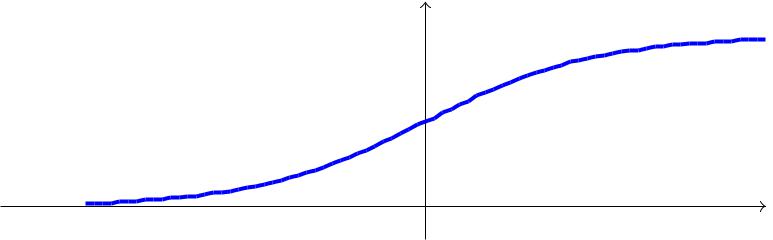
and we can nd the optimal vector of coe cients a by using the Least Squares algorithm as shown earlier.

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 classi cation 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 sig : R ! [0; 1] over the class of linear functions Ld. In particular, the sigmoid function used in logistic regression is the logistic function, de ned as

|  |  |  |  |
| --- | --- | --- | --- |
| sig(z) = | 1 | : | (9.9) |
| 1 + exp( z) |

The name \sigmoid" means \S-shaped," referring to the plot of this function, shown in the gure:



|  |  |
| --- | --- |
| 9.3 Logistic Regression | 127 |
|  |  |

The hypothesis class is therefore (where for simplicity we are using homogenous linear functions):

Hsig = sig Ld = fx 7! sig(hw; xi) : w 2 Rdg:

Note that when hw; xi is very large then sig(hw; xi) is close to 1, whereas if hw; xi is very small then sig(hw; xi) is close to 0. Recall that the prediction of the halfspace corresponding to a vector w is sign(hw; xi). Therefore, the predictions of the halfspace hypothesis and the logistic hypothesis are very similar whenever jhw; xij is large. However, when jhw; xij is close to 0 we have that sig(hw; xi)

12 . Intuitively, the logistic hypothesis is not sure about the value of the label so it guesses that the label is sign(hw; xi) with probability slightly larger than 50%. In contrast, the halfspace hypothesis always outputs a deterministic prediction of either 1 or 1, even if jhw; xij is very close to 0.

Next, we need to specify a loss function. That is, we should de ne how bad it is to predict some hw(x) 2 [0; 1] given that the true label is y 2 f 1g. Clearly, we would like that hw(x) would be large if y = 1 and that 1 hw(x) (i.e., the probability of predicting 1) would be large if y = 1. Note that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  | h |  | (x) = 1 |  |  | 1 | |  |  |  | = |  | exp( hw; xi) | | |  | = | 1 |  |  |  |  | : |  |
| w | 1 + exp( | |  | w; x | ) | |  | | | ) | 1 + exp( | w; x | | | ) |  |
|  |  |  | h |  | 1 + exp( | | h | w; x |  |  |  |
|  |  |  |  |  |  |  |  | i |  |  |  |  |  | i |  |  | h |  |  | i |  |  |  |
| Therefore, any reasonable loss function would increase monotonically with | | | | | | | | | | | | | | | | | | | | |  |  | 1 | | , |
| 1+exp(yhw;xi) | | | |

or equivalently, would increase monotonically with 1 + exp( yhw; xi). The lo-gistic loss function used in logistic regression penalizes hw based on the log of 1 + exp( yhw; xi) (recall that log is a monotonic function). That is,

`(hw; (x; y)) = log (1 + exp( yhw; xi)) :

Therefore, given a training set S = (x1; y1); : : : ; (xm; ym), the ERM problem associated with logistic regression is

m

argmin 1 X log (1 + exp( yihw; xii)) : (9.10)

w2Rd m i=1

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 e ciently 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.

The ERM problem associated with logistic regression (Equation ([9.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page127))) is iden-tical to the problem of nding a Maximum Likelihood Estimator, a well-known statistical approach for nding the parameters that maximize the joint probabil-ity of a given data set assuming a speci c parametric probability function. We will study the Maximum Likelihood approach in Chapter [24](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page342).

1. Linear Predictors

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 e cient 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 e cient learning.

Naturally, linear predictors are e ective whenever we assume, as prior knowl-edge, 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, Noviko 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)) = jh(x) yj, as a linear program; namely, show how to write the problem

m

X

min jhw; xii yij

w

i=1

as a linear program.

Hint: Start with proving that for any c 2 R,

jcj = min a s.t. c a and c a:

a 0

1. Show that the matrix A de ned in Equation ([9.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page124)) is invertible if and only if x1; : : : ; xm span Rd.
2. Show that Theorem [9.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page120) is tight in the following sense: For any positive integer

m, there exist a vector w 2 Rd (for some appropriate d) and a sequence of examples f(x1; y1); : : : ; (xm; ym)g such that the following hold:

R = maxi kxik 1.

kw k2 = m, and for all i m, yihxi; w i 1. Note that, using the notation

in Theorem [9.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page120), we therefore get

p

B = minfkwk : 8i 2 [m]; yihw; xii 1g m:

|  |  |
| --- | --- |
| 9.6 Exercises | 129 |
|  |  |

Thus, (BR)2 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.

1. (\*) Given any number m, nd an example of a sequence of labeled examples ((x1; y1); : : : ; (xm; ym)) 2 (R3 f 1; +1g)m on which the upper bound of Theorem [9.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page120) 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 a2 + b2 = R2 1. Let w be the vector (0; 0; 1). Now, go over the proof of the Perceptron's upper bound (Theorem [9.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page120)), see where we used inequalities

( ) rather than equalities (=), and gure out scenarios where the inequality actually holds with equality.

1. 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) + yixi for some > 0. Prove that the modi ed Per-ceptron 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.
2. In this problem, we will get bounds on the VC-dimension of the class of (closed) balls in Rd, that is,

Bd = fBv;r

where

Bv;r(x) =

1. v 2 Rd; r > 0g;
2. if kx vk r :

0 otherwise

1. Consider the mapping : Rd ! Rd+1 de ned by (x) = (x; kxk2). Show

that if x1; : : : ; xm are shattered by Bd then (x1); : : : ; (xm) 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(Bd)?

1. (\*) Find a set of d + 1 points in Rd that is shattered by Bd. Conclude that

d + 1 VCdim(Bd) d + 2:

1. 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 rst is the bias-complexity tradeo . We have seen (in Chapter [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page60)) 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 tradeo between these two considerations. The boosting paradigm allows the learner to have smooth control over this tradeo . 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page100), for many interesting concept classes the task of nding an ERM hypothesis may be computationally infeasible. A boosting algorithm ampli es 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 e ciently, 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 algo-rithm, 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 tradeo 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page137).

AdaBoost stemmed from the theoretical question of whether an e cient weak learner can be \boosted" into an e cient strong learner. This question was raised

|  |  |
| --- | --- |
| 10.1 Weak Learnability | 131 |
|  |  |

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 rst 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 de nition of PAC learning given in Chapter [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page43): A hypothesis class,

H, is PAC learnable if there exist mH : (0; 1)2 ! N and a learning algorithm

with the following property: For every ; 2 (0; 1), for every distribution D over X , and for every labeling function f : X ! f 1g, if the realizable assumption holds with respect to H; D; f, then when running the learning algorithm on m mH( ; ) 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 , L(D;f)(h) .

Furthermore, the fundamental theorem of learning theory (Theorem [6.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72) in Chapter [6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page67)) characterizes the family of learnable classes and states that every PAC learnable class can be learned using any ERM algorithm. However, the de nition of PAC learning and the fundamental theorem of learning theory ignores the computational aspect of learning. Indeed, as we have shown in Chapter [8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page100), 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 e ciently computable learning algorithm whose error is just slightly better than a random guess? This motivates the following de nition.

definition 10.1 ( -Weak-Learnability)

A learning algorithm, A, is a -weak-learner for a class H if there exists a func-tion mH : (0; 1) ! N such that for every 2 (0; 1), for every distribution D over X , and for every labeling function f : X ! f 1g, if the realizable assumption holds with respect to H; D; f, then when running the learning algorithm on m mH( ) 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 , L(D;f)(h) 1=2 .

A hypothesis class H is -weak-learnable if there exists a -weak-learner for that class.

1. Boosting

This de nition is almost identical to the de nition of PAC learning, which here we will call strong learning, with one crucial di erence: Strong learnability implies the ability to nd an arbitrarily good classi er (with error rate at most

for an arbitrarily small > 0). In weak learnability, however, we only need to

output a hypothesis whose error rate is at most 1=2 , 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 e cient weak learners than with e cient (full) PAC learners.

The fundamental theorem of learning (Theorem [6.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72)) states that if a hypothesis class H has a VC dimension d, then the sample complexity of PAC learning H

satis es mH( ; ) C1 d+log(1= ) , where C1 is a constant. Applying this with

= 1=2 we immediately obtain that if d = 1 then H is not -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 satis es the requirements of weak learning and can be implemented e ciently.

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 e ciently implementable.

For every sample that is labeled by some hypothesis from H, any ERMB hypothesis will have an error of at most 1=2 .

Then, the immediate question is whether we can boost an e cient weak learner into an e cient strong learner. In the next section we will show that this is indeed possible, but before that, let us show an example in which e cient weak learnability of a class H is possible using a base hypothesis class B.

Example 10.1 (Weak Learning of 3-Piece Classi ers Using Decision Stumps) Let X = R and let H be the class of 3-piece classi ers, namely, H = fh 1; 2;b : 1; 2 2 R; 1 < 2; b 2 f 1gg, where for every x,

(

+b if x < 1 or x > 2

h 1; 2;b(x) =

b if 1 x 2

An example hypothesis (for b = 1) is illustrated as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| + |  |  |  | + |
|  | 1 | | | 2 |

Let B be the class of Decision Stumps, that is, B = fx 7!sign(x ) b : 2 R; b 2 f 1gg. In the following we show that ERMB is a -weak learner for H, for = 1=12.

|  |  |
| --- | --- |
| 10.1 Weak Learnability | 133 |
|  |  |

To see that, we rst show that for every distribution that is consistent with H, there exists a decision stump with LD(h) 1=3. Indeed, just note that every classi er 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 2 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 (log(1= )= 2), then with probability of at least 1 , the ERMB rule returns a hypothesis with an error of at most 1=3 + . Setting = 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 -weak learner for H. We next show how to implement the ERM rule e ciently for decision stumps.

10.1.1 E cient Implementation of ERM for Decision Stumps

Let X = Rd and consider the base hypothesis class of decision stumps over Rd, namely,

HDS = fx 7!sign( xi) b : 2 R; i 2 [d]; b 2 f 1gg:

For simplicity, assume that b = 1; that is, we focus on all the hypotheses in HDS of the form sign( xi). Let S = ((x1; y1); : : : ; (xm; ym)) be a training set. We will show how to implement an ERM rule, namely, how to nd a decision stump that minimizes LS(h). Furthermore, since in the next section we will show that AdaBoost requires nding 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 Rm (that is, all elements of D are

P

nonnegative and i Di = 1). The weak learner we describe later receives D and S and outputs a decision stump h : X ! Y that minimizes the risk w.r.t. D,

m

X

LD(h) = Di1[h(xi)6=yi]:

i=1

Note that if D = (1=m; : : : ; 1=m) then LD(h) = LS(h).

Recall that each decision stump is parameterized by an index j 2 [d] and a threshold . Therefore, minimizing LD(h) amounts to solving the problem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j | [d] |  |  | 0 | Di [xi;j> ] + | Xi | Di [xi;j ] | 1 | : |  |
| 2 |  | min | | @ | i:Xi | 1 | A |  | (10.1) |
| min | |  | 1 |  |  |  |
|  |  |  | 2R |  | y =1 | i:y = 1 | |  |  |  |

Fix j 2 [d] and let us sort the examples so that x1;j x2;j : : : xm;j. De ne

j = fxi;j+xi+1;j : i 2 [m 1]g [ f(x1;j 1); (xm;j + 1)g. Note that for any 2 R

2

there exists 0 2 j that yields the same predictions for the sample S as the

1. Boosting

threshold . Therefore, instead of minimizing over 2 R we can minimize over

2 j.

This already gives us an e cient procedure: Choose j 2 [d] and 2 j that minimize the objective value of Equation ([10.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page133)). For every j and 2 j 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

2 (xi 1;j; xi;j). Let F ( ) be the value of the objective. Then, when we consider 0 2 (xi;j; xi+1;j) we have that

F ( 0) = F ( ) Di1[yi=1] + Di1[yi= 1] = F ( ) yiDi:

Therefore, we can calculate the objective at 0 in a constant time, given the objective at the previous threshold, . 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 S = (x1; y1); : : : ; (xm; ym)

distribution vector D

goal: Find j?; ? that solve Equation ([10.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page133))

initialize: F ? = 1

for j = 1; : : : ; d

sort S using the j'th coordinate, and denote

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | def |
| x1;j x2;jxm;j xm+1;j = xm;j + 1 | | | |
| if F | P | |  |
| F = | i:yi=1 Di | |  |
|  | < F ? | |  |
| F ? = F , ? = x1;j 1, j? = j | | | |
| for i = 1; : : : ; m | | |  |
| F = F yiDi | | |  |
| if | F < F ? and xi;j 6= xi+1;j | | |
| F ? = F , ? = | | 1 | (xi;j + xi+1;j), j? = j |
|  |
|  | 2 | |  |

output j?; ?

10.2 AdaBoost

AdaBoost (short for Adaptive Boosting) is an algorithm that has access to a weak learner and nds 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 rst de nes

|  |  |
| --- | --- |
| 10.2 AdaBoost | 135 |
|  |  |

a distribution over the examples in S, denoted D(t). That is, D(t) 2 Rm+ and

P

|  |  |  |
| --- | --- | --- |
| m | D(t) | = 1. Then, the booster passes the distribution D(t) and the sample S |
| i=1 | i |  |

to the weak learner. (That way, the weak learner can construct i.i.d. examples according to D(t) and f.) The weak learner is assumed to return a \weak" hypothesis, ht, whose error,

|  |  |  |  |
| --- | --- | --- | --- |
| def | def | m |  |
| (t) |  |
| t = LD(t) (ht) = | | Xi | 1[ht(xi)6=yi]; |
| Di |
|  |  | =1 |  |

is at most 12 (of course, there is a probability of at most that the weak learner

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| fails). Then, AdaBoost assigns a weight for ht as follows: wt = | 1 | log | |  | 1 | 1 . |
| 2 | t | |
| That is, the weight of ht is inversely proportional to the error of ht. | | | At the end | | | |
|  |  |  |  |

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" classi er 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 | | |  |  |  |  |  |  |  |  |  |
| for t = 1; : : : ; T : | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| invoke weak learner ht = WL(D(t); S) | | | | | | | | | | | | | | | | | | |  |
| compute t = | | |  | m | | | | D(t) 1 | | | | | |  |  |  |  |  |  |
| let wt = 2 log | | | Pt | |  | 1 | | | i |  |  | [yi6=ht(xi)] | | | | |  |  |  |
|  |  |  |  | i=1 | | | |  |  |  |  |  |  |
|  | (t+1) | | | |  |  |  | Di | | | exp( wtyiht(xi)) | | | | | | | |  |
| 1 | | | 1 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | (t) | |  |  |  |  |  |  |  |  |  |
| update Di | | | = | |  |  |  |  |  |  |  | | |  |  |  |  |  | for all i = 1; : : : ; m |
|  | P | | m |  | D | (t) | | | exp( |  | t j | t | T |
|  |  |  |  |  |  | j=1 | | j | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | w y | h | (xj)) |  |

P

output the hypothesis hs(x) = sign t=1 wtht(x) .

The following theorem shows that the training error of the output hypothesis decreases exponentially fast with the number of boosting rounds.

theorem 10.2 Let S be a training set and assume that at each iteration of AdaBoost, the weak learner returns a hypothesis for which t 1=2 . Then, the training error of the output hypothesis of AdaBoost is at most

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | |  | m |  |
|  |  |  | Xi | exp( 2 2 T ) : |
| LS(hs) = m | | | 1[hs(xi)6=yi] |
|  |  |  | =1 |  |

P

Proof For each t, denote ft = p t wphp. Therefore, the output of AdaBoost

e yift(xi)

1. Boosting

is fT . In addition, denote

m

Zt = m1 X e yift(xi):

i=1

Note that for any hypothesis we have that 1[h(x)6=y] e yh(x). Therefore, LS(fT ) ZT , so it su ces to show that ZT e 2 2T . To upper bound ZT we rewrite it as

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ZT = | ZT | = |  | ZT |  |  | ZT 1 |  | Z2 | |  | Z1 | ; | (10.2) |
|  |  |  |  |  |  |  |  |
| Z0 |  | ZT 1 |  | ZT 2 | Z1 | | Z0 |

where we used the fact that Z0 = 1 because f0 0. Therefore, it su ces to show that for every round t,

|  |  |  |
| --- | --- | --- |
| Zt+1 | 2 |  |
|  | e 2 : | (10.3) |
| Zt |

To do so, we rst note that using a simple inductive argument, for all t and i,

Di(t+1) = Pm e yjft(xj) :

j=1

Hence,

Zt+1

Zt

Pm e yift+1(xi)

= i=1

m

P e yjft(xj)

j=1

Pm e yift(xi)e yiwt+1ht+1(xi)

= i=1

m

1. e yjft(xj)

j=1

m

1. X Di(t+1)e yiwt+1ht+1(xi) i=1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| = e wt+1 | D(t+1) | + ewt+1 | X |  | D(t+1) |
|  | i tX |  | i |
|  | i |  |  |  |
|  | i:y h +1(xi)=1 |  | i:yiht+1(xi)= |  | 1 |

1. e wt+1 (1 t+1) + ewt+1 t+1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| = |  |  | (1 t+1) + p1= t+1 1 t+1 | | | | | | | | | | |
|  |  |  |  |  |  |
| p | |  |  |  |
| 1= t+1 1 | | |  |
| r | |  | |  | | |  |  | t+1 | s |  |  | |  |  | t+1 | |
|  |  |  |  |  |  |  | t+1 | | |  |
| 1 t+1 | | |  | |  |  |  |
| = |  |  | t+1 | | (1 | |  |  |  | ) + |  | 1 t+1 | |  |  |  |  |
|  |  |  |  |  |  |  | |  |  |

p

= 2 t+1(1 t+1):

By our assumption, t+1 12 . Since the function g(a) = a(1 a) is mono-tonically increasing in [0; 1=2], we obtain that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2p t+1 | | (1 t+1) 2 | | s |  |  |  |  |  |  | = p1 4 2: | | |
| 2 | |  | 2 + | | |
|  |  |  |  | 1 | | |  | 1 | | |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 10.3 Linear Combinations of Base Hypotheses | | | | | | | | | | | 137 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 2 2 |  |  |  |  | e a we have that | |  |  |  |  |  | e 4 2=2 = |
|  |  | a | 1 | |  | 4 2 |
| Finally, using the inequality 1 | | |  |  |  |  |
| e |  | . This shows that Equation ([10.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page136)) holds and thus | | | | | concludes our proof. | | | | | | |
|  |  | p | |  |  |  |  |

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 e ciently (as happens in the case of ERM with respect to decision stumps) then the total training process will be e cient.

Remark 10.2 Theorem [10.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page135) assumes that at each iteration of AdaBoost, the weak learner returns a hypothesis with weighted sample error of at most 1=2 . According to the de nition of a weak learner, it can fail with probability . Using the union bound, the probability that the weak learner will not fail at all of the iterations is at least 1 T . As we show in Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page142), the dependence of the sample complexity on can always be logarithmic in 1= , and therefore invoking the weak learner with a very small is not problematic. We can therefore assume that T 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., = 0). This is the case, for example, in the weak learner that nds the minimum value of LD(h) for decision stumps, as described in the previous section.

Theorem [10.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page135) 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 following class:

(

T

!

)

X

t

That is, each h 2 L(B; T ) is parameterized by T base hypotheses from B and by a vector w 2 RT . The prediction of such an h on an instance x is ob-tained by rst applying the T base hypotheses to construct the vector (x) =

1. Boosting

(h1(x); : : : ; hT (x)) 2 RT , and then applying the (homogenous) halfspace de ned by w on (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 Ad-aBoost 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 tradeo .

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,

HDS1 = fx 7!sign(x ) b : 2 R; b 2 f 1gg:

Note that in this one dimensional case, HDS1 is in fact equivalent to (nonho-mogenous) 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 1 = 0 < 1 < 2 < < r = 1 such that

r

X

gr(x) = i1[x2( i 1; i]] 8i; i 2 f 1g:

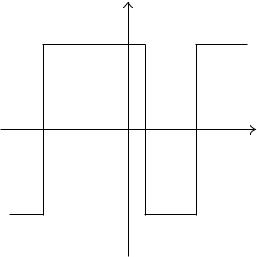
i=1

Denote by Gr the class of all such piece-wise constant classi ers 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 classi ers with at most

1. pieces.

Indeed, without loss of generality consider any g 2 GT with t = ( 1)t. This implies that if x is in the interval ( t 1; t], then g(x) = ( 1)t. For example:



Now, the function

|  |  |  |  |
| --- | --- | --- | --- |
| h(x) = sign | =1 wt sign(x t 1)! | ; | (10.5) |
|  | T |  |  |

X

t

where w1 = 0:5 and for t > 1, wt = ( 1)t, is in L(HDS1; T ) and is equal to g (see Exercise [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page142)).

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 10.3 Linear Combinations of Base Hypotheses | 139 |
|  |  | | |
|  | 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 tradeo : 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 | | |
|  | ~ | ~ |  |
|  | by O(VCdim(B) T ) (the O notation ignores constants and logarithmic factors). | | |
|  | lemma 10.3 | Let B be a base class and let L(B; T ) be as de ned in Equa- | |
|  | tion ([10.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page137)). 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 = fx1; : : : ; xmg be a set that is shat-tered by L(B; T ). Each labeling of C by h 2 L(B; T ) is obtained by rst choos-ing h1; : : : ; hT 2 B and then applying a halfspace hypothesis over the vector (h1(x); : : : ; hT (x)). By Sauer's lemma, there are at most (em=d)d di erent di-chotomies (i.e., labelings) induced by B over C. Therefore, we need to choose

1. hypotheses, out of at most (em=d)d di erent 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

2m m(d+1)T :

Therefore,

(d + 1)T

m log(m) :

Lemma [A.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419) in Chapter [A](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419) tells us that a necessary condition for the above to hold is that

m 2 (d + 1)T log (d + 1)T (d + 1)T (3 log((d + 1)T ) + 2);

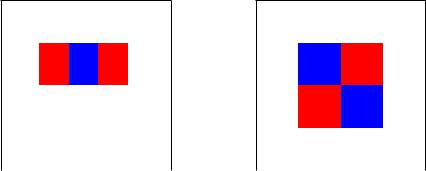
log(2) log(2)

which concludes our proof.

In Exercise [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page143) we show that for some base classes, B, it also holds that VCdim(L(B; T )) (VCdim(B) T ).

1. Boosting

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A |  |  |  |  |  |  |  |  |  | B |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C |  |  |  | D |
|  |  |

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 di erence 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 di erence 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 classi er, h : X ! f 1g, 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 : R24;24 ! 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 2 fA; B; C; Dg. Each type corresponds to a mask, as depicted in Figure [10.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page140).

To calculate g we stretch the mask t to t 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 rst 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 rst step very

|  |  |
| --- | --- |
| 10.5 Summary | 141 |
|  |  |

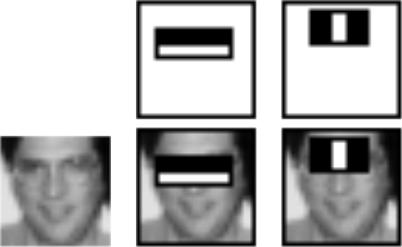


Figure 5: The first and second features selected by AdaBoost. The two features are shown in the top row Figure 10.2 The rst and second features selected by AdaBoost, as implemented by

and then overlayed on a typical training face in the bottom row. The first feature measures the difference in Viola intensityandJonesbetween.Thetheregiontwo offeaturesyesthe andareregionshownacrossinthetheuppertopcheeksrow. Theandfeathenurecapitalizesoverlaidon theon a

typicalobsetrainingvationthatfacetheeyeinregionthe bottomisofendarkerrowthan.Thetcheeksrst. featureThesecondmeasuresfaturecomparthes dithe intensitieserence in

in the eye regions to the intensity across the bridge of the nose.

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

directly increases computation time.

cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of the nose.

**4 The Attentional Cascade**

e cientlyThissectionby describesapreprocessinganalgorithmforstepconstructingwhichacasadeweofclassifierscalculatewhichtheachievesintegralincreased imagedetec- of

tion performance while radically reducing computation time. The key insight is that smaller, and therefore

each image in the training set. See Exercise [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page143) for details.

more efficient, boosted classifiers can be constructed which reject many of the negative sub-windows while

In Figure [10.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page141) we depict the rst two features selected by AdaBoost when

detecting almost all positive instances. Simpler classifiers are used to reject the majority of sub-windows

running it with the base features proposed by Viola and Jones.

before more complex classifiers are called upon to achieve low false positive rates.

Stages in the cascade are constructed by training classifiers using AdaBoost. Starting with a two-feature strong classifier, an effective face filter can be obtained by adjusting the strong classifier threshold to min-

10.5 Summaryiizefalse negatives. The initial AdaBoost threshold, , is designed to yield a low error rate on the training data. A lower threshold yields higher detection rates and higher false positive rates. Based on

performance measured using a validation training set, the two-feature classifier can be adjusted to detect

Boosting is a method for amplifying the accuracy of weak learners. In this chapter

100% of the faces with a false positive rate of 40%. See Figure 5 for a description of the two features used

we described the AdaBoost algorithm. We have shown that after T iterations of

in this classifier.

AdaBoost, it returns a hypothesis from the class L(B; T ), obtained by composing

The detection performance of the two-feature classifier is far from acceptable as an object detection

a linear classi er on T hypotheses from a base class B. We have demonstrated

system. Nevertheless the classifier can significantly reduce the number sub-windows th t need further pro-

how thecessingparameterwithveryfewToperations:controls the tradeo between approximation and estimation

errors. In the next chapter we will study how to tune parameters such as T , based

1. Evaluate the rectangle features (requires between 6 and 9 array references per feature).

on the data.

2. Compute the weak classifier for each feature (requires one threshold operation per feature).

11

10.6 Bibliographic Remarks

As mentioned before, boosting stemmed from the theoretical question of whether an e cient weak learner can be \boosted" into an e cient strong learner (Kearns

1. 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

1. Boosting

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 classi er 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page202). It can also be viewed as a forward greedy selection algorithm, a topic that will be presented in Chapter [25](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page357). 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 eld has produced.

10.7 Exercises

1. Boosting the Con dence: Let A be an algorithm that guarantees the fol-

lowing: There exist some constant 0 2 (0; 1) and a function mH : (0; 1) ! N such that for every 2 (0; 1), if m mH( ) then for every distribution D it holds that with probability of at least 1 0, LD(A(S)) minh2H LD(h) + .

Suggest a procedure that relies on A and learns H in the usual agnostic PAC learning model and has a sample complexity of

|  |  |  |  |
| --- | --- | --- | --- |
| mH( ; ) k mH( ) + 2 | | 2 | ; |
|  |  | log(4k= ) |  |
|  |  |  |  |

where

k = dlog( )= log( 0)e:

Hint: Divide the data into k + 1 chunks, where each of the rst k chunks is of size mH( ) examples. Train the rst k chunks using A. Argue that the probability that for all of these chunks we have LD(A(S)) > minh2H LD(h)+ is at most 0k =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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page57)).

1. Prove that the function h given in Equation ([10.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page138)) equals the piece-wise con-stant function de ned according to the same thresholds as h.
2. 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 nd some rigorous justi cation for this argument.

|  |  |
| --- | --- |
| 10.7 Exercises | 143 |
|  |  |

Show that the error of ht w.r.t. the distribution D(t+1) is exactly 1=2. That is, show that for every t 2 [T ]

m

* + - 1. Di(t+1) 1[yi6=ht(xi)] = 1=2:

i=1

1. 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 1, VCdim(B) VCdim(L(B; T )). Find a class B for which VCdim(B) = VCdim(L(B; T ))

for every T 1.

Hint: Take X to be a nite set.

* 1. Let Bd be the class of decision stumps over Rd. Prove that log(d) VCdim(Bd) 5 + 2 log(d).

Hints:

For the upper bound, rely on Exercise [11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page80).

For the lower bound, assume d = 2k. Let A be a k d matrix whose columns are all the d binary vectors in f 1gk. The rows of A form a set of k vectors in Rd. Show that this set is shattered by decision stumps over Rd.

* 1. Let T 1 be any integer. Prove that VCdim(L(Bd; T )) 0:5 T log(d).

Hint: Construct a set of T2 k instances by taking the rows of the matrix A

from the previous question, and the rows of the matrices 2A; 3A; 4A; : : : ; T2 A. Show that the resulting set is shattered by L(Bd; T ).

1. E ciently Calculating the Viola and Jones Features Using an Inte-gral Image: Let A be a 24 24 matrix representing an image. The integral

P

image of A, denoted by I(A), is the matrix B such that Bi;j = i0 i;j0 j Ai;j.

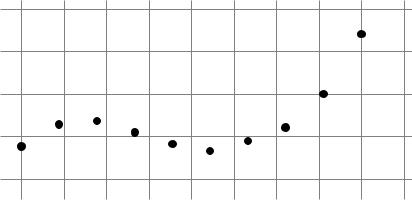
Show that I(A) can be calculated from A in time linear in the size of A. Show how every Viola and Jones feature can be calculated from I(A) in a

constant amount of time (that is, the runtime does not depend on the size of the rectangle de ning the feature).

1. Model Selection and Validation

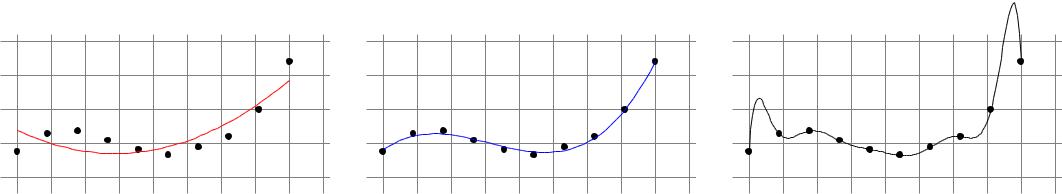
In the previous chapter we have described the AdaBoost algorithm and have shown how the parameter T of AdaBoost controls the bias-complexity trade-o . But, how do we set T in practice? More generally, when approaching some practical problem, we usually can think of several algorithms that may yield a 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 ! R. Suppose that we obtain a training set as depicted in the gure.



We can consider tting a polynomial to the data, as described in Chapter [9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117). However, we might be uncertain regarding which degree d would give the best results for our data set: A small degree may not t the data well (i.e., it will have a large approximation error), whereas a high degree may lead to over tting (i.e., it will have a large estimation error). In the following we depict the result of tting 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



|  |  |
| --- | --- |
| 11.1 Model Selection Using SRM | 145 |
|  |  |

In this chapter we will present two approaches for model selection. The rst approach is based on the Structural Risk Minimization (SRM) paradigm we have described and analyzed in Chapter [7.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page85). SRM is particularly useful when a learning algorithm depends on a parameter that controls the bias-complexity tradeo (such as the degree of the tted 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 nd the right balance between approxi-mation and estimation errors. More generally, if our learning algorithm fails to nd a predictor with a small risk, it is important to understand whether we su er from over tting or under tting. In Section [11.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page151) 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page85). Here we show how SRM can be used for tuning the tradeo between bias and complexity without deciding on a speci c hypothesis class in advance. Consider a countable sequence of hypothesis classes H1; H2; H3; : : :. For example, in the problem of polynomial regression mentioned, we can take Hd to be the set of polynomials of degree at most d. Another example is taking Hd to be the class L(B; d) used by AdaBoost, as described in the previous chapter.

We assume that for every d, the class Hd enjoys the uniform convergence property (see De nition [4.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page55) in Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54)) with a sample complexity function of the form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| UC | ( ; ) | g(d) log(1= ) |  |  |
| mHd | 2 | ; | (11.1) |

where g : N ! R is some monotonically increasing function. For example, in the case of binary classi cation problems, we can take g(d) to be the VC-dimension of the class Hd multiplied by a universal constant (the one appearing in the fundamental theorem of learning; see Theorem [6.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72)). 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 , for every

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d 2 N and h 2 Hd, | r |  |  |  |  |  |  |  |
| LD(h) LS(h) + |  |  |  |  |  |  | (11.2) |
| g(d)(log(1= ) + 2 | | m | : | | |
|  |  |  |  | log(d) + log( 2 | =6)) |  |  |  |

This bound, which follows directly from Theorem [7.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page86), shows that for every d and every h 2 Hd, the true risk is bounded by two terms { the empirical risk, LS(h),

1. Model Selection and Validation

and a complexity term that depends on d. The SRM rule will search for d and h 2 Hd that minimize the right-hand side of Equation ([11.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page145)).

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 re ected 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page145)) 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 pre-dictor 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 dis-tributions. 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page147).

11.2.1 Hold Out Set

The simplest way to estimate the true error of a predictor h is by sampling an ad-ditional set of examples, independent of the training set, and using the empirical error on this validation set as our estimator. Formally, let V = (x1; y1); : : : ; (xmv ; ymv ) be a set of fresh mv examples that are sampled according to D (independently of the m examples of the training set S). Using Hoe ding's inequality ( Lemma [4.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page56)) 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 2 (0; 1), with probability of at least 1 over the choice of a validation set V of size mv we have

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| jLV (h) LD(h)j | s |  |  |  | : |
|  | 2 mv | |
|  |  |  | log(2= ) | |  |
|  |  |  |  |  |  |

The bound in Theorem [11.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page146) 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

|  |  |
| --- | --- |
| 11.2 Validation | 147 |
|  |  |

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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72)) we obtain the bound

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LD(h) LS(h) + r |  |  |  |  | ; |
| C d + | | m | |
|  |  |  | log(1= ) | | |
|  |  |  |  |  |  |

where C is the constant appearing in Theorem [6.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72). In contrast, from Theo-rem [11.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page146) we obtain the bound

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LD(h) LV (h) + | s |  |  |  | : |
|  | 2mv | |
|  |  |  | log(2= ) | |  |
|  |  |  |  |  |  |

Therefore, taking mv 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 rst train di erent algorithms (or the same algorithm with di erent parameters) on the given training set. Let H = fh1; : : : ; hrg be the set of all output predictors of the di erent algorithms. For example, in the case of training polynomial regressors, we would have each hr 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 ERMH over the validation set.

This process is very similar to learning a nite hypothesis class. The only di erence is that H is not xed ahead of time but rather depends on the train-ing 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 nite hypothesis classes holds here as well. In particular, combining Theorem [11.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page146) with the union bound we obtain:

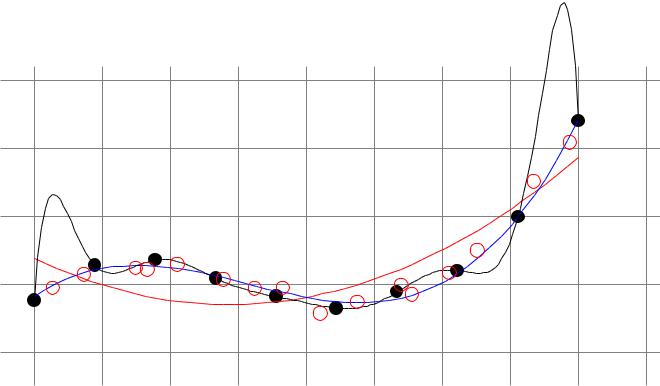
theorem 11.2 Let H = fh1; : : : ; hrg be an arbitrary set of predictors and assume that the loss function is in [0; 1]. Assume that a validation set V of size mv is sampled independent of H. Then, with probability of at least 1 over the choice of V we have

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 |  | 2 H |  | j | D |  | V |  | j s |  |  |  |
|  |  |  | 2 mv | | |
|  | h |  | ; |  | L (h) | L |  | (h) |  | log(2jHj= ) | : | |
|  |  |  |  |  |  |

1. Model Selection and Validation

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 jHj that is large relative to the size of the validation set) then we're in danger of over tting.

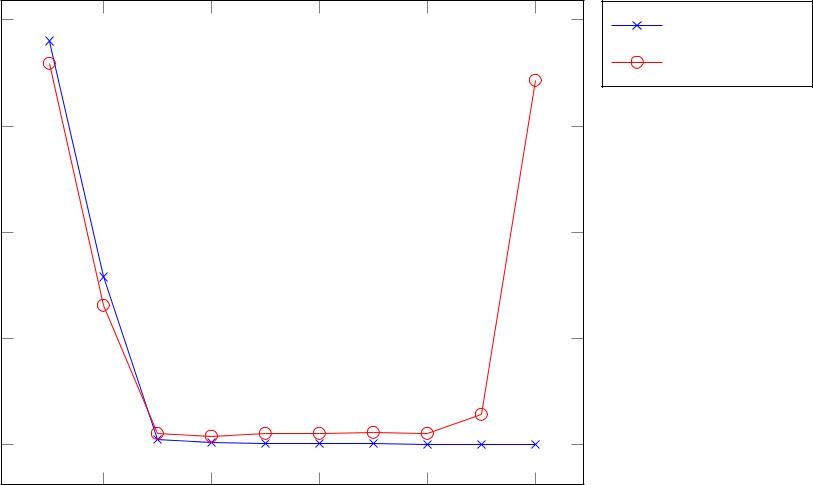
To illustrate how validation is useful for model selection, consider again the example of tting a one dimensional polynomial as described in the beginning of this chapter. In the following we depict the same training set, with ERM polynomials of degree 2, 3, and 10, but this time we also depict an additional validation set (marked as red, un lled 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 func-tion of the complexity of the model considered. For example, for the polynomial tting problem mentioned previously, the curve will look like:

|  |  |
| --- | --- |
| 11.2 Validation | 149 |
|  |  |



|  |
| --- |
| error |

|  |  |
| --- | --- |
| 0:4 | train |
|  | validation |
| 0:3 |  |

0:2

0:1

0

2 4 6 8 10

d

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 rst decreases but then starts to increase, which indicates that we are starting to su er from over tting.

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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page171) 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 ner grid to search over. It is important to verify that we are in the relevant regime. For example, in the polynomial tting problem described, if we start searching degrees from the set of values f1; 10; 20g and do not employ a ner 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

1. Model Selection and Validation

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, , 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 2 . 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 = (x1; y1); : : : ; (xm; ym)

set of parameter values

learning algorithm A

integer k

partition S into S1; S2; : : : ; Sk

foreach 2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| for i = 1 : : : k | | | | |  |  |  |  |  |
| hi; = A(S nk Si; ) | | | | | | |  |  |  |
| error( ) = |  | 1 | |  | L |  | (h |  | ) |
| k Pi=1 | | | Si | i; |
| output |  |  |  |  |
| ? = argmin |  | | [error( )] | | | |  |  |  |
|  |  |  |  |  |  |  |  |

h ? = A(S; ?)

The cross validation method often works very well in practice. However, it might sometime fail, as the arti cial example given in Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page154) shows. Rig-orously 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page258)) the cross validation proce-dure 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page171)).

11.2.5 Train-Validation-Test Split

In most practical applications, we split the available examples into three sets. The rst 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 | 151 |
|  |  |

11.3 What to Do If Learning Fails

Consider the following scenario: You were given a learning task and have ap-proached it with a choice of a hypothesis class, a learning algorithm, and param-eters. 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 unsatis-factory. What went wrong then, and what should you do next?

There are many elements that can be \ xed." The main approaches are listed in the following:

Get a larger sample

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 nd the best remedy, it is essential rst to understand the cause of the bad performance. Recall that in Chapter [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page60) we decomposed the true er-ror of the learned predictor into approximation error and estimation error. The approximation error is de ned to be LD(h?) for some h? 2 argminh2H LD(h), while the estimation error is de ned to be LD(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 bene cial 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 di erent hypothesis class). We can also consider applying the same hypothesis class but on a di erent feature representation of the data (see Chapter [25](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page357)).

The estimation error of the class does depend on the sample size. Therefore, if we have a large estimation error we can make an e ort 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 nding the best remedy. In the previous section we saw how to estimate LD(hS) using the empirical risk on a validation set. However, it is more di cult to estimate the approximation error of the class.

1. Model Selection and Validation

Instead, we give a di erent error decomposition, one that can be estimated from the train and validation sets.

LD(hS) = (LD(hS) LV (hS)) + (LV (hS) LS(hS)) + LS(hS):

The rst term, (LD(hS) LV (hS)), can be bounded quite tightly using Theo-rem [11.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page146). Intuitively, when the second term, (LV (hS) LS(hS)), is large we say that our algorithm su ers from \over tting" while when the empirical risk term, LS(hS), is large we say that our algorithm su ers from \under tting." Note that these two terms are not necessarily good estimates of the estimation and ap-

proximation 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, LS(hS), and the approximation error, LD(h?), can be signi cantly di erent. Nevertheless, as we show later, the values of LS(hS) and (LV (hS) LS(hS)) still provide us useful information.

Consider rst the case in which LS(hS) is large. We can write

LS(hS) = (LS(hS) LS(h?)) + (LS(h?) LD(h?)) + LD(h?):

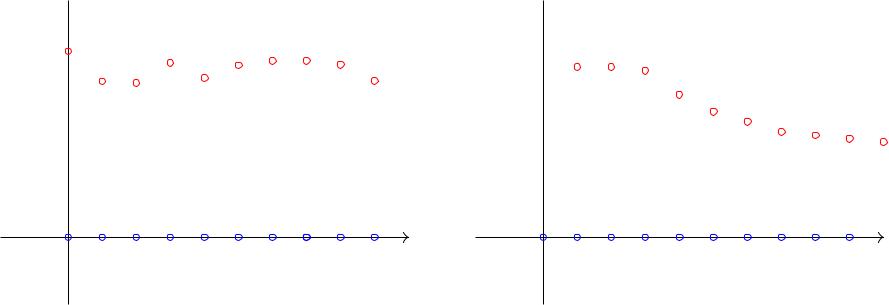
When hS is an ERMH hypothesis we have that LS(hS) LS(h?) 0. In addition, since h? does not depend on S, the term (LS(h?) LD(h?)) can be bounded quite tightly (as in Theorem [11.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page146)). The last term is the 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 nding an ERM hypothesis is computationally hard, and our algorithm applies some heuristic trying to nd 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 di erent 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 rst 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

|  |  |
| --- | --- |
| 11.3 What to Do If Learning Fails | 153 |
|  |  |

error error



validation error

train error

train error

|  |  |
| --- | --- |
| m | m |

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 pre xes of the data of increasing sizes. For example, we can rst train the algorithm on the rst 10% of the examples, then on 20% of them, and so on. For each pre x we calculate the training error (on the pre x the algorithm is being trained on) and the validation error (on a prede ned validation set). Such learning curves can help us distinguish between the two aforementioned scenarios. In the rst scenario we expect the validation error to be approximately 1=2 for all pre xes, 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page153).

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 nite, when the sample size goes to in nity, 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 approx-imation error, or at least to get a rough estimate on an interval in which the approximation error resides.

Getting back to the problem of nding 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 su cient for learning the class H. We can then plot a learning curve. If we see that the

1. Model Selection and Validation

validation error is starting to decrease then the best solution is to increase the number of examples (if we can a ord 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 rst 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page148)).
  2. If the training error is excessively large consider enlarging the hypothesis class, 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 hy-pothesis 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 nd 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 di erence 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

|  |  |
| --- | --- |
| 11.5 Exercises | 155 |
|  |  |

Divide the m examples into a training set of size (1 )m and a validation set of size m, for some 2 (0; 1). Then, apply the approach of model

selection using validation. That is, rst train each class Hi on the (1 )m training examples using the ERM rule with respect to Hi, and let

h^1; : : : ; h^k be the resulting hypotheses. Second, apply the ERM rule with

respect to the nite class fh^1; : : : ; h^kg on the m validation examples.

Describe scenarios in which the rst method is better than the second and vice versa.

1. 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 e ciently 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 e ciently. 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 de nitions of convexity. Besides convexity, we will de ne Lipschitzness and smoothness, which are additional properties of the loss function that facilitate successful learning. We next turn to de ning convex learning problems and demonstrate the necessity for further constraints such as Boundedness and Lipschitzness or Smoothness. We de ne 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page167), 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 e cient solutions but might increase the risk of the learned predictor.

12.1 Convexity, Lipschitzness, and Smoothness

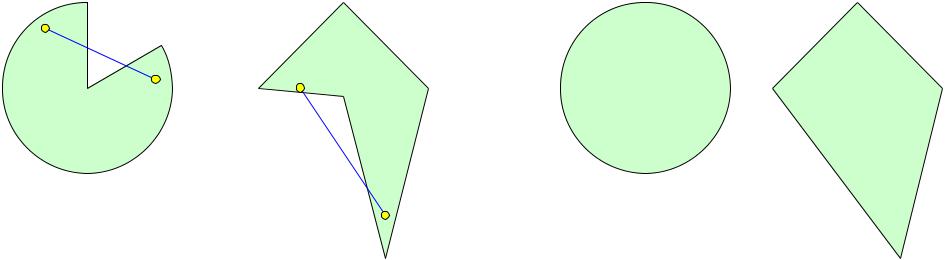
12.1.1 Convexity

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 2 [0; 1] we have that u + (1 )v 2 C.

Examples of convex and nonconvex sets in R2 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.

|  |  |
| --- | --- |
| 12.1 Convexity, Lipschitzness, and Smoothness | 157 |
|  |  |

non-convex convex



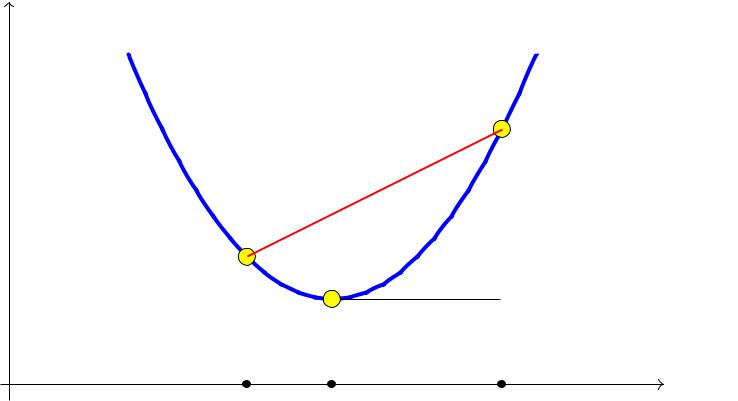
Given 2 [0; 1], the combination, u + (1 )v of the points u; v is called a convex combination.

definition 12.2 (Convex Function) Let C be a convex set. A function f :

C ! R is convex if for every u; v 2 C and 2 [0; 1],

f( u + (1 )v) f(u) + (1 )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 ! R, is depicted in the following.



f(v)

 f(u) + (1 )f(v)

1. (u)

 f( u + (1 )v)

1. v

u + (1 )v

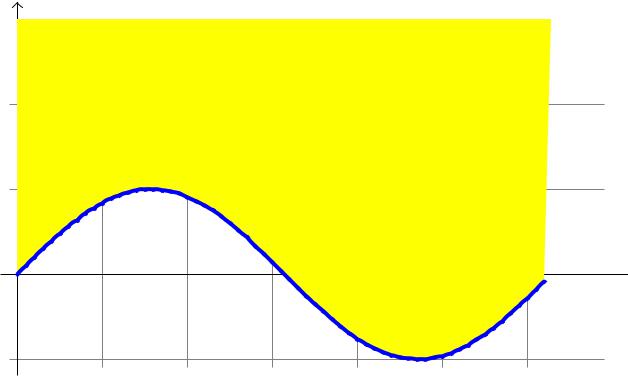
The epigraph of a function f is the set

|  |  |
| --- | --- |
| epigraph(f) = f(x; ) : f(x) g: | (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 ! R, along with its epigraph, is given in the following.

1. Convex Learning Problems

f(x)



 x

An important property of convex functions is that every local minimum of the function is also a global minimum. Formally, let B(u; r) = fv : kv uk r g 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 2 B(u; r) we have f(v) f(u). It follows that for any v (not necessarily in B), there is a small enough > 0 such that u + (v u) 2 B(u; r) and therefore

|  |  |
| --- | --- |
| f(u) f(u + (v u)) : | (12.2) |

If f is convex, we also have that

|  |  |
| --- | --- |
| f(u + (v u)) = f( v + (1 )u) (1 )f(u) + f(v) : | (12.3) |

Combining these two equations and rearranging terms, we conclude that f(u) f(v). Since this holds for every v, it follows that f(u) is also a global minimum of f.

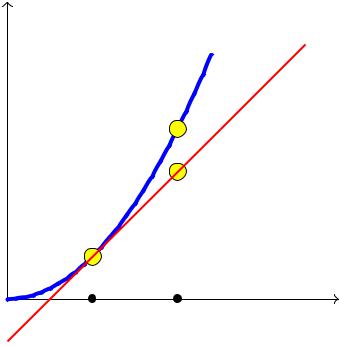
Another important property of convex functions is that for every w we can construct a tangent to f at w that lies below f everywhere. If f is di erentiable, this tangent is the linear function l(u) = f(w) + hrf(w); u wi, where rf(w) is the gradient of f at w, namely, the vector of partial derivatives of f, rf(w) =

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | @f(w) | | | ; : : : ; | @f(w) | . That is, for convex di erentiable functions, |  |
|  | @w1 |  | @wd |  |
|  |  |  |  |  |  | 8u; f(u) f(w) + hrf(w); u wi: | (12.4) |

In Chapter [14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page184) we will generalize this inequality to nondi erentiable functions.

An illustration of Equation ([12.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page158)) is given in the following.

|  |  |
| --- | --- |
| 12.1 Convexity, Lipschitzness, and Smoothness | 159 |
|  |  |



1. (u)
2. (w)
3. u

If f is a scalar di erentiable function, there is an easy way to check if it is convex.

lemma 12.3 Let f : R ! R be a scalar twice di erential function, and let f0; f00 be its rst and second derivatives, respectively. Then, the following are equivalent:

1. f is convex
2. f0 is monotonically nondecreasing
3. f00 is nonnegative

Example 12.1

The scalar function f(x) = x2 is convex. To see this, note that f0(x) = 2x and f00(x) = 2 > 0.

The scalar function f(x) = log(1 + exp(x)) is convex. To see this, observe that

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f0(x) = | exp(x) |  | = |  | 1 | . This is a monotonically increasing function |
| 1+exp(x) | | exp( x)+1 | |
|  |  |  |

since the exponent function is a monotonically increasing function.

The following claim shows that the composition of a convex scalar function with a linear function yields a convex vector-valued function.

claim 12.4 Assume that f : Rd ! R can be written as f(w) = g(hw; xi + y),

for some x 2 Rd; y 2 R, and g : R ! R. Then, convexity of g implies the convexity of f.

Proof Let w1; w2 2 Rd and 2 [0; 1]. We have

f( w1 + (1 )w2) = g(h w1 + (1 )w2; xi + y)

1. g( hw1; xi + (1 )hw2; xi + y)
2. g( (hw1; xi + y) + (1 )(hw2; xi + y)) g(hw1; xi + y) + (1 )g(hw2; xi + y);

where the last inequality follows from the convexity of g.

Example 12.2

1. Convex Learning Problems

Given some x 2 Rd and y 2 R, let f : Rd ! R be de ned as f(w) = (hw; xi 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 2 Rd and y 2 f 1g, let f : Rd ! R be de ned as f(w) = log(1 + exp( yhw; xi)). 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 fi : Rd ! R be a convex function. The following functions from Rd to R are also convex.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | g(x) = maxi2[r] fi(x) | | |  |  |  |
|  |  |  |  | r |  |  |  |
|  | g(x) = | | i=1 wifi(x), where for all i, wi 0. | | |  |
|  | Proof | | The rst claim follows by | |  |  |  |
|  |  | P |  |  |  |
|  |  |  |  | g( u + (1 )v) = maxi | | fi( u + (1 )v) |  |
|  |  |  |  | maxi | | [ fi(u) + (1 )fi(v)] | |
|  |  |  |  | maxi fi(u) + (1 ) maxi fi(v) | | | |
|  |  |  |  | = g(u) + (1 )g(v): | | |  |
|  | For the second claim | | | |  |  |  |
|  |  |  |  | g( u + (1 )v) = X wifi( u + (1 )v) | | |  |
|  |  |  |  | i |  |  |  |
|  |  |  |  | X wi [ fi(u) + (1 )fi(v)] | | | |
|  |  |  |  | i | wifi(u) + (1 ) Xi | |  |
|  |  |  |  | = Xi | wifi(v) |
|  |  |  |  | = g(u) + (1 )g(v): | | |  |
|  | Example 12.3 The function g(x) = jxj is convex. To see this, note that g(x) = | | | | | | |
|  | maxfx; xg and that both the function f1(x) = x and f2(x) = x are convex. | | | | | | |
| 12.1.2 | Lipschitzness | | | |  |  |  |
|  | The de nition of Lipschitzness below is with respect to the Euclidean norm over | | | | | | |
|  | Rd. However, it is possible to de ne Lipschitzness with respect to any norm. | | | | | | |

definition 12.6 (Lipschitzness) Let C Rd. A function f : Rd ! Rk is -Lipschitz over C if for every w1; w2 2 C we have that kf(w1) f(w2)k

kw1 w2k.

|  |  |
| --- | --- |
| 12.1 Convexity, Lipschitzness, and Smoothness | 161 |
|  |  |

Intuitively, a Lipschitz function cannot change too fast. Note that if f : R ! R is di erentiable, then by the mean value theorem we have

f(w1) f(w2) = f0(u)(w1 w2) ;

where u is some point between w1 and w2. It follows that if the derivative of f is everywhere bounded (in absolute value) by , then the function is -Lipschitz.

Example 12.4

The function f(x) = jxj is 1-Lipschitz over R. This follows from the triangle inequality: For every x1; x2,

jx1j jx2j = jx1 x2 + x2j jx2j jx1 x2j + jx2j jx2j = jx1 x2j:

Since this holds for both x1; x2 and x2; x1, we obtain that jjx1j jx2jj jx1 x2j.

The function f(x) = log(1 + exp(x)) is 1-Lipschitz over R. To see this, observe

that

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| jf0(x)j = |  | 1 + exp(x) | = | exp( | |  | x) + 1 | | 1: |
|  |  | exp(x) |  |  | | 1 | |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

The function f(x) = x2 is not -Lipschitz over R for any . To see this, take x1 = 0 and x2 = 1 + , then

f(x2) f(x1) = (1 + )2 > (1 + ) = jx2 x1j:

However, this function is -Lipschitz over the set C = fx : jxj =2g.

Indeed, for any x1; x2 2 C we have

jx21 x22j = jx1 + x2j jx1 x2j 2( =2) jx1 x2j = jx1 x2j:

The linear function f : Rd ! R de ned by f(w) = hv; wi + b where v 2 Rd is kvk-Lipschitz. Indeed, using Cauchy-Schwartz inequality,

jf(w1) f(w2)j = jhv; w1 w2ij kvk kw1 w2k:

The following claim shows that composition of Lipschitz functions preserves Lipschitzness.

claim 12.7 Let f(x) = g1(g2(x)), where g1 is 1-Lipschitz and g2 is 2-Lipschitz. Then, f is ( 1 2)-Lipschitz. In particular, if g2 is the linear function, g2(x) = hv; xi + b, for some v 2 Rd; b 2 R, then f is ( 1 kvk)-Lipschitz.

Proof

jf(w1) f(w2)j = jg1(g2(w1)) g1(g2(w2))j

1kg2(w1) g2(w2)k 1 2 kw1 w2k:

1. Convex Learning Problems

12.1.3 Smoothness

The de nition of a smooth function relies on the notion of gradient. Recall that the gradient of a di erentiable function f : Rd ! R at w, denoted rf(w), is the

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| vector of partial derivatives of f, namely, rf(w) = | @f(w) | ; : : : ; | |  | @f(w) | | . |
| @w1 |  | @wd |  |
| definition 12.8 (Smoothness) A di erentiable function f | | | : Rd ! R is - | | | | |

smooth if its gradient is -Lipschitz; namely, for all v; w we have krf(v) rf(w)k kv wk.

|  |  |  |  |
| --- | --- | --- | --- |
| It is possible to show that smoothness implies that for all v; w we have | | |  |
|  |  | |  |
| f(v) f(w) + hrf(w); v wi + |  | kv wk2 : | (12.5) |
| 2 |

Recall that convexity of f implies that f(v) f(w)+hrf(w); v wi. Therefore, when a function is both convex and smooth, we have both upper and lower bounds on the di erence between the function and its rst order approximation.

Setting v = w 1 rf(w) in the right-hand side of Equation ([12.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page162)) and rear-ranging terms, we obtain

21 krf(w)k2 f(w) f(v):

If we further assume that f(v) 0 for all v we conclude that smoothness implies the following:

|  |  |
| --- | --- |
| krf(w)k2 2 f(w) : | (12.6) |

A function that satis es 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 f0(x) = 2x. Note that for this particular function Equation ([12.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page162)) and Equation ([12.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page162)) hold with equality.

The function f(x) = log(1 + exp(x)) is (1=4)-smooth. Indeed, since f0(x) =

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  | we have that | | | |  |  |  |  |  |
| 1+exp( x) | | |  |  |  |  |  |
| j | f00 | (x) | | = |  | exp( x) | = |  | 1 |  | 1=4: |
| (1 + exp( x))2 | | (1 + exp( x))(1 + exp(x)) | |
|  |  | j |  |  |  |

Hence, f0 is (1=4)-Lipschitz. Since this function is nonnegative, Equa-tion ([12.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page162)) 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(hw; xi+b), where g : R ! R is a -smooth function, x 2 Rd, and b 2 R. Then, f is ( kxk2)-smooth.

|  |  |
| --- | --- |
| 12.2 Convex Learning Problems | 163 |
|  |  |

Proof By the chain rule we have that rf(w) = g0(hw; xi + b)x, where g0 is the derivative of g. Using the smoothness of g and the Cauchy-Schwartz inequality we therefore obtain

f(v) = g(hv; xi + b)

g(hw; xi + b) + g0(hw; xi + b)hv w; xi + 2 (hv w; xi)2 g(hw; xi + b) + g0(hw; xi + b)hv w; xi + 2 (kv wk kxk)2

1. f(w) + hrf(w); v wi + kxk2 kv wk2: 2

Example 12.6

For any x 2 Rd and y 2 R, let f(w) = (hw; xi y)2. Then, f is (2kxk2)-smooth.

For any x 2 Rd and y 2 f 1g, let f(w) = log(1 + exp( yhw; xi)). Then, f is (kxk2=4)-smooth.

12.2 Convex Learning Problems

Recall that in our general de nition of learning (De nition [3.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page49) in Chapter [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page43)), we have a hypothesis class H, a set of examples Z, and a loss function ` : H Z ! 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 Y, and H being a set of functions from

1. 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 Rd. That is, every hypothesis is some real-valued vector. We shall, therefore, denote a hypothesis in H by w. Now we can nally de ne convex learning problems:

definition 12.10 (Convex Learning Problem) A learning problem, (H; Z; `), is called convex if the hypothesis class H is a convex set and for all z 2 Z, the loss function, `( ; z), is a convex function (where, for any z, `( ; z) denotes the function f : H ! R de ned by f(w) = `(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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117)). 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. In Chapter [9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117) we de ned the hypothesis class as the set of homogenous linear functions, H = fx 7! wh; xi : w 2 Rdg, and used the squared loss function, `(h; (x; y)) = (h(x) y)2. However, we can equivalently model the learning problem as a convex learning problem as follows.

1. Convex Learning Problems

Each linear function is parameterized by a vector w 2 Rd. Hence, we can de ne 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)) = (hw; xi y)2. Clearly, the set H is a convex set. The loss function is also convex with respect to its rst argument (see Example [12.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page159)).

lemma 12.11 If ` is a convex loss function and the class H is convex, then the ERMH problem, of minimizing the empirical loss over H, is a convex optimiza-tion problem (that is, a problem of minimizing a convex function over a convex set).

Proof Recall that the ERMH problem is de ned by

ERMH(S) = argmin LS(w):

w2H

Since, for a sample S = z1; : : : ; zm, for every w, LS(w) = 1 Pm `(w; zi),

m i=1

Claim [12.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page160) implies that LS(w) is a convex function. Therefore, the ERM rule 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 e ciently using generic optimization algorithms. In particular, in Chapter [14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page184) 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 e ciently. But is convexity a su cient condition for the learnability of a problem?

To make the quesion more speci c: In VC theory, we saw that halfspaces in d-dimension are learnable (perhaps ine ciently). We also argued in Chapter [9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117) 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page164) 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 classi cation 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 nite number of bits su ces. 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

|  |  |
| --- | --- |
| 12.2 Convex Learning Problems | 165 |
|  |  |

homogenous case). Let A be any deterministic algorithm.[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page165) 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 ; if A receives a training set of size m m( ; ), it should output, with probability of at least 1 , a hypothesis w^ = A(S), such that LD(w^) minw LD(w) .

Choose = 1=100; = 1=2, let m m( ; ), and set = log(100=99) . We will

2m

de ne two distributions, and will show that A is likely to fail on at least one of them. The rst distribution, D1, is supported on two examples, z1 = (1; 0) and z2 = ( ; 1), where the probability mass of the rst example is while the probability mass of the second example is 1 . 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 )m e 2 m = 0:99:

Since we assume that A is a deterministic algorithm, upon receiving a training set of m examples, each of which is ( ; 1), the algorithm will output some w^. Now, if w^ < 1=(2 ), we will set the distribution to be D1. Hence,

LD1 (w^) (w^)2 1=(4 ):

However,

min LD1 (w) LD1 (0) = (1 ):

w

It follows that

1

LD1 (w^) min LD1 (w) 4 (1 ) > :

w

Therefore, such algorithm A fails on D1. On the other hand, if w^ 1=(2 ) then we'll set the distribution to be D2. Then we have that LD2 (w^) 1=4 while minw LD2 (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 hypoth-esis class. In addition to the convexity requirement, we require that H will be bounded; namely, we assume that for some prede ned scalar B, every hypothesis

1. 2 H satis es kwk B.

Boundedness and convexity alone are still not su cient for ensuring that the problem is learnable, as the following example demonstrates.

Example 12.9 As in Example [12.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page164), consider a regression problem with the squared loss. However, this time let H = fw : jwj 1g R be a bounded

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 will also fail to learn the problem.

1. Convex Learning Problems

hypothesis class. It is easy to verify that H is convex. The argument will be the same as in Example [12.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page164), except that now the two distributions, D1; D2 will be supported on z1 = (1= ; 0) and z2 = (1; 1). If the algorithm A returns w^ < 1=2 upon receiving m examples of the second type, then we will set the distribution to be D1 and have that

LD1 (w^) min LD1 (w) (w=^ )2 LD1 (0) 1=(4 ) (1 ) > :

w

Similarly, if w^ 1=2 we will set the distribution to be D2 and have that

LD2 (w^) min LD2 (w) ( 1=2 + 1)2 0 > :

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 de nition of two families of learning problems, convex-Lipschitz-bounded and convex-smooth-bounded, which are de ned 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 ; B if the following holds:

The hypothesis class H is a convex set and for all w 2 H we have kwk B. For all z 2 Z, the loss function, `( ; z), is a convex and -Lipschitz function.

Example 12.10 Let X = fx 2 Rd : kxk g and Y = R. Let H = fw 2 Rd : kwk Bg and let the loss function be `(w; (x; y)) = jhw; xi yj. This corre-sponds to a regression problem with the absolute-value loss, where we assume that the instances are in a ball of radius and we restrict the hypotheses to be homogenous linear functions de ned by a vector w whose norm is bounded by B. Then, the resulting problem is Convex-Lipschitz-Bounded with parameters ; B.

definition 12.13 (Convex-Smooth-Bounded Learning Problem) A learning problem, (H; Z; `), is called Convex-Smooth-Bounded, with parameters ; B if the following holds:

The hypothesis class H is a convex set and for all w 2 H we have kwk B. For all z 2 Z, the loss function, `( ; z), is a convex, nonnegative, and -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.

|  |  |
| --- | --- |
| 12.3 Surrogate Loss Functions | 167 |
|  |  |

Example 12.11 Let X = fx 2 Rd : kxk =2g and Y = R. Let H = fw 2

Rd : kwk B g and let the loss function be `(w; (x; y)) = (hw; xi y)2. This corresponds to a regression problem with the squared loss, where we assume that the instances are in a ball of radius =2 and we restrict the hypotheses to be homogenous linear functions de ned by a vector w whose norm is bounded by B. Then, the resulting problem is Convex-Smooth-Bounded with parameters ; 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 su cient 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 e ciently. 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,

`0 1(w; (x; y)) = 1[y6=sign(hw;xi)] = 1[yhw;xi 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](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page169)). Furthermore, as discussed in Chapter [8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page100), 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:

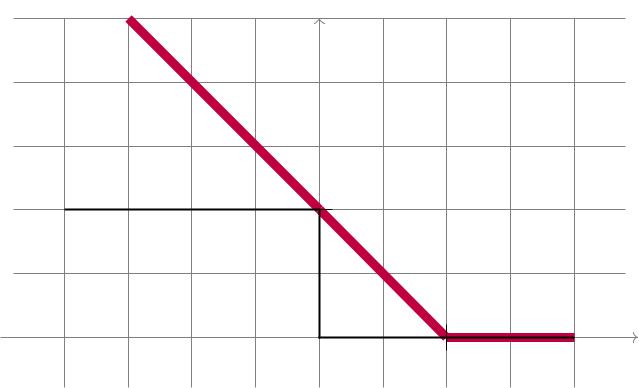
1. It should be convex.
2. It should upper bound the original loss.

For example, in the context of learning halfspaces, we can de ne the so-called hinge loss as a convex surrogate for the 0 1 loss, as follows:

|  |  |
| --- | --- |
| def | yhw; xig: |
| `hinge(w; (x; y)) = maxf0; 1 |

Clearly, for all w and all (x; y), `0 1(w; (x; y)) `hinge(w; (x; y)). In addition, the convexity of the hinge loss follows directly from Claim [12.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page160). Hence, the hinge loss satis es the requirements of a convex surrogate loss function for the zero-one loss. An illustration of the functions `0 1 and `hinge is given in the following.

1. Convex Learning Problems



`hinge

`0 1

1

1 yhw; xi

Once we have de ned 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| hinge | (A(S)) |  | min | hinge | (w) + ; |
| LD |  | w2H | LD |

where LhingeD(w) = E(x;y) D[`hinge(w; (x; y))]. Using the surrogate property, we can lower bound the left-hand side by L0D 1(A(S)), which yields

L0D 1(A(S)) min LhingeD(w) + :

w2H

We can further rewrite the upper bound as follows:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | LD | (A(S)) | w2H | LD | (w) + | w2H | LD | (w) | w2H | LD | (w) + : |
|  | 0 1 |  | min | 0 1 |  | min | hinge | | min | 0 1 |  |
|  | That is, the 0 1 error of the learned predictor is upper bounded by three terms: | | | | | | | | | | |
|  | Approximation error: This is the term minw2H LD0 1(w), which measures how | | | | | | | | | | |
|  | well the hypothesis class performs on the distribution. We already elabo- | | | | | | | | | | |
|  | rated on this error term in Chapter [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page60). | | | | | | |  |  |  |  |
|  | Estimation error: This is the error that results from the fact that we only | | | | | | | | | | |
|  | receive a training set and do not observe the distribution D. We already | | | | | | | | | | |
|  | elaborated on this error term in Chapter [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page60). | | | | | | |  |  |  |  |
|  | Optimization error: This is the term | | | | | minw2H LDhinge(w) minw2H LD0 1(w) | | | | | |
|  | that measures the di erence between the approximation error with respect | | | | | | | | | | |
|  |  |  |  |  |  | |  |  |  |  |  |
|  | to the surrogate loss and the approximation error with respect to the orig- | | | | | | | | | | |
|  | inal 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 speci c distribution of the data and on the speci c 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

|  |  |
| --- | --- |
| 12.5 Bibliographic Remarks | 169 |
|  |

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 & Lemarechal 1996). Regarding learning problems, the family of convex-Lipschitz-bounded problems was rst studied by Zinkevich (2003) in the context of online learning and by Shalev-Shwartz, Shamir, Sridharan & Srebro (2009) in the con-text of PAC learning.

12.6 Exercises

1. Construct an example showing that the 0 1 loss function may su er from local minima; namely, construct a training sample S 2 (X f 1g)m (say, for X = R2), for which there exist a vector w and some > 0 such that
   1. For any w0 such that kw w0k we have LS(w) LS(w0) (where the loss here is the 0 1 loss). This means that w is a local minimum of LS.
   2. There exists some w such that LS(w ) < LS(w). This means that w is not a global minimum of LS.
2. Consider the learning problem of logistic regression: Let H = X = fx 2

Rd : kxk Bg, for some scalar B > 0, let Y = f 1g, and let the loss function ` be de ned as `(w; (x; y)) = log(1 + exp( yhw; xi)). Show that the resulting learning problem is both convex-Lipschitz-bounded and convex-smooth-bounded. Specify the parameters of Lipschitzness and smoothness.

1. Consider the problem of learning halfspaces with the hinge loss. We limit our domain to the Euclidean ball with radius R. That is, X = fx : kxk2 Rg. The label set is Y = f 1g and the loss function ` is de ned by `(w; (x; y)) = maxf0; 1 yhw; xig. We already know that the loss function is convex. Show that it is R-Lipschitz.
2. (\*) Convex-Lipschitz-Boundedness Is Not Su cient for Computa-tional E ciency: In the next chapter we show that from the statistical perspective, all convex-Lipschitz-bounded problems are learnable (in the ag-nostic PAC model). However, our main motivation to learn such problems resulted from the computational perspective { convex optimization is often e ciently solvable. Yet the goal of this exercise is to show that convexity alone is not su cient for e ciency. 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

1. Convex Learning Problems

the set of all Turing machines. De ne the loss function as follows. For every Turing machine T 2 Z, let `(0; T ) = 1 if T halts on the input 0 and `(0; T ) = 0 if T doesn't halt on the input 0. Similarly, let `(1; T ) = 0 if T halts on the input 0 and `(1; T ) = 1 if T doesn't halt on the input 0. Finally, for h 2 (0; 1), let `(h; T ) = h`(0; T ) + (1 h)`(1; T ).

1. Show that the resulting learning problem is convex-Lipschitz-bounded.
2. Show that no computable algorithm can learn the problem.

1. Regularization and Stability

In the previous chapter we introduced the families of convex-Lipschitz-bounded and convex-smooth-bounded learning problems. In this section we show that all learning problems in these two families are learnable. For some learning problems of this type it is possible to show that uniform convergence holds; hence they are learnable using the ERM rule. However, this is not true for all learning problems of this type. Yet, we will introduce another learning rule and will show that it learns all convex-Lipschitz-bounded and convex-smooth-bounded learning problems.

The new learning paradigm we introduce in this chapter is called Regularized Loss Minimization, or RLM for short. In RLM we minimize the sum of the em-pirical risk and a regularization function. Intuitively, the regularization function measures the complexity of hypotheses. Indeed, one interpretation of the reg-ularization function is the structural risk minimization paradigm we discussed in Chapter [7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page83). Another view of regularization is as a stabilizer of the learning algorithm. An algorithm is considered stable if a slight change of its input does not change its output much. We will formally de ne the notion of stability (what we mean by \slight change of input" and by \does not change much the out-put") and prove its close relation to learnability. Finally, we will show that using the squared `2 norm as a regularization function stabilizes all convex-Lipschitz or convex-smooth learning problems. Hence, RLM can be used as a general learning rule for these families of learning problems.

13.1 Regularized Loss Minimization

Regularized Loss Minimization (RLM) is a learning rule in which we jointly min-imize the empirical risk and a regularization function. Formally, a regularization function is a mapping R : Rd ! R, and the regularized loss minimization rule outputs a hypothesis in

|  |  |
| --- | --- |
| argmin (LS(w) + R(w)) : | (13.1) |
| w |  |

Regularized loss minimization shares similarities with minimum description length algorithms and structural risk minimization (see Chapter [7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page83)). Intuitively, the \complexity" of hypotheses is measured by the value of the regularization func-

1. Regularization and Stability

tion, and the algorithm balances between low empirical risk and \simpler," or \less complex," hypotheses.

There are many possible regularization functions one can use, re ecting some prior belief about the problem (similarly to the description language in Minimum Description Length). Throughout this section we will focus on one of the most

simple regularization functions: R(w) = kwk2, where > 0 is a scalar and the

q

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | d | 2 |  |  |  | the learning rule: | | |  |
| norm is the `2 norm, kwk = | Pi=1 wi . This yields | | | | |  |
| k | 2 |  |  |
| A(S) = | w |  | S | k | w |  | (13.2) |
|  | argmin | L |  | (w) + |  |  | : |

This type of regularization function is often called Tikhonov regularization.

As mentioned before, one interpretation of Equation ([13.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page172)) is using structural risk minimization, where the norm of w is a measure of its \complexity." Recall that in the previous chapter we introduced the notion of bounded hypothesis classes. Therefore, we can de ne a sequence of hypothesis classes, H1 H2 H3 : : :, where Hi = fw : kwk2 ig. If the sample complexity of each Hi depends on i then the RLM rule is similar to the SRM rule for this sequence of nested classes.

A di erent interpretation of regularization is as a stabilizer. In the next section we de ne the notion of stability and prove that stable learning rules do not over t. But rst, let us demonstrate the RLM rule for linear regression with the squared loss.

13.1.1 Ridge Regression

Applying the RLM rule with Tikhonov regularization to linear regression with the squared loss, we obtain the following learning rule:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w Rd | k | k2 |  | m | =1 | 2 | | h |  | ii | i | ! | : | (13.3) |
| argmin |  | w 2 | + | 1 | m | 1 | | ( | w; x |  | y | )2 |  |  |
|  | Xi |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |

Performing linear regression using Equation ([13.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page172)) is called ridge regression. To solve Equation ([13.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page172)) we compare the gradient of the objective to zero and

obtain the set of linear equations

(2 mI + A)w = b;

where I is the identity matrix and A; b are as de ned in Equation ([9.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page124)), namely,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A = | m | xi xi>! | and b = | m | yixi : | (13.4) |
|  | X |  |  | Xi |  |  |
|  | i=1 |  |  | =1 |  |  |

Since A is a positive semide nite matrix, the matrix 2 mI + A has all its eigen-values bounded below by 2 m. Hence, this matrix is invertible and the solution to ridge regression becomes

|  |  |
| --- | --- |
| w = (2 mI + A) 1 b: | (13.5) |

|  |  |
| --- | --- |
| 13.2 Stable Rules Do Not Over t | 173 |
|  |  |

In the next section we formally show how regularization stabilizes the algo-rithm and prevents over tting. In particular, the analysis presented in the next sections (particularly, Corollary [13.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page180)) will yield:

theorem 13.1 Let D be a distribution over X [ 1; 1], where X = fx 2 Rd : kxk 1g. Let H = fw 2 Rd : kwk Bg. For any 2 (0; 1), let m 150 B2= 2. Then, applying the ridge regression algorithm with parameter = =(3B2) satis es

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S | E | m [LD(A(S))] |  | min | | LD(w) + : |
| w | 2H |
|  | D |  |  |  |  |

Remark 13.1 The preceding theorem tells us how many examples are needed to guarantee that the expected value of the risk of the learned predictor will be bounded by the approximation error of the class plus . In the usual de nition of agnostic PAC learning we require that the risk of the learned predictor will be bounded with probability of at least 1 . In Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page181) we show how an algorithm with a bounded expected risk can be used to construct an agnostic PAC learner.

13.2 Stable Rules Do Not Over t

Intuitively, a learning algorithm is stable if a small change of the input to the algorithm does not change the output of the algorithm much. Of course, there are many ways to de ne what we mean by \a small change of the input" and what we mean by \does not change the output much". In this section we de ne a speci c notion of stability and prove that under this de nition, stable rules do not over t.

Let A be a learning algorithm, let S = (z1; : : : ; zm) be a training set of m examples, and let A(S) denote the output of A. The algorithm A su ers from over tting if the di erence between the true risk of its output, LD(A(S)), and the empirical risk of its output, LS(A(S)), is large. As mentioned in Remark [13.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page173), throughout this chapter we focus on the expectation (with respect to the choice of S) of this quantity, namely, ES[LD(A(S)) LS(A(S))].

We next de ne the notion of stability. Given the training set S and an ad-ditional example z0, let S(i) be the training set obtained by replacing the i'th example of S with z0; namely, S(i) = (z1; : : : ; zi 1; z0; zi+1; : : : ; zm). In our de - nition of stability, \a small change of the input" means that we feed A with S(i) instead of with S. That is, we only replace one training example. We measure the e ect of this small change of the input on the output of A, by comparing the loss of the hypothesis A(S) on zi to the loss of the hypothesis A(S(i)) on zi. Intuitively, a good learning algorithm will have `(A(S(i)); zi) `(A(S); zi) 0, since in the rst term the learning algorithm does not observe the example zi while in the second term zi is indeed observed. If the preceding di erence is very large we suspect that the learning algorithm might over t. This is because the

1. Regularization and Stability

learning algorithm drastically changes its prediction on zi if it observes it in the training set. This is formalized in the following theorem.

theorem 13.2 Let D be a distribution. Let S = (z1; : : : ; zm) be an i.i.d. se-quence of examples and let z0 be another i.i.d. example. Let U(m) be the uniform distribution over [m]. Then, for any learning algorithm,

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | [L | D | (A(S)) |  | LS(A(S))] = | E |  | [`(A(S(i); zi)) | |  | `(A(S); zi)]: |
| S Dm |  |  |  | (S;z0) Dm+1;i U(m) | | |  | (13.6) |
|  |  |  |  |  |  |  |  | D |  |  |
| Proof | Since S and z0 are both drawn i.i.d. from | | | | | | | , we have that for every i, | | |
|  |
|  |  |  | E[L | | (A(S))] = E [`(A(S); z0)] = | | E [`(A(S(i)); zi)]: | | | |  |
|  |  |  | S | D | S;z0 |  | S;z0 | |  |  |  |

On the other hand, we can write

E[LS(A(S))] = E [`(A(S); zi)]:

1. S;i

Combining the two equations we conclude our proof.

When the right-hand side of Equation ([13.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page174)) is small, we say that A is a stable algorithm { changing a single example in the training set does not lead to a signi cant change. Formally,

definition 13.3 (On-Average-Replace-One-Stable) Let : N ! R be a mono-tonically decreasing function. We say that a learning algorithm A is on-average-replace-one-stable with rate (m) if for every distribution D

1. [`(A(S(i); zi)) `(A(S); zi)] (m):

(S;z0) Dm+1;i U(m)

Theorem [13.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page174) tells us that a learning algorithm does not over t if and only if it is on-average-replace-one-stable. Of course, a learning algorithm that does not over t is not necessarily a good learning algorithm { take, for example, an algorithm A that always outputs the same hypothesis. A useful algorithm should nd a hypothesis that on one hand ts the training set (i.e., has a low empirical risk) and on the other hand does not over t. Or, in light of Theorem [13.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page174), the algorithm should both t the training set and at the same time be stable. As we shall see, the parameter of the RLM rule balances between tting the training set and being stable.

13.3 Tikhonov Regularization as a Stabilizer

In the previous section we saw that stable rules do not over t. In this section we show that applying the RLM rule with Tikhonov regularization, kw k2, leads to a stable algorithm. We will assume that the loss function is convex and that it is either Lipschitz or smooth.

The main property of the Tikhonov regularization that we rely on is that it makes the objective of RLM strongly convex, as de ned in the following.

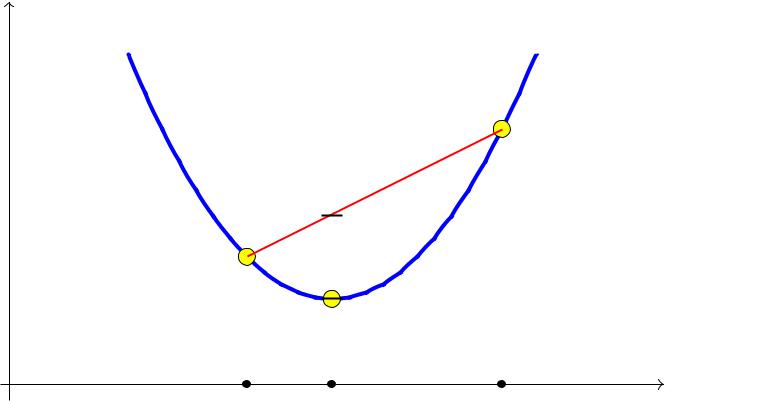
|  |  |
| --- | --- |
| 13.3 Tikhonov Regularization as a Stabilizer | 175 |
|  |  |

definition 13.4 (Strongly Convex Functions) A function f is -strongly con-vex if for all w; u and 2 (0; 1) we have

f( w + (1 )u) f(w) + (1 )f(u) (1 )kw uk2:

2

convexity is given in the following gure.



f(u)

f(w)

 2 (1 )ku

wk2

1. u

w + (1 )u

The following lemma implies that the objective of RLM is (2 )-strongly con-vex. In addition, it underscores an important property of strong convexity.

lemma 13.5

1. The function f(w) = kwk2 is 2 -strongly convex.
2. If f is -strongly convex and g is convex, then f + g is -strongly convex.
3. If f is -strongly convex and u is a minimizer of f, then, for any w,

f(w) f(u) 2 kw uk2:

Proof The rst two points follow directly from the de nition. To prove the last point, we divide the de nition of strong convexity by and rearrange terms to get that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| f(u + (w u)) f(u) |  | f(w) |  | f(u) |  |  | (1 |  | ) | w |  | u | 2: |
|  | 2 |
|  |  |  | k |  | k |  |

Taking the limit ! 0 we obtain that the right-hand side converges to f(w) f(u) 2 kw uk2. On the other hand, the left-hand side becomes the derivative of the function g( ) = f(u + (w u)) at = 0. Since u is a minimizer of f, it follows that = 0 is a minimizer of g, and therefore the left-hand side of the preceding goes to zero in the limit ! 0, which concludes our proof. 

We now turn to prove that RLM is stable. Let S = (z1; : : : ; zm) be a training set, let z0 be an additional example, and let S(i) = (z1; : : : ; zi 1; z0; zi+1; : : : ; zm). Let A be the RLM rule, namely,

A(S) = argmin LS(w) + kwk2 :

w

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 176 | Regularization and Stability | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | | | | | | | | | | | | | | |  | | |  | | | | |  |  |  |  |  |  |  |  |  |  |  |
|  | Denote fS(w) = LS(w) + kwk2, and based on Lemma [13.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page175) we know that fS is | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  | (2 )-strongly convex. Relying on part 3 of the lemma, it follows that for any v, | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | fS(v) fS(A(S)) kv A(S)k2: | | | | | | | | | | | | | | | | | | (13.7) | | |  |
|  | On the other hand, for any v and u, and for all i, we have | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
|  |  |  |  | fS(v) fS(u) = LS(v) + kvk2 (LS(u) + kuk2) | | | | | | | | | | | | | | | | | | | | | | | | | | | | (13.8) | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | = LS(i) (v) + kvk2 (LS(i) (u) + kuk2) | | | | | | | | | | | | | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | + | | `(v; zi) `(u; zi) | | | | | + | | `(u; z0) `(v; z0) | | | | | : |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | m |  |  |  |  |  |  | m | |  |  |  |  |
|  | In particular, choosing v = A(S(i)), u = A(S), and using the fact that v mini- | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  | mizes LS(i) (w) + kwk2, we obtain that | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  | f | S | (A(S(i))) | |  | | f | S | (A(S)) | | | | | |  |  | `(A(S(i)); zi) `(A(S); zi) | | | | | | | | | | | | + | `(A(S); z0) `(A(S(i)); z0) | | | | | : |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | m |  |  |  |  |  |  |  | m | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | (13.9) | | |  |
|  | Combining this with Equation ([13.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page176)) we obtain that | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
|  |  | k | A(S(i)) |  | | A(S) | | | | k | 2 | |  | | `(A(S(i)); zi) `(A(S); zi) | | | | | | | | | | | + | | `(A(S); z0) `(A(S(i)); z0) | | | | | : | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | m | | |  |  |  |  |  |  |  |  | m | (13.10) | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | The two subsections that follow continue the stability analysis for either Lip- | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  | schitz or smooth loss functions. For both families of loss functions we show that | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  | RLM is stable and therefore it does not over t. | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| 13.3.1 | Lipschitz Loss | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | If the loss function, `( ; zi), is -Lipschitz, then by the de nition of Lipschitzness, | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  |  |  | `(A(S(i)); zi) `(A(S); zi) kA(S(i)) A(S)k: | | | | | | | | | | | | | | | | | | | | | | | | (13.11) | | |  |
|  | Similarly, | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | `(A(S); z0) `(A(S(i)); z0) kA(S(i)) A(S)k: | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
|  | Plugging these inequalities into Equation ([13.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page176)) we obtain | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | | k | | A(S(i)) | | | |  | A(S) | | k | 2 |  | 2 kA(S(i)) A(S)k | | | | | | | ; |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | m | | | | | |  |  |  |  |
|  | which yields | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | kA(S(i)) A(S)k | | | | | | | | 2 | | | | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | : | | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | m | | |  |  |  |  |

Plugging the preceding back into Equation ([13.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page176)) we conclude that

`(A(S(i)); zi) `(A(S); zi) 2 2 :

m

Since this holds for any S; z0; i we immediately obtain:

|  |  |
| --- | --- |
| 13.3 Tikhonov Regularization as a Stabilizer | 177 |
|  |  |

corollary 13.6 Assume that the loss function is convex and -Lipschitz. Then, the RLM rule with the regularizer kwk2 is on-average-replace-one-stable with rate 2 2 . It follows (using Theorem [13.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page174)) that

m

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E | m[LD(A(S)) LS(A(S))] | 2 2 | |
| S |  | : |
| m |
|  | D |  |  |  |

13.3.2 Smooth and Nonnegative Loss

If the loss is -smooth and nonnegative then it is also self-bounded (see Sec-tion [12.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page156)):

|  |  |
| --- | --- |
| krf(w)k2 2 f(w): | (13.12) |

We further assume that 2m , or, in other words, that m=2. By the smoothness assumption we have that

`(A(S(i)); zi) `(A(S); zi) hr`(A(S); zi); A(S(i)) A(S)i+ kA(S(i)) A(S)k2 :

2

Using the Cauchy-Schwartz inequality and Equation ([12.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page162)) we further obtain that

`(A(S(i)); zi) `(A(S); zi)

kr`(A(S); zi)k kA(S(i)) A(S)k + 2 kA(S(i)) A(S)k2

p2 `(A(S); zi) kA(S(i)) A(S)k + kA(S(i)) A(S)k2 :

2

By a symmetric argument it holds that,

`(A(S); z0) `(A(S(i)); z0)

q2 `(A(S(i)); z0) kA(S(i)) A(S)k + 2 kA(S(i)) A(S)k2 :

Plugging these inequalities into Equation ([13.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page176)) and rearranging terms we ob-tain that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | p |  | ) | | | `(A(S); zi) + `(A(S(i)); z0) | | | | | | | : |
| kA(S(i)) A(S)k ( m2 | | | |
|  |  |  |  |  |  | p |  |  |  | q |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |
| Combining the preceding with the assumptionm=2 yields | | | | | | | | | | | | | |  |
|  |  | p |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 8 |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| kA(S(i)) A(S)k | | p`(A(S); zi) + q`(A(S(i)); z0) : | | | | | | | | | |  |
|  | |  |
| m | |  |

1. Regularization and Stability

Combining the preceding with Equation ([13.14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page177)) and again using the assumption

m=2 yield

`(A(S(i)); zi) `(A(S); zi)

p2 `(A(S); zi) kA(S(i)) A(S)k + 2 kA(S(i)) A(S)k2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 4 | |  |  |  | 8 2 |  |  |  |  |  |  |  |  | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | | + |  |  | | `(A(S); zi) + `(A(S(i)); z0) | | | | | |  |
| m | |  | ( m)2 | |  |
|  |  |  |  |  |  |  |  | p | |  |  | q | | 2 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| m | | | | | p |  | | | + q`(A(S(i)); z0) | | | | |  |  |  |
| `(A(S); zi) | | |  |  |  |
|  | 8 | |  | |  |  |  |  |  |  |  |  |  |  |  |  |

24m `(A(S); zi) + `(A(S(i)); z0) ;

where in the last step we used the inequality (a+b)2 3(a2+b2). Taking expecta-tion with respect to S; z0; i and noting that E[`(A(S); zi)] = E[`(A(S(i)); z0)] = E[LS(A(S))], we conclude that:

corollary 13.7 Assume that the loss function is -smooth and nonnegative. Then, the RLM rule with the regularizer kwk2, where 2m , satis es

E h`(A(S(i)); zi) `(A(S); zi)i 48m E[LS(A(S))]:

Note that if for all z we have `(0; z) C, for some scalar C > 0, then for every S,

LS(A(S)) LS(A(S)) + kA(S)k2 LS(0) + k0k2 = LS(0) C:

Hence, Corollary [13.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page178) also implies that

E h`(A(S(i)); zi) `(A(S); zi)i 48 C :

m

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 13.4 | Controlling the Fitting-Stability Tradeo | | | | |  |  |  |
|  | We can rewrite the expected risk of a learning algorithm as | | | | | | |  |
|  | E[L | D | (A(S))] = E[LS(A(S))] + E[L (A(S)) | | |  | LS(A(S))]: | (13.15) |
|  | S | S | S | D |  |  |

The rst term re ects how well A(S) ts the training set while the second term re ects the di erence between the true and empirical risks of A(S). As we have shown in Theorem [13.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page174), the second term is equivalent to the stability of A. Since our goal is to minimize the risk of the algorithm, we need that the sum of both terms will be small.

In the previous section we have bounded the stability term. We have shown that the stability term decreases as the regularization parameter, , increases. On the other hand, the empirical risk increases with . We therefore face a

|  |  |
| --- | --- |
| 13.4 Controlling the Fitting-Stability Tradeo | 179 |
|  |  |

tradeo between tting and over tting. This tradeo is quite similar to the bias-complexity tradeo we discussed previously in the book.

We now derive bounds on the empirical risk term for the RLM rule. Recall that the RLM rule is de ned as A(S) = argminw LS(w) + kwk2 . Fix some arbitrary vector w . We have

LS(A(S)) LS(A(S)) + kA(S)k2 LS(w ) + kw k2:

Taking expectation of both sides with respect to S and noting that ES[LS(w )] = LD(w ), we obtain that

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| E[LS(A(S))] |  | L | (w ) + | w | 2: | (13.16) |
| S | D | k | k |  |  |

Plugging this into Equation ([13.15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page178)) we obtain

E[LD(A(S))] LD(w ) + kw k2 + E[LD(A(S)) LS(A(S))]:

S S

Combining the preceding with Corollary [13.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page176) we conclude:

corollary 13.8 Assume that the loss function is convex and -Lipschitz. Then, the RLM rule with the regularization function kwk2 satis es

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | w ; E[L (A(S))] |  |  | (w ) + | w | 2 + | 2 2 | |
| L |  | : |
|  |
| S D | D | k | k |  | m | |

This bound is often called an oracle inequality { if we think of w as a hy-pothesis with low risk, the bound tells us how many examples are needed so that A(S) will be almost as good as w , had we known the norm of w . In practice, however, we usually do not know the norm of w . We therefore usually tune on the basis of a validation set, as described in Chapter [11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page144).

We can also easily derive a PAC-like guarantee[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page179) from Corollary [13.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page179) for convex-

Lipschitz-bounded learning problems:

corollary 13.9 Let (H; Z; `) be a convex-Lipschitz-bounded learning problem

q

with parameters ; B. For any training set size m, let = 2 2 . Then, the

B2 m

RLM rule with the regularization function kwk2 satis es

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E[L (A(S))] | |  | min L |  | (w) + B | | r | 8 |  | : |
| D |  |  |
| S | D | w2H |  |  | m | | |
|  |  |  | m | 8 2B2 | | |  |  | every distribution D, | |
| In particular, for every | > 0, | if |  |  | then | for | |
|  | 2 |

ES[LD(A(S))] minw2H LD(w) + .

The preceding corollary holds for Lipschitz loss functions. If instead the loss function is smooth and nonnegative, then we can combine Equation ([13.16](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page179)) with Corollary [13.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page178) to get:

1. Again, the bound below is on the expected risk, but using Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page181) it can be used to derive an agnostic PAC learning guarantee.

1. Regularization and Stability

corollary 13.10 Assume that the loss function is convex, -smooth, and nonnegative. Then, the RLM rule with the regularization function kwk2, for

2m , satis es the following for all w :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S D |  |  | m | | S |  |  |  |  |  | m | |  | D | k | k |  |  |
| E[L (A(S))] |  | 1 + | 48 |  | E[LS | (A(S))] | | |  | 1 + | 48 |  |  | L (w ) + | | w | 2 | : |
|  |  |  |  |  |  |
| For example, if we choose = | | | | | | 48 | | we obtain from the preceding that the | | | | | | | | | | |
|  | m |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

expected true risk of A(S) is at most twice the expected empirical risk of A(S).

Furthermore, for this value of , the expected empirical risk of A(S) is at most LD(w ) + 48m kw k2.

We can also derive a learnability guarantee for convex-smooth-bounded learn-ing problems based on Corollary [13.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page179).

corollary 13.11 Let (H; Z; `) be a convex-smooth-bounded learning problem with parameters ; B. Assume in addition that `(0; z) 1 for all z 2 Z. For any

2 (0; 1) let m 150 B2 and set = =(3B2). Then, for every distribution D,

2

E[LD(A(S))] min LD(w) + :

S w2H

13.5 Summary

We introduced stability and showed that if an algorithm is stable then it does not over t. Furthermore, for convex-Lipschitz-bounded or convex-smooth-bounded problems, the RLM rule with Tikhonov regularization leads to a stable learning algorithm. We discussed how the regularization parameter, , controls the trade-o between tting and over tting. Finally, we have shown that all learning prob-lems that are from the families of convex-Lipschitz-bounded and convex-smooth-bounded problems are learnable using the RLM rule. The RLM paradigm is the basis for many popular learning algorithms, including ridge regression (which we discussed in this chapter) and support vector machines (which will be discussed in Chapter [15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page202)).

In the next chapter we will present Stochastic Gradient Descent, which gives us a very practical alternative way to learn convex-Lipschitz-bounded and convex-smooth-bounded problems and can also be used for e ciently implementing the RLM rule.

13.6 Bibliographic Remarks

Stability is widely used in many mathematical contexts. For example, the neces-sity of stability for so-called inverse problems to be well posed was rst recognized by Hadamard (1902). The idea of regularization and its relation to stability be-came widely known through the works of Tikhonov (1943) and Phillips (1962).

|  |  |
| --- | --- |
| 13.7 Exercises | 181 |
|  |  |

In the context of modern learning theory, the use of stability can be traced back at least to the work of Rogers & Wagner (1978), which noted that the sensitiv-ity of a learning algorithm with regard to small changes in the sample controls the variance of the leave-one-out estimate. The authors used this observation to obtain generalization bounds for the k-nearest neighbor algorithm (see Chap-ter [19](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page258)). These results were later extended to other \local" learning algorithms (see Devroye, Gy•or & Lugosi (1996) and references therein). In addition, practi-cal methods have been developed to introduce stability into learning algorithms, in particular the Bagging technique introduced by (Breiman 1996).

Over the last decade, stability was studied as a generic condition for learnabil-ity. See (Kearns & Ron 1999, Bousquet & Elissee 2002, Kutin & Niyogi 2002, Rakhlin, Mukherjee & Poggio 2005, Mukherjee, Niyogi, Poggio & Rifkin 2006). Our presentation follows the work of Shalev-Shwartz, Shamir, Srebro & Sridha-ran (2010), who showed that stability is su cient and necessary for learning. They have also shown that all convex-Lipschitz-bounded learning problems are learnable using RLM, even though for some convex-Lipschitz-bounded learning problems uniform convergence does not hold in a strong sense.

13.7 Exercises

1. From Bounded Expected Risk to Agnostic PAC Learning: Let A be

an algorithm that guarantees the following: If m mH( ) then for every distribution D it holds that

E [LD(A(S))] min LD(h) + :

S Dm h2H

Show that for every 2 (0; 1), if m mH( ) then with probability of at least 1 it holds that LD(A(S)) minh2H LD(h) + .

Hint: Observe that the random variable LD(A(S)) minh2H LD(h) is nonnegative and rely on Markov's inequality.

For every 2 (0; 1) let

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| mH( ; ) = mH( =2)dlog2(1= )e + log(4= ) + | | 2 d | 2 | e | | : |
|  |  | log( log | | (1= ) ) | |  |
|  |  |  |  |  |  |  |

Suggest a procedure that agnostic PAC learns the problem with sample complexity of mH( ; ), assuming that the loss function is bounded by 1.

Hint: Let k = dlog2(1= )e. Divide the data into k +1 chunks, where each of the rst k chunks is of size mH( =2) examples. Train the rst k chunks using A. On the basis of the previous question argue that the probability that for all of these chunks we have LD(A(S)) > minh2H LD(h) + is at most 2 k =2. Finally, use the last chunk as a validation set.

2. Learnability without Uniform Convergence: Let B be the unit ball of

1. Regularization and Stability

Rd, let H = B, let Z = B f0; 1gd, and let ` : Z H ! R be de ned as follows:

d

X

`(w; (x; )) = i(xi wi)2:

i=1

This problem corresponds to an unsupervised learning task, meaning that we do not try to predict the label of x. Instead, what we try to do is to nd the \center of mass" of the distribution over B. However, there is a twist, modeled by the vectors . Each example is a pair (x; ), where x is the instance x and

indicates which features of x are \active" and which are \turned o ." A hypothesis is a vector w representing the center of mass of the distribution, and the loss function is the squared Euclidean distance between x and w, but only with respect to the \active" elements of x.

Show that this problem is learnable using the RLM rule with a sample complexity that does not depend on d.

Consider a distribution D over Z as follows: x is xed to be some x0, and each element of is sampled to be either 1 or 0 with equal probability. Show that the rate of uniform convergence of this problem grows with

d.

Hint: Let m be a training set size. Show that if d 2m, then there is a high probability of sampling a set of examples such that there exists some j 2 [d] for which j = 1 for all the examples in the training set. Show that such a sample cannot be -representative. Conclude that the sample complexity of uniform convergence must grow with log(d).

Conclude that if we take d to in nity we obtain a problem that is learnable but for which the uniform convergence property does not hold. Compare to the fundamental theorem of statistical learning.

1. Stability and Asymptotic ERM Are Su cient for Learnability:

We say that a learning rule A is an AERM (Asymptotic Empirical Risk Minimizer) with rate (m) if for every distribution D it holds that

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S Dm | LS(A(S)) |  | h2H |  |  |
| E |  | min LS(h) |  | (m): |

We say that a learning rule A learns a class H with rate (m) if for every distribution D it holds that

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S Dm |  | D | (A(S)) | h2H D |  | (m): |
| E | L |  | min L (h) |  |
| Prove the following: |  |  |  |  |  |  |

theorem 13.12 If a learning algorithm A is on-average-replace-one-stable with rate 1(m) and is an AERM with rate 2(m), then it learns H with rate 1(m) + 2(m).

|  |  |
| --- | --- |
| 13.7 Exercises | 183 |
|  |  |

1. Strong Convexity with Respect to General Norms:

Throughout the section we used the `2 norm. In this exercise we generalize some of the results to general norms. Let k k be some arbitrary norm, and let f be a strongly convex function with respect to this norm (see De nition [13.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page174)).

1. Show that items 2{3 of Lemma [13.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page175) hold for every norm.
2. (\*) Give an example of a norm for which item 1 of Lemma [13.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page175) does not hold.
3. Let R(w) be a function that is (2 )-strongly convex with respect to some norm k k. Let A be an RLM rule with respect to R, namely,

A(S) = argmin (LS(w) + R(w)) :

w

Assume that for every z, the loss function `( ; z) is -Lipschitz with respect to the same norm, namely,

8z; 8w; v; `(w; z) `(v; z) kw vk :

Prove that A is on-average-replace-one-stable with rate 2 2 .

m

4. (\*) Let q 2 (1; 2) and consider the `q-norm

|  |  |  |  |
| --- | --- | --- | --- |
| kwkq = | d | jwijq!1=q | : |
|  | Xi |  |  |
|  | =1 |  |  |

It can be shown (see, for example, Shalev-Shwartz (2007)) that the function

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | | kwkq2 |  |  |  |  |  |
|  |  | R(w) = |  |  |  |  |  |  |
|  |  | 2(q 1) |  |  |  |  |  |
| is 1-strongly convex with respect to kwkq. Show that if q = | | | | | |  | log(d) |  | then |
| log(d) 1 | | |
| R(w) is | 1 | -strongly convex with respect to the `1 | | | norm over Rd. | | | | |
| 3 log(d) |

1. Stochastic Gradient Descent

Recall that the goal of learning is to minimize the risk function, LD(h) = Ez D[`(h; z)]. We cannot directly minimize the risk function since it depends on the unknown distribution D. So far in the book, we have discussed learning methods that depend on the empirical risk. That is, we rst sample a training set S and de ne the empirical risk function LS(h). Then, the learner picks a hypothesis based on the value of LS(h). For example, the ERM rule tells us to pick the hypothesis that minimizes LS(h) over the hypothesis class, H. Or, in the previous chapter, we discussed regularized risk minimization, in which we pick a hypothesis that jointly minimizes LS(h) and a regularization function over h.

In this chapter we describe and analyze a rather di erent learning approach, which is called Stochastic Gradient Descent (SGD). As in Chapter [12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page156) we will focus on the important family of convex learning problems, and following the notation in that chapter, we will refer to hypotheses as vectors w that come from a convex hypothesis class, H. In SGD, we try to minimize the risk function LD(w) directly using a gradient descent procedure. Gradient descent is an iterative optimization procedure in which at each step we improve the solution by taking a step along the negative of the gradient of the function to be minimized at the current point. Of course, in our case, we are minimizing the risk function, and since we do not know D we also do not know the gradient of LD(w). SGD circumvents this problem by allowing the optimization procedure to take a step along a random direction, as long as the expected value of the direction is the negative of the gradient. And, as we shall see, nding a random direction whose expected value corresponds to the gradient is rather simple even though we do not know the underlying distribution D.

The advantage of SGD, in the context of convex learning problems, over the regularized risk minimization learning rule is that SGD is an e cient algorithm that can be implemented in a few lines of code, yet still enjoys the same sample complexity as the regularized risk minimization rule. The simplicity of SGD also allows us to use it in situations when it is not possible to apply methods that are based on the empirical risk, but this is beyond the scope of this book.

We start this chapter with the basic gradient descent algorithm and analyze its convergence rate for convex-Lipschitz functions. Next, we introduce the notion of subgradient and show that gradient descent can be applied for nondi erentiable functions as well. The core of this chapter is Section [14.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page191), in which we describe

|  |  |
| --- | --- |
| 14.1 Gradient Descent | 185 |
|  |  |

the Stochastic Gradient Descent algorithm, along with several useful variants. We show that SGD enjoys an expected convergence rate similar to the rate of gradient descent. Finally, we turn to the applicability of SGD to learning problems.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 14.1 | Gradient Descent |  |  |  |  |  |  |  |
|  | Before we describe the stochastic gradient descent method, we would like to | | | | | | | |
|  | describe the standard gradient descent approach for minimizing a di erentiable | | | | | | | |
|  | convex function f(w). |  |  |  |  |  |  |  |
|  | The gradient of a di erentiable function f : Rd ! R at w, denoted rf(w), | | | | | | | |
|  | is the vector of partial derivatives of f, namely, rf(w) = | |  | @f(w) | | @f(w) | | |
|  |  |  | ; : : : ; |  |  | . |
|  |  | @w[1] | @w[d] |
|  | Gradient descent is an iterative algorithm. We start with | an initial value of w | | | | | | |
|  |  |  | | |  |  |  |
|  | (say, w(1) = 0). Then, at each iteration, we take a step in the direction of the | | | | | | | |
|  | negative of the gradient at the current point. That is, the update step is | | | | | | | |
|  | w(t+1) = w(t)rf(w(t)); |  |  |  |  | (14.1) | | |

where > 0 is a parameter to be discussed later. Intuitively, since the gradi-ent points in the direction of the greatest rate of increase of f around w(t), the algorithm makes a small step in the opposite direction, thus decreasing the value of the function. Eventually, after T iterations, the algorithm outputs the averaged vector, w = T1 PTt=1 w(t). The output could also be the last vector, w(T ), or the best performing vector, argmint2[T ] f(w(t)), but taking the average turns out to be rather useful, especially when we generalize gradient descent to nondi erentiable functions and to the stochastic case.

Another way to motivate gradient descent is by relying on Taylor approxima-tion. The gradient of f at w yields the rst order Taylor approximation of f around w by f(u) f(w) + hu w; rf(w)i. When f is convex, this approxi-mation lower bounds f, that is,

f(u) f(w) + hu w; rf(w)i:

Therefore, for w close to w(t) we have that f(w) f(w(t))+hw w(t); rf(w(t))i. Hence we can minimize the approximation of f(w). However, the approximation might become loose for w, which is far away from w(t). Therefore, we would like to minimize jointly the distance between w and w(t) and the approximation of

1. around w(t). If the parameter controls the tradeo between the two terms, we obtain the update rule

w(t+1) = argmin 1 kw w(t)k2 + f(w(t)) + hw w(t); rf(w(t))i :

w 2

Solving the preceding by taking the derivative with respect to w and comparing it to zero yields the same update rule as in Equation ([14.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page185)).

1. Stochastic Gradient Descent

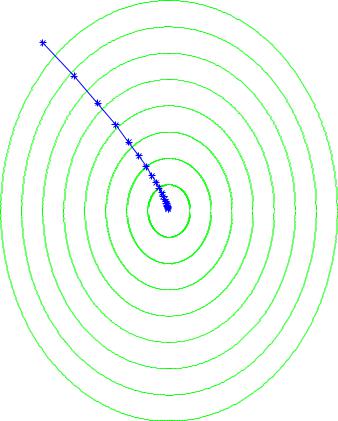


Figure 14.1 An illustration of the gradient descent algorithm. The function to be minimized is 1:25(x1 + 6)2 + (x2 8)2.

14.1.1 Analysis of GD for Convex-Lipschitz Functions

To analyze the convergence rate of the GD algorithm, we limit ourselves to the case of convex-Lipschitz functions (as we have seen, many problems lend themselves easily to this setting). Let w? be any vector and let B be an upper bound on kw?k. It is convenient to think of w? as the minimizer of f(w), but the analysis that follows holds for every w?.

We would like to obtain an upper bound on the suboptimality of our solution

with respect to w?, namely, f(w) f(w?), where w = 1 PT w(t). From the

T t=1

de nition of w, and using Jensen's inequality, we have that

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| f(w) f(w ) = f | T | =1 w | ! | f(w ) |
|  |  | T |  |  |
| ? | 1 | Xt | (t) | ? |
|  |

T

1 X

T t=1 f(w(t)) f(w?)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| = | 1 | T |  | f(w(t)) f(w?) | : | (14.2) |
| T | t=1 |
|  |  | X | |  |  |  |

For every t, because of the convexity of f, we have that

|  |  |
| --- | --- |
| f(w(t)) f(w?) hw(t) w?; rf(w(t))i: | (14.3) |

Combining the preceding we obtain

T

f(w) f(w?) T1 Xhw(t) w?; rf(w(t))i:

t=1

To bound the right-hand side we rely on the following lemma:

|  |  |
| --- | --- |
| 14.1 Gradient Descent | 187 |
|  |  |

lemma 14.1 Let v1; : : : ; vT be an arbitrary sequence of vectors. Any algorithm with an initialization w(1) = 0 and an update rule of the form

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | w(t+1) = w(t)vt | | |  |  |  |  |  | (14.4) |
| satis es |  |  |  |  |  |  |  |  |  |  |  |  |
| T |  | w(t) |  | w?; v | kw?k2 | + |  | | T | v 2 | : | (14.5) |
| Xt |  |  |  |  |  | X |
| h |  |  | ti2 | |  |  |  |  |  |  |
| =1 |  |  |  | 2 t=1 k tk | | |  |  |

In particular, for every B; > 0, if for all t we have that kvtk and if we set

q

=

Proof

B2 , then for every w? with kw?k B we have

2 T

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | T |  | B | | |
|  | Xt |  |  |  |  |
|  |  |  |  |  |
| T | hw(t) w?; vti pT : | | | |
| =1 |
|  |  |  |  |  |

Using algebraic manipulations (completing the square), we obtain:

hw(t) w?; vti = 1 hw(t) w?; vti

1. 21 ( kw(t) w? vtk2 + kw(t) w?k2 + 2kvtk2)
2. 21 ( kw(t+1) w?k2 + kw(t) w?k2) + 2 kvtk2;

where the last equality follows from the de nition of the update rule. Summing the equality over t, we have

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | hw(t) w?; vti = |  | 1 T | |  | kw(t+1) | w?k2 | + kw(t) w?k2 + | T | | kvtk2 | : |
| t=1 | 2 t=1 | | | 2 t=1 | |
| X |  |  |  | X | |  |  |  |  | X | |  |
|  |  |  |  |  |  |  |  |  |  |  | (14.6) | |

The rst sum on the right-hand side is a telescopic sum that collapses to

kw(1) w?k2 kw(T +1) w?k2:

Plugging this in Equation ([14.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page187)), we have

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T | 1 | |  |  | T |
| Xt |  |  |  |  | X |
| 2 (kw(1) w?k2 kw(T +1) w?k2) + | | | 2 |
| hw(t) w?; vti = | kvtk2 |
| =1 |  |  |  |  | t=1 |

T

1 kw(1) w?k2 + X kvtk2

2 2

T

= 21 kw?k2 + 2 X kvtk2;

t=1

where the last equality is due to the de nition w(1) = 0. This proves the rst part of the lemma (Equation ([14.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page187))). The second part follows by upper bounding kw?k by B, kvtk by , dividing by T , and plugging in the value of . 

1. Stochastic Gradient Descent

|  |  |
| --- | --- |
| Lemma [14.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page186) applies to the GD algorithm with vt = rf(w(t)). As we will | |
| show later in Lemma [14.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page190), if f is -Lipschitz, then krf(w(t))k . We therefore | |
| satisfy the lemma's conditions and achieve the following corollary: | |
| corollary 14.2 Let f be a convex, -Lipschitz function, and let w? 2 argminfw:kwk Bg f(w). | |
| If we run the GD algorithm on f for T steps with = | B2 , then the output |
| vector w satis es | q 2 T |

|  |  |  |  |
| --- | --- | --- | --- |
| f(w) f(w?) | B | |  |
| p |  | : |
| T |

Furthermore, for every > 0, to achieve f(w) f(w?) , it su ces to run the GD algorithm for a number of iterations that satis es

B2 2

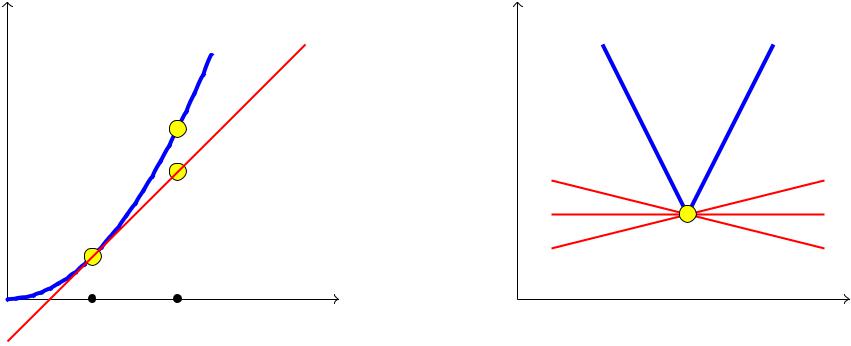
T 2 :

|  |  |  |
| --- | --- | --- |
| 14.2 | Subgradients |  |
|  | The GD algorithm requires that the function f be di erentiable. We now gener- | |
|  | alize the discussion beyond di erentiable functions. We will show that the GD | |
|  | algorithm can be applied to nondi erentiable functions by using a so-called sub- | |
|  | gradient of f(w) at w(t), instead of the gradient. |  |
|  | To motivate the de nition of subgradients, recall that for a convex function f, | |
|  | the gradient at w de nes the slope of a tangent that lies below f, that is, |  |
|  | 8u; f(u) f(w) + hu w; rf(w)i: | (14.7) |
|  | An illustration is given on the left-hand side of Figure [14.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page189). |  |
|  | The existence of a tangent that lies below f is an important property of convex | |
|  | functions, which is in fact an alternative characterization of convexity. |  |
|  | lemma 14.3 Let S be an open convex set. A function f : S ! R is convex i | |
|  | for every w 2 S there exists v such that |  |
|  | 8u 2 S; f(u) f(w) + hu w; vi: | (14.8) |
|  | The proof of this lemma can be found in many convex analysis textbooks (e.g., | |
|  | (Borwein & Lewis 2006)). The preceding inequality leads us to the de nition of | |
|  | subgradients. |  |

definition 14.4 (Subgradients) A vector v that satis es Equation ([14.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page188)) is called a subgradient of f at w. The set of subgradients of f at w is called the di erential set and denoted @f(w).

An illustration of subgradients is given on the right-hand side of Figure [14.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page189). For scalar functions, a subgradient of a convex function f at w is a slope of a line that touches f at w and is not above f elsewhere.

|  |  |
| --- | --- |
| 14.2 Subgradients | 189 |
|  |  |



1. (u)
2. (w)
3. u

Figure 14.2 Left: The right-hand side of Equation ([14.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page188)) is the tangent of f at w. For

a convex function, the tangent lower bounds f. Right: Illustration of several subgradients of a nondi erentiable convex function.

14.2.1 Calculating Subgradients

How do we construct subgradients of a given convex function? If a function is di erentiable at a point w, then the di erential set is trivial, as the following claim shows.

claim 14.5 If f is di erentiable at w then @f(w) contains a single element { the gradient of f at w, rf(w).

Example 14.1 (The Di erential Set of the Absolute Function) Consider the absolute value function f(x) = jxj. Using Claim [14.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page189), we can easily construct the di erential set for the di erentiable parts of f, and the only point that requires special attention is x0 = 0. At that point, it is easy to verify that the subdi erential is the set of all numbers between 1 and 1. Hence:

|  |  |  |  |
| --- | --- | --- | --- |
| @f(x) = | 8 | f1g | if x > 0 |
|  | > | f 1g | if x < 0 |
|  | > |  |  |
|  | < |  |  |
|  | > | [ 1; 1] | if x = 0 |
|  | > |  |  |
|  | : |  |  |

For many practical uses, we do not need to calculate the whole set of subgra-dients at a given point, as one member of this set would su ce. The following claim shows how to construct a sub-gradient for pointwise maximum functions.

claim 14.6 Let g(w) = maxi2[r] gi(w) for r convex di erentiable functions g1; : : : ; gr. Given some w, let j 2 argmaxi gi(w). Then rgj(w) 2 @g(w).

Proof Since gj is convex we have that for all u

gj(u) gj(w) + hu w; rgj(w)i:

Since g(w) = gj(w) and g(u) gj(u) we obtain that

g(u) g(w) + hu w; rgj(w)i;

which concludes our proof.

1. Stochastic Gradient Descent

Example 14.2 (A Subgradient of the Hinge Loss) Recall the hinge loss function from Section [12.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page167), f(w) = maxf0; 1 yhw; xig for some vector x and scalar y. To calculate a subgradient of the hinge loss at some w we rely on the preceding claim and obtain that the vector v de ned in the following is a subgradient of the hinge loss at w:

(

0 if 1 yhw; xi 0

1. =

yx if 1 yhw; xi > 0

14.2.2 Subgradients of Lipschitz Functions

Recall that a function f : A ! R is -Lipschitz if for all u; v 2 A

jf(u) f(v)j ku vk:

The following lemma gives an equivalent de nition using norms of subgradients.

lemma 14.7 Let A be a convex open set and let f : A ! R be a convex function. Then, f is -Lipschitz over A i for all w 2 A and v 2 @f(w) we have that kvk .

Proof Assume that for all v 2 @f(w) we have that kvk . Since v 2 @f(w) we have

f(w) f(u) hv; w ui:

Bounding the right-hand side using Cauchy-Schwartz inequality we obtain

f(w) f(u) hv; w ui kvk kw uk kw uk:

An analogous argument can show that f(u) f(w) kw uk. Hence f is -Lipschitz.

Now assume that f is -Lipschitz. Choose some w 2 A; v 2 @f(w). Since A is open, there exists > 0 such that u = w + v=kvk belongs to A. Therefore, hu w; vi = kvk and ku wk = . From the de nition of the subgradient,

f(u) f(w) hv; u wi = kvk:

On the other hand, from the Lipschitzness of f we have

= ku wk f(u) f(w):

Combining the two inequalities we conclude that kvk .

14.2.3 Subgradient Descent

The gradient descent algorithm can be generalized to nondi erentiable functions by using a subgradient of f(w) at w(t), instead of the gradient. The analysis of the convergence rate remains unchanged: Simply note that Equation ([14.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page186)) is true for subgradients as well.

|  |  |
| --- | --- |
| 14.3 Stochastic Gradient Descent (SGD) | 191 |
|  |  |

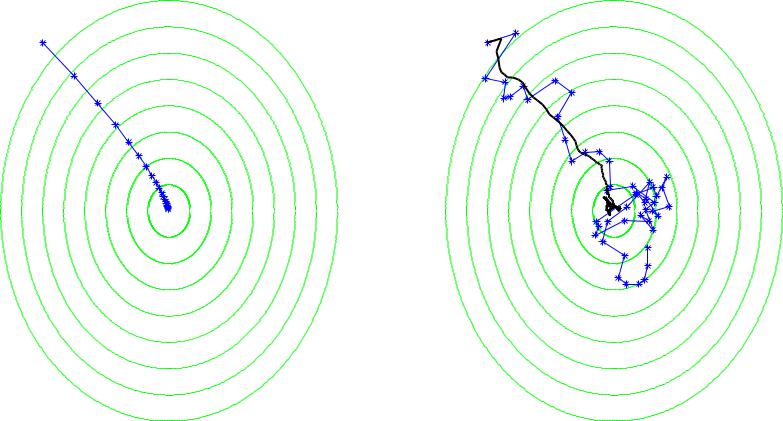


Figure 14.3 An illustration of the gradient descent algorithm (left) and the stochastic gradient descent algorithm (right). The function to be minimized is

1:25(x + 6)2 + (y 8)2. For the stochastic case, the black line depicts the averaged value of w.

14.3 Stochastic Gradient Descent (SGD)

In stochastic gradient descent we do not require the update direction to be based exactly on the gradient. Instead, we allow the direction to be a random vector and only require that its expected value at each iteration will equal the gradient direction. Or, more generally, we require that the expected value of the random vector will be a subgradient of the function at the current vector.

Stochastic Gradient Descent (SGD) for minimizing

1. (w)

parameters: Scalar > 0, integer T > 0

initialize: w(1) = 0

for t = 1; 2; : : : ; T

choose vt at random from a distribution such that E[vt j w(t)] 2 @f(w(t))

update w(t+1) = w(t) vt

output w = 1 PT w(t)

1. t=1

An illustration of stochastic gradient descent versus gradient descent is given in Figure [14.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page191). As we will see in Section [14.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page196), in the context of learning problems, it is easy to nd a random vector whose expectation is a subgradient of the risk function.

14.3.1 Analysis of SGD for Convex-Lipschitz-Bounded Functions

Recall the bound we achieved for the GD algorithm in Corollary [14.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page188). For the stochastic case, in which only the expectation of vt is in @f(w(t)), we cannot directly apply Equation ([14.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page186)). However, since the expected value of vt is a

1. Stochastic Gradient Descent

subgradient of f at w(t), we can still derive a similar bound on the expected output of stochastic gradient descent. This is formalized in the following theorem. theorem 14.8 Let B; > 0. Let f be a convex function and let w? 2 argminw:kwk B f(w).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Assume that SGD is run for T iterations with = | | | | | | |  |  | B2 | . Assume also that for |
|  |  |  |
| all t, | k | vt | k | with probability 1. Then, | | | q 2 T | | | |
|  |  |
|  |  |  |  | E [f(w)] f(w?) | B | | | | | |
|  |  |  |  | p |  |  | : |  |  |
|  |  |  |  | T |  |  |  |

Therefore, for any > 0, to achieve E[f(w)] f(w?) , it su ces to run the SGD algorithm for a number of iterations that satis es

B2 2

T 2 :

Proof Let us introduce the notation v1:t to denote the sequence v1; : : : ; vt. Taking expectation of Equation ([14.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page186)), we obtain

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| v1:T |  | v1:T "T | | =1 |  | # |
|  |  |  |  | T |  |  |
|  |  |  |  | Xt |  |  |
| E [f(w) | f(w?)] | E | 1 | (f(w(t)) |  | f(w?)) : |
|  |  |

Since Lemma [14.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page186) holds for any sequence v1; v2; :::vT , it applies to SGD as well.

By taking expectation of the bound in the lemma we have

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | v1:T " | | T | | =1 | h |  |  |  |  |  |  |  | i# | p | | | T | | |  |  |  |
|  |  |  |  | E |  | 1 | | T |  | w(t) |  | w? | | ; vt | | |  |  |  | B | |  | : |  |  |  |
|  |  |  |  |  |  |  | Xt |  |  |  |  |  | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| It is left to show that | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | " |  | T |  |  |  |  |  |  |  |  |  | " |  |  |  | T |  |  |  |  |  |  |  |  |  |
| E | 1 |  | (f(w(t)) |  | f(w?)) | | | | | E | |  | 1 |  |  | h | w(t) | |  | | | w?; vt | i | ; |
| T | t=1 | T | t=1 | |
| v1:T |  |  |  |  |  | # | v1:T | |  |  |  |  | # |
|  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  | X | |  |  |  |  |  |  |  |  |  |

(14.9)

(14.10)

which we will hereby prove.

Using the linearity of the expectation we have

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | " |  | 1 T | | h | w(t) |  | w?; vt | i | = | 1 T | | E [ | | w(t) |  | w?; vt | ]: |
|  |  |  |  |  |
| T t=1 | | | T t=1 | |
| v1:T |  |  | # | v1:T | h |  | i |  |
|  |  |  |  | X | |  |  |  |  |  |  | X | |  |  |  |  |  |

Next, we recall the law of total expectation: For every two random variables ; ,

and a function g, E [g( )] = E E [g( )j ]: Setting = v1:t and = v1:t 1 we get that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E [ | w(t) |  | w?; vt | ] = E [ | w(t) |  | w?; vt | ] |
| v1:T h |  | i | v1:t h |  | i |  |

1. E E [hw(t) w?; vti j v1:t 1] : v1:t 1 v1:t

Once we know v1:t 1, the value of w(t) is not random any more and therefore

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E E [ | w(t) |  | w?; vt | i j | v1:t |  | 1 | ] = E | w(t) |  | w?; E[vt | j | v1:t |  | 1] | i | : |
| v1:t 1 v1:t h |  |  |  |  | v1:t 1h |  | vt |  |  |  |

|  |  |
| --- | --- |
| 14.4 Variants | 193 |
|  |  |

Since w(t) only depends on v1:t 1 and SGD requires that Evt [vt j w(t)] 2 @f(w(t))

we obtain that Evt [vt j v1:t 1] 2 @f(w(t)). Thus,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | w(t) |  | | w? | ; E[vt | | j | v1: t |  | 1 | ] | E [f(w(t)) | | | | |  | f(w?)]: |
| v1: t 1h |  |  | vt |  |  |  | i | v1: t 1 |  |  |  |  |  |
| Overall, we have shown that | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | E [ | | w(t) | |  | w?; vt ] | | |  | | E [f(w(t)) | | |  | | f(w?)] | | |
|  | v1:T | h |  |  |  |  | i | v1:t 1 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | = E [f(w(t)) | | | |  | | f(w?)] : | | | |
|  |  |  |  |  |  |  |  |  |  |  | v1:T |  |  |  |  |  |

Summing over t, dividing by T , and using the linearity of expectation, we get that Equation ([14.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page192)) holds, which concludes our proof. 

14.4 Variants

In this section we describe several variants of Stochastic Gradient Descent.

14.4.1 Adding a Projection Step

In the previous analyses of the GD and SGD algorithms, we required that the norm of w? will be at most B, which is equivalent to requiring that w? is in the set H = fw : kwk Bg. In terms of learning, this means restricting ourselves to a B-bounded hypothesis class. Yet any step we take in the opposite direction of the gradient (or its expected direction) might result in stepping out of this bound, and there is even no guarantee that w satis es it. We show in the following how to overcome this problem while maintaining the same convergence rate.

The basic idea is to add a projection step; namely, we will now have a two-step update rule, where we rst subtract a subgradient from the current value of w and then project the resulting vector onto H. Formally,

1.. w(t+ 12 ) = w(t) vt

2.. w(t+1) = argminw2H kw w(t+ 12 )k

The projection step replaces the current value of w by the vector in H closest to it.

Clearly, the projection step guarantees that w(t) 2 H for all t. Since H is convex this also implies that w 2 H as required. We next show that the analysis of SGD with projections remains the same. This is based on the following lemma.

lemma 14.9 (Projection Lemma) Let H be a closed convex set and let v be the projection of w onto H, namely,

1. = argmin kx wk2:

x2H

1. Stochastic Gradient Descent Then, for every u 2 H,

kw uk2 kv uk2 0:

Proof By the convexity of H, for every 2 (0; 1) we have that v+ (u v) 2 H.

Therefore, from the optimality of v we obtain

kv wk2 kv + (u v) wk2

1. kv wk2 + 2 hv w; u vi + 2ku vk2:

Rearranging, we obtain

2hv w; u vi ku vk2:

Taking the limit ! 0 we get that

hv w; u vi 0:

Therefore,

kw uk2 = kw v + v uk2

1. kw vk2 + kv uk2 + 2hv w; u vi kv uk2:

Equipped with the preceding lemma, we can easily adapt the analysis of SGD to the case in which we add projection steps on a closed and convex set. Simply note that for every t,

kw(t+1) w?k2 kw(t) w?k2

1. kw(t+1) w?k2 kw(t+ 12 ) w?k2 + kw(t+ 12 ) w?k2 kw(t) w?k2 kw(t+ 12 ) w?k2 kw(t) w?k2:

Therefore, Lemma [14.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page186) holds when we add projection steps and hence the rest of the analysis follows directly.

14.4.2 Variable Step Size

Another variant of SGD is decreasing the step size as a function of t. That is, rather than updating with a constant , we use t. For instance, we can set

= B and achieve a bound similar to Theorem [14.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page192). The idea is that when

t pt

we are closer to the minimum of the function, we take our steps more carefully, so as not to \overshoot" the minimum.

|  |  |
| --- | --- |
| 14.4 Variants | 195 |
|  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 14.4.3 | Other Averaging Techniques |  |  |  |  |  |  |  |
|  | We have set the output vector to be w = | |  | 1 | T | w(t). There are alternative | | |
|  | T | | t=1 |
|  |  | (t) |  |  |  |
|  |  |  |  | random t | | 2 | [t], or outputting the |
|  | approaches such as outputting w for some | | | |
|  | P |  |  |

average of w(t) over the last T iterations, for some 2 (0; 1). One can also take a weighted average of the last few iterates. These more sophisticated averaging schemes can improve the convergence speed in some situations, such as in the case of strongly convex functions de ned in the following.

14.4.4 Strongly Convex Functions\*

In this section we show a variant of SGD that enjoys a faster convergence rate for problems in which the objective function is strongly convex (see De nition [13.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page174) of strong convexity in the previous chapter). We rely on the following claim, which generalizes Lemma [13.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page175).

claim 14.10 If f is -strongly convex then for every w; u and v 2 @f(w) we have

hw u; vi f(w) f(u) + 2 kw uk2:

The proof is similar to the proof of Lemma [13.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page175) and is left as an exercise.

SGD for minimizing a -strongly convex function

Goal: Solve minw2H f(w)

parameter: T

initialize: w(1) = 0

for t = 1; : : : ; T

Choose a random vector vt s.t. E[vtjw(t)] 2 @f(w(t))

Set t = 1=( t)

Set w(t+ 12 ) = w(t) tvt

(t+1) (t+ 1 ) 2

Set w = arg minw2H kw w 2 k

theorem 14.11 Assume that f is -strongly convex and that E[kvtk2] 2.

Let w? 2 argminw2H f(w) be an optimal solution. Then,

|  |  |  |  |
| --- | --- | --- | --- |
| E[f(w)] f(w?) |  | 2 | |
|  |  | (1 + log(T )): |
| 2 T |

Proof Let r(t) = E[vtjw(t)]. Since f is strongly convex and r(t) subgradient set of f at w(t) we have that

hw(t) w?; r(t)i f(w(t)) f(w?) + 2 kw(t) w?k2 :

Next, we show that

is in the

(14.11)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w(t) |  | w?; | r | (t) | i |  | E[kw(t) w?k2 kw(t+1) w?k2] | + | t | | 2: (14.12) |
|  |  |  |  |
| h |  |  |  | 2 t | | 2 | |  |

1. Stochastic Gradient Descent

Since w(t+1) is the projection of w(t+ 12 ) onto H, and w? 2 H we have that kw(t+ 12 ) w?k2 kw(t+1) w?k2. Therefore,

kw(t) w?k2 kw(t+1) w?k2 kw(t) w?k2 kw(t+ 12 ) w?k2 = 2 thw(t) w?; vti t2kvtk2 :

Taking expectation of both sides, rearranging, and using the assumption E[kvtk2] 2 yield Equation ([14.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page195)). Comparing Equation ([14.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page195)) and Equation ([14.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page195)) and summing over t we obtain

T

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Xt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| (E[f(w(t))] f(w?)) | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| =1 | "t=1 k | |  |  | k |  | 2 t | 2 k | | |  |  | k | # | 2 | |  |  |  |
| E | w(t) |  | w(t) | t=1 | t | : |
|  | T | |  | w? | 2 | kw(t+1) w?k2 |  |  | |  | w? | 2 | + 2 | | T |  |
|  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |

Next, we use the de nition t = 1=( t) and note that the rst sum on the right-hand side of the equation collapses to T kw(T +1) w?k2 0. Thus,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| T |  | 2 | T | 1 |  | 2 | |
| X |  |  | Xt |  |  |  |  |
| 2 | | t | 2 (1 + log(T )): | | |
| (E[f(w(t))] f(w?)) | =1 |
| t=1 |  |  |  |  |  |  |

The theorem follows from the preceding by dividing by T and using Jensen's inequality. 

Remark 14.3 Rakhlin, Shamir & Sridharan (2012) derived a convergence rate in which the log(T ) term is eliminated for a variant of the algorithm in which

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| we output the average of the last T =2 iterates, w = |  | 2 |  | T | w(t). Shamir & | |
|  | T |  | t=T =2+1 |
|  |  |  |  | = w(T ). |
| Zhang (2013) have shown that Theorem [14.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page195) holds | even if we output w | | | | |
|  |  | P |  |  |  |

14.5 Learning with SGD

We have so far introduced and analyzed the SGD algorithm for general convex functions. Now we shall consider its applicability to learning tasks.

14.5.1 SGD for Risk Minimization

Recall that in learning we face the problem of minimizing the risk function

LD(w) = E [`(w; z)]:

z D

We have seen the method of empirical risk minimization, where we minimize the empirical risk, LS(w), as an estimate to minimizing LD(w). SGD allows us to take a di erent approach and minimize LD(w) directly. Since we do not know D, we cannot simply calculate rLD(w(t)) and minimize it with the GD method. With SGD, however, all we need is to nd an unbiased estimate of the gradient of

|  |  |
| --- | --- |
| 14.5 Learning with SGD | 197 |
|  |  |

LD(w), that is, a random vector whose conditional expected value is rLD(w(t)).

We shall now see how such an estimate can be easily constructed.

For simplicity, let us rst consider the case of di erentiable loss functions. Hence the risk function LD is also di erentiable. The construction of the random vector vt will be as follows: First, sample z D. Then, de ne vt to be the gradient of the function `(w; z) with respect to w, at the point w(t). Then, by the linearity of the gradient we have

|  |  |  |  |
| --- | --- | --- | --- |
| E[vtjw(t)] = z | E [r`(w(t); z)] = r | z | E [`(w(t); z)] = rLD(w(t)): (14.13) |
|  | D |  | D |

The gradient of the loss function `(w; z) at w(t) is therefore an unbiased estimate of the gradient of the risk function LD(w(t)) and is easily constructed by sampling a single fresh example z D at each iteration t.

The same argument holds for nondi erentiable loss functions. We simply let vt be a subgradient of `(w; z) at w(t). Then, for every u we have

`(u; z) `(w(t); z) hu w(t); vti:

Taking expectation on both sides with respect to z D and conditioned on the value of w(t) we obtain

LD(u) LD(w(t)) = E[`(u; z) `(w(t); z)jw(t)]

E[hu w(t); vtijw(t)] = hu w(t); E[vtjw(t)]i:

It follows that E[vtjw(t)] is a subgradient of LD(w) at w(t).

To summarize, the stochastic gradient descent framework for minimizing the risk is as follows.

Stochastic Gradient Descent (SGD) for minimizing

LD(w)

parameters: Scalar > 0, integer T > 0

initialize: w(1) = 0

for t = 1; 2; : : : ; T

sample z D

pick vt 2 @`(w(t); z)

update w(t+1) = w(t) vt

output w = 1 PT w(t)

T t=1

We shall now use our analysis of SGD to obtain a sample complexity anal-ysis for learning convex-Lipschitz-bounded problems. Theorem [14.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page192) yields the following:

corollary 14.12 Consider a convex-Lipschitz-bounded learning problem with parameters ; B. Then, for every > 0, if we run the SGD method for minimizing

1. Stochastic Gradient Descent

LD(w) with a number of iterations (i.e., number of examples)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | T | | B2 2 | |
|  |  |  |  | 2 |  |
| and with = q | | B2 |  | , then the output of SGD satis es | | | |
| 2 T | |
|  |  |  |  |  | min | | |
|  |  |  |  | E [LD(w)] | w2H LD(w) + : | | |

It is interesting to note that the required sample complexity is of the same order of magnitude as the sample complexity guarantee we derived for regularized loss minimization. In fact, the sample complexity of SGD is even better than what we have derived for regularized loss minimization by a factor of 8.

14.5.2 Analyzing SGD for Convex-Smooth Learning Problems

In the previous chapter we saw that the regularized loss minimization rule also learns the class of convex-smooth-bounded learning problems. We now show that the SGD algorithm can be also used for such problems.

theorem 14.13 Assume that for all z, the loss function `( ; z) is convex, - smooth, and nonnegative. Then, if we run the SGD algorithm for minimizing LD(w) we have that for every w?,

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | [L |  | (w)] |  |  | 1 |  | L (w?) + | w?k2 | : |
| E | D |  | 1 | | k2 T |
|  |  | D |  |

Proof Recall that if a function is -smooth and nonnegative then it is self-bounded:

krf(w)k2 2 f(w):

To analyze SGD for convex-smooth problems, let us de ne z1; : : : ; zT the random samples of the SGD algorithm, let ft( ) = `( ; zt), and note that vt = rft(w(t)). For all t, ft is a convex function and therefore ft(w(t)) ft(w?) hvt; w(t) w?i. Summing over t and using Lemma [14.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page186) we obtain

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | (f (w(t)) f (w?)) | | | | | | | | |  | T | | v ; w(t) | | | | | |  | w? | |  | kw?k2 | | | + |  | |  | T | v 2: |
| X |  |  |  |  |  |  |  |  |  | X | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Xt | |  |
|  | t |  |  |  |  | t | | |  | t=1h t | | | |  |  |  |  |  | i | | | | |  |  |  |  |  |  |  | k tk |
| t=1 |  |  |  |  |  |  |  |  |  |  | 2 | |  | 2 | |  | =1 |
| Combining the preceding with the self-boundedness of ft yields | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |
|  |  |  |  | T | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | T | |  |  |  |  |  |  |
|  |  |  | X | | | (f (w(t)) f (w?)) | | | | | | | |  |  | kw?k2 | | | | | + | |  |  | f (w(t)): | | | | |  |  |
|  |  |  |  | t |  | t | | |  |  |  | |  |  | 2 | |  |  |  |  | Xt | |  |  |  |  |  |  |
|  |  |  | t=1 | | |  |  |  |  |  |  |  |  |  | =1 | | t |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Dividing by T and rearranging, we obtain | | | | | | | | | | | | | | | | | | |  |  |  |  |  | k2 T | | | ! | | |  |  |
|  |  | T | |  | t=1 | |  | t |  | 1T | | | | | | | | | =1 | | t |  |  |  |  |
|  |  | 1 | |  | T | | f (w(t)) | | |  |  | 1 |  |  |  | 1 | |  | T |  | f | (w?) + | | | w?k2 | | |  |  | : |  |
|  |  |  |  |  | X | |  |  |  |  |  |  |  |  |  | Xt | |  | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Next, we take expectation of the two sides of the preceding equation with respect

|  |  |
| --- | --- |
| 14.5 Learning with SGD | 199 |
|  |  |

to z1; : : : ; zT . Clearly, E[ft(w?)] = LD(w?). In addition, using the same argument as in the proof of Theorem [14.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page192) we have that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | "T | | =1 ft(w(t))# | = E | "T | | t=1 LD(w(t))# | E[LD(w)]: |
|  | 1 | | T |  | 1 | | T |  |
|  |  |  | Xt |  |  |  | X |  |
|  |  |  |  |  |  |  |

Combining all we conclude our proof.

As a direct corollary we obtain:

corollary 14.14 Consider a convex-smooth-bounded learning problem with parameters ; B. Assume in addition that `(0; z) 1 for all z 2 Z. For every

> 0, set = 1 . Then, running SGD with T 12B2 = 2 yields

(1+3= )

E[LD(w)] min LD(w) + :

w2H

14.5.3 SGD for Regularized Loss Minimization

We have shown that SGD enjoys the same worst-case sample complexity bound as regularized loss minimization. However, on some distributions, regularized loss minimization may yield a better solution. Therefore, in some cases we may want to solve the optimization problem associated with regularized loss minimization, namely,[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page199)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| w | 2 kwk + LS(w) | | | : |  |
| min |  |  | 2 |  | (14.14) |
|  |  |  |
|  |  |  |  |

Since we are dealing with convex learning problems in which the loss function is convex, the preceding problem is also a convex optimization problem that can be solved using SGD as well, as we shall see in this section.

De ne f(w) = 2 kwk2 + LS(w). Note that f is a -strongly convex function; therefore, we can apply the SGD variant given in Section [14.4.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page195) (with H = Rd). To apply this algorithm, we only need to nd a way to construct an unbiased estimate of a subgradient of f at w(t). This is easily done by noting that if we pick z uniformly at random from S, and choose vt 2 @`(w(t); z) then the expected value of w(t) + vt is a subgradient of f at w(t).

To analyze the resulting algorithm, we rst rewrite the update rule (assuming

1. We divided by 2 for convenience.

1. Stochastic Gradient Descent

that H = Rd and therefore the projection step does not matter) as follows

w(t+1) = w(t) 1t w(t) + vt

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| = | 1 t | | | w(t) | t vt | |
|  | 1 | |  |  | 1 |  |
|  |  |  |  |  |  |  |

1. t t 1 w(t) 1t vt

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | t | | t 1 | | |  | (t 1) |  | t 1 |
| = | t 1 | | | t | 2 | w(t 1) |  | 1 | v |  |
|  |  |  | t |  |  |  |  |
| 1 | | |  |  |  |  |  |  |  |
|  | Xi |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| = t | | | | vi: | |  |  |  |  |
| =1 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

1

t vt

(14.15)

If we assume that the loss function is -Lipschitz, it follows that for all t we have kvtk and therefore k w(t)k , which yields

k w(t) + vtk 2 :

Theorem [14.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page195) therefore tells us that after performing T iterations we have that

E[f(w)] f(w?) 4 2 (1 + log(T )):

T

14.6 Summary

We have introduced the Gradient Descent and Stochastic Gradient Descent algo-rithms, along with several of their variants. We have analyzed their convergence rate and calculated the number of iterations that would guarantee an expected objective of at most plus the optimal objective. Most importantly, we have shown that by using SGD we can directly minimize the risk function. We do so by sampling a point i.i.d from D and using a subgradient of the loss of the current hypothesis w(t) at this point as an unbiased estimate of the gradient (or a subgradient) of the risk function. This implies that a bound on the number of iterations also yields a sample complexity bound. Finally, we have also shown how to apply the SGD method to the problem of regularized risk minimization. In future chapters we show how this yields extremely simple solvers to some optimization problems associated with regularized risk minimization.

14.7 Bibliographic Remarks

SGD dates back to Robbins & Monro (1951). It is especially e ective in large scale machine learning problems. See, for example, (Murata 1998, Le Cun 2004, Zhang 2004, Bottou & Bousquet 2008, Shalev-Shwartz, Singer & Srebro 2007, Shalev-Shwartz & Srebro 2008). In the optimization community it was studied

|  |  |
| --- | --- |
| 14.8 Exercises | 201 |
|  |  |

in the context of stochastic optimization. See, for example, (Nemirovski & Yudin 1978, Nesterov & Nesterov 2004, Nesterov 2005, Nemirovski, Juditsky, Lan & Shapiro 2009, Shapiro, Dentcheva & Ruszczynski 2009).

The bound we have derived for strongly convex function is due to Hazan, Agarwal & Kale (2007). As mentioned previously, improved bounds have been obtained in Rakhlin et al. (2012).

14.8 Exercises

1. Prove Claim [14.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page195). Hint: Extend the proof of Lemma [13.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page175).
2. Prove Corollary [14.14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page199).
3. Perceptron as a subgradient descent algorithm: Let S = ((x1; y1); : : : ; (xm; ym)) 2 (Rd f 1g)m. Assume that there exists w 2 Rd such that for every i 2 [m]

we have yihw; xii 1, and let w? be a vector that has the minimal norm among all vectors that satisfy the preceding requirement. Let R = maxi kxik. De ne a function

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| f(w) = | max (1 | |  | y | i h | w; x | ii | ) : |
| i | [m] |  |  |  |
|  | 2 |  |  |  |  |  |  |  |

Show that minw:kwk kw?k f(w) = 0 and show that any w for which f(w) < 1 separates the examples in S.

Show how to calculate a subgradient of f.

Describe and analyze the subgradient descent algorithm for this case. Com-pare the algorithm and the analysis to the Batch Perceptron algorithm given in Section [9.1.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page120).

1. Variable step size (\*): Prove an analog of Theorem [14.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page192) for SGD with a

|  |  |  |  |
| --- | --- | --- | --- |
|  | B | | |
| variable step size, t = | p |  | . |
| t |

1. Support Vector Machines

In this chapter and the next we discuss a very useful machine learning tool: the support vector machine paradigm (SVM) for learning linear predictors in high dimensional feature spaces. The high dimensionality of the feature space raises both sample complexity and computational complexity challenges.

The SVM algorithmic paradigm tackles the sample complexity challenge by searching for \large margin" separators. Roughly speaking, a halfspace separates a training set with a large margin if all the examples are not only on the correct side of the separating hyperplane but also far away from it. Restricting the algorithm to output a large margin separator can yield a small sample complexity even if the dimensionality of the feature space is high (and even in nite). We introduce the concept of margin and relate it to the regularized loss minimization paradigm as well as to the convergence rate of the Perceptron algorithm.

In the next chapter we will tackle the computational complexity challenge using the idea of kernels.

15.1 Margin and Hard-SVM

Let S = (x1; y1); : : : ; (xm; ym) be a training set of examples, where each xi 2 Rd and yi 2 f 1g. We say that this training set is linearly separable, if there exists

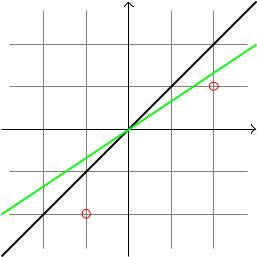
1. halfspace, (w; b), such that yi = sign(hw; xii + b) for all i. Alternatively, this condition can be rewritten as

8i 2 [m]; yi(hw; xii + b) > 0:

All halfspaces (w; b) that satisfy this condition are ERM hypotheses (their 0-1 error is zero, which is the minimum possible error). For any separable training sample, there are many ERM halfspaces. Which one of them should the learner pick?

Consider, for example, the training set described in the picture that follows.

|  |  |
| --- | --- |
| 15.1 Margin and Hard-SVM | 203 |
|  |  |



x

x

While both the dashed-black and solid-green hyperplanes separate the four ex-amples, our intuition would probably lead us to prefer the black hyperplane over the green one. One way to formalize this intuition is using the concept of margin.

The margin of a hyperplane with respect to a training set is de ned to be the minimal distance between a point in the training set and the hyperplane. If a hyperplane has a large margin, then it will still separate the training set even if we slightly perturb each instance.

We will see later on that the true error of a halfspace can be bounded in terms of the margin it has over the training sample (the larger the margin, the smaller the error), regardless of the Euclidean dimension in which this halfspace resides.

Hard-SVM is the learning rule in which we return an ERM hyperplane that separates the training set with the largest possible margin. To de ne Hard-SVM formally, we rst express the distance between a point x to a hyperplane using the parameters de ning the halfspace.

claim 15.1 The distance between a point x and the hyperplane de ned by (w; b) where kwk = 1 is jhw; xi + bj.

Proof The distance between a point x and the hyperplane is de ned as

minfkx vk : hw; vi + b = 0g:

Taking v = x (hw; xi + b)w we have that

hw; vi + b = hw; xi (hw; xi + b)kwk2 + b = 0;

and

kx vk = jhw; xi + bj kwk = jhw; xi + bj:

Hence, the distance is at most jhw; xi + bj. Next, take any other point u on the hyperplane, thus hw; ui + b = 0. We have

kx uk2 = kx v + v uk2

1. kx vk2 + kv uk2 + 2hx v; v ui kx vk2 + 2hx v; v ui
2. kx vk2 + 2(hw; xi + b)hw; v ui
3. kx vk2;

where the last equality is because hw; vi = hw; ui = b. Hence, the distance

1. Support Vector Machines

between x and u is at least the distance between x and v, which concludes our proof. 

On the basis of the preceding claim, the closest point in the training set to the separating hyperplane is mini2[m] jhw; xii + bj. Therefore, the Hard-SVM rule is

argmax min jhw; xii + bj s.t. 8i; yi(hw; xii + b) > 0:

(w;b):kwk=1 i2[m]

Whenever there is a solution to the preceding problem (i.e., we are in the sepa-rable case), we can write an equivalent problem as follows (see Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page214)):

|  |  |
| --- | --- |
| argmax min yi(hw; xii + b): | (15.1) |

(w;b):kwk=1 i2[m]

Next, we give another equivalent formulation of the Hard-SVM rule as a quadratic optimization problem.[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page204)

Hard-SVM

input: (x1; y1); : : : ; (xm; ym)

solve:

(w0; b0) = argmin kwk2 s.t. 8i; yi(hw; xii + b) 1 (15.2)

(w;b)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| output: w^ |  | w0 |  | ^ |  | b0 |
| = | kw0k | ; | b | = | kw0k |

The lemma that follows shows that the output of hard-SVM is indeed the separating hyperplane with the largest margin. Intuitively, hard-SVM searches for w of minimal norm among all the vectors that separate the data and for which jhw; xii + bj 1 for all i. In other words, we enforce the margin to be 1, but now the units in which we measure the margin scale with the norm of w. Therefore, nding the largest margin halfspace boils down to nding w whose norm is minimal. Formally:

lemma 15.2 The output of Hard-SVM is a solution of Equation ([15.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page204)).

Proof Let (w?; b?) be a solution of Equation ([15.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page204)) and de ne the margin achieved by (w?; b?) to be ? = mini2[m] yi(hw?; xii + b?). Therefore, for all i we have

yi(hw?; xii + b?) ?

or equivalently

yi(hw?? ; xii + b?? ) 1:

Hence, the pair ( w? ; b? ) satis es the conditions of the quadratic optimization

? ?

1 A quadratic optimization problem is an optimization problem in which the objective is a convex quadratic function and the constraints are linear inequalities.

|  |  |
| --- | --- |
| 15.1 Margin and Hard-SVM | 205 |
|  |  |

problem given in Equation ([15.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page204)). Therefore, kw0k kw?? k = 1? . It follows that for all i,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ^ |  | 1 |  | 1 | | ? |  |
|  |  |  |  |  |  |  |
| yi(hw^ | ; xii + b) = | kw0k | | yi(hw0 | ; xii + b0) | kw0k |  | : |
| Since kw^k = 1 we obtain that (w^ | | | ^ |  |  |  |  |  |
| ; b) is an optimal solution of Equation ([15.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page204)). | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15.1.1 | The Homogenous Case |  |  |  |  |  |  |  |  |  |  |  |
|  | It is often more convenient to consider homogenous halfspaces, namely, halfspaces | | | | | | | | | | | |
|  | that pass through the origin and are thus de ned by sign(hw; xi), where the bias | | | | | | | | | | | |
|  | term b is set to be zero. Hard-SVM for homogenous halfspaces amounts to solving | | | | | | | | | | | |
|  | min | k | w | k | 2 | s.t. | i; y | ih | w; x | ii | 1: | (15.3) |
|  | w |  |  |  | 8 |  |  |  |

As we discussed in Chapter [9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117), we can reduce the problem of learning nonhomogenous halfspaces to the problem of learning homogenous halfspaces by adding one more feature to each instance of xi, thus increasing the dimension to d + 1.

Note, however, that the optimization problem given in Equation ([15.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page204)) does not regularize the bias term b, while if we learn a homogenous halfspace in Rd+1 using Equation ([15.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page205)) then we regularize the bias term (i.e., the d+ 1 component of the weight vector) as well. However, regularizing b usually does not make a signi cant di erence to the sample complexity.

15.1.2 The Sample Complexity of Hard-SVM

Recall that the VC-dimension of halfspaces in Rd is d + 1. It follows that the sample complexity of learning halfspaces grows with the dimensionality of the problem. Furthermore, the fundamental theorem of learning tells us that if the number of examples is signi cantly smaller than d= then no algorithm can learn an -accurate halfspace. This is problematic when d is very large.

To overcome this problem, we will make an additional assumption on the underlying data distribution. In particular, we will de ne a \separability with margin " assumption and will show that if the data is separable with margin

then the sample complexity is bounded from above by a function of 1= 2. It follows that even if the dimensionality is very large (or even in nite), as long as the data adheres to the separability with margin assumption we can still have a small sample complexity. There is no contradiction to the lower bound given in the fundamental theorem of learning because we are now making an additional assumption on the underlying data distribution.

Before we formally de ne the separability with margin assumption, there is a scaling issue we need to resolve. Suppose that a training set S = (x1; y1); : : : ; (xm; ym) is separable with a margin , namely, the maximal objective value of Equa-tion ([15.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page204)) is at least . Then, for any positive scalar > 0, the training set

1. Support Vector Machines

S0 = ( x1; y1); : : : ; ( xm; ym) is separable with a margin of . That is, a sim-ple scaling of the data can make it separable with an arbitrarily large margin. It follows that in order to give a meaningful de nition of margin we must take into account the scale of the examples as well. One way to formalize this is using the de nition that follows.

definition 15.3 Let D be a distribution over Rd f 1g. We say that D is separable with a ( ; )-margin if there exists (w?; b?) such that kw?k = 1 and such that with probability 1 over the choice of (x; y) D we have that y(hw?; xi+ b?) and kxk . Similarly, we say that D is separable with a ( ; )-margin using a homogenous halfspace if the preceding holds with a halfspace of the form (w?; 0).

In the advanced part of the book (Chapter [26](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page375)), we will prove that the sample complexity of Hard-SVM depends on ( = )2 and is independent of the dimension d. In particular, Theorem [26.13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384) in Section [26.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page383) states the following:

theorem 15.4 Let D be a distribution over Rd f 1g that satis es the ( ; )-separability with margin assumption using a homogenous halfspace. Then, with probability of at least 1 over the choice of a training set of size m, the 0-1 error of the output of Hard-SVM is at most

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r | 4 ( m | | | + | r |  |  | m | | : |
|  |  | = )2 |  |  |  |  | 2 log(2= ) | | |  |
|  | |  |  | | |  |  |  |  | |
| Remark 15.1 (Margin and the Perceptron) | | | | | | | | In Section [9.1.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page120) we have described | | |

and analyzed the Perceptron algorithm for nding an ERM hypothesis with respect to the class of halfspaces. In particular, in Theorem [9.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page120) we upper bounded the number of updates the Perceptron might make on a given training set. It can be shown (see Exercise [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page214)) that the upper bound is exactly ( = )2, where is the radius of examples and is the margin.

15.2 Soft-SVM and Norm Regularization

The Hard-SVM formulation assumes that the training set is linearly separable, which is a rather strong assumption. Soft-SVM can be viewed as a relaxation of the Hard-SVM rule that can be applied even if the training set is not linearly separable.

The optimization problem in Equation ([15.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page204)) enforces the hard constraints yi(hw; xii + b) 1 for all i. A natural relaxation is to allow the constraint to be violated for some of the examples in the training set. This can be modeled by introducing nonnegative slack variables, 1; : : : ; m, and replacing each constraint yi(hw; xii+ b) 1 by the constraint yi(hw; xii+ b) 1 i. That is, i measures by how much the constraint yi(hw; xii+b) 1 is being violated. Soft-SVM jointly minimizes the norm of w (corresponding to the margin) and the average of i (corresponding to the violations of the constraints). The tradeo between the two

|  |  |
| --- | --- |
| 15.2 Soft-SVM and Norm Regularization | 207 |
|  |  |

terms is controlled by a parameter . This leads to the Soft-SVM optimization problem:

Soft-SVM

input: (x1; y1); : : : ; (xm; ym)

parameter: > 0

solve:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | m |  | ! |  |
| min | 2 | Xi |  |  |  |
|  |  |  |
|  |  |  |  |
| kwk + m | |  | i | (15.4) |
| w;b; | =1 |  |
|  |  |  |  |  |  |
| s.t. 8i; yi(hw; xii + b) 1 i | | | | | | and i 0 |

output: w; b

We can rewrite Equation ([15.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207)) as a regularized loss minimization problem.

Recall the de nition of the hinge loss:

`hinge((w; b); (x; y)) = maxf0; 1 y(hw; xi + b)g:

Given (w; b) and a training set S, the averaged hinge loss on S is denoted by

hinge

LS ((w; b)). Now, consider the regularized loss minimization problem:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w;b | k |  | k |  | + LS | ((w; b)) |  |  |
| min |  | w |  | 2 | hinge |  | : | (15.5) |

claim 15.5 Equation ([15.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207)) and Equation ([15.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207)) are equivalent.

Proof Fix some w; b and consider the minimization over in Equation ([15.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207)). Fix some i. Since i must be nonnegative, the best assignment to i would be 0 if yi(hw; xii + b) 1 and would be 1 yi(hw; xii + b) otherwise. In other words, i = `hinge((w; b); (xi; yi)) for all i, and the claim follows. 

We therefore see that Soft-SVM falls into the paradigm of regularized loss minimization that we studied in the previous chapter. A Soft-SVM algorithm, that is, a solution for Equation ([15.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207)), has a bias toward low norm separators. The objective function that we aim to minimize in Equation ([15.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207)) penalizes not only for training errors but also for large norm.

It is often more convenient to consider Soft-SVM for learning a homogenous halfspace, where the bias term b is set to be zero, which yields the following optimization problem:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| w | kwk | 2 | + LS | (w) | ; |  |
| min |  | hinge | |  | (15.6) |
|  |  |  |  |  |

where

m

LhingeS(w) = m1 X maxf0; 1 yhw; xiig:

i=1

1. Support Vector Machines

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15.2.1 | The Sample Complexity of Soft-SVM | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | We now analyze the sample complexity of Soft-SVM for the case of homogenous | | | | | | | | | | | | | | | | | | | | |
|  | halfspaces (namely, the output of Equation ([15.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207))). In Corollary [13.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page179) we derived | | | | | | | | | | | | | | | | | | | | |
|  | a generalization bound for the regularized loss minimization framework assuming | | | | | | | | | | | | | | | | | | | | |
|  | that the loss function is convex and Lipschitz. We have already shown that the | | | | | | | | | | | | | | | | | | | | |
|  | hinge loss is convex so it is only left to analyze the Lipschitzness of the hinge | | | | | | | | | | | | | | | | | | | | |
|  | loss. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | claim 15.6 Let f(w) = maxf0; 1 yhw; xig. Then, f is kxk-Lipschitz. | | | | | | | | | | | | | | | | | |  |  |  |
|  | Proof It is easy to verify that any subgradient of f at w is of the form x where | | | | | | | | | | | | | | | | | | | | |
|  | j j 1. The claim now follows from Lemma [14.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page190). | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Corollary [13.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page179) therefore yields the following: | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  | corollary 15.7 |  | Let D be a distribution over X f0; 1g, where X = fx : | | | | | | | | | | | | | | | | | | |
|  | kxk g. Consider running Soft-SVM (Equation ([15.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207))) on a training set S | | | | | | | | | | | | | | | | | | | | |
|  | Dm and let A(S) be the solution of Soft-SVM. Then, for every u, | | | | | | | | | | | | | | | | | |  |  |  |
|  | S E | m[LDhinge(A(S))] | | LDhinge(u) + kuk2 | | | | | | |  |  | 2 2 | | | | | |  |  |  |
|  | + |  |  | |  | : |  |  |  |  |  |
|  | m | | |  |  |  |
|  | D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Furthermore, since the hinge loss upper bounds the 0 1 loss we also have | | | | | | | | | | | | | | | | | |  |  |  |
|  | S E |  | m[LD0 1(A(S))] | LDhinge(u) + kuk2 | | | | | | | + | 2 2 | |  | : | |  |  |  |  |  |
|  |  | m | | |  |  |  |  |  |
|  | D | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Last, for every B > 0, if we set = q | | | | | 2 2 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | then | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  | B2m |  |  |  |  |  |  |  |  |  |  |  |  |
|  | S Dm D |  | S Dm D |  |  |  |  |  | w:kwk B | | D | | |  |  | r |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | m | | |
|  | E [L0 1(A(S))] |  | E [Lhinge(A(S))] | | | | | |  |  | Lhinge | | |  |  |  |  | 8 2B2 | | | |
|  |  |  | min | (w) + | | | |  |  | : | |
|  |  |  |  |  |

We therefore see that we can control the sample complexity of learning a half-space as a function of the norm of that halfspace, independently of the Euclidean dimension of the space over which the halfspace is de ned. This becomes highly signi cant when we learn via embeddings into high dimensional feature spaces, as we will consider in the next chapter.

Remark 15.2 The condition that X will contain vectors with a bounded norm follows from the requirement that the loss function will be Lipschitz. This is not just a technicality. As we discussed before, separation with large margin is meaningless without imposing a restriction on the scale of the instances. In-deed, without a constraint on the scale, we can always enlarge the margin by multiplying all instances by a large scalar.

15.2.2 Margin and Norm-Based Bounds versus Dimension

The bounds we have derived for Hard-SVM and Soft-SVM do not depend on the dimension of the instance space. Instead, the bounds depend on the norm of the

|  |  |
| --- | --- |
| 15.2 Soft-SVM and Norm Regularization | 209 |
|  |  |

examples, , the norm of the halfspace B (or equivalently the margin parameter

) and, in the nonseparable case, the bounds also depend on the minimum hinge loss of all halfspaces of norm B. In contrast, the VC-dimension of the class of

homogenous halfspaces is d, which implies that the error of an ERM hypothesis

p

decreases as d=m does. We now give an example in which 2B2 d; hence the bound given in Corollary [15.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page208) is much better than the VC bound.

Consider the problem of learning to classify a short text document according to its topic, say, whether the document is about sports or not. We rst need to represent documents as vectors. One simple yet e ective way is to use a bag-of-words representation. That is, we de ne a dictionary of words and set the dimension d to be the number of words in the dictionary. Given a document, we represent it as a vector x 2 f0; 1gd, where xi = 1 if the i'th word in the dictionary appears in the document and xi = 0 otherwise. Therefore, for this problem, the value of 2 will be the maximal number of distinct words in a given document.

A halfspace for this problem assigns weights to words. It is natural to assume that by assigning positive and negative weights to a few dozen words we will be able to determine whether a given document is about sports or not with reasonable accuracy. Therefore, for this problem, the value of B2 can be set to be less than 100. Overall, it is reasonable to say that the value of B2 2 is smaller than 10,000.

On the other hand, a typical size of a dictionary is much larger than 10,000. For example, there are more than 100,000 distinct words in English. We have therefore shown a problem in which there can be an order of magnitude di erence between learning a halfspace with the SVM rule and learning a halfspace using the vanilla ERM rule.

Of course, it is possible to construct problems in which the SVM bound will be worse than the VC bound. When we use SVM, we in fact introduce another form of inductive bias { we prefer large margin halfspaces. While this induc-tive bias can signi cantly decrease our estimation error, it can also enlarge the approximation error.

15.2.3 The Ramp Loss\*

The margin-based bounds we have derived in Corollary [15.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page208) rely on the fact that we minimize the hinge loss. As we have shown in the previous subsection, the

p

term 2B2=m can be much smaller than the corresponding term in the VC

p

bound, d=m. However, the approximation error in Corollary [15.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page208) is measured with respect to the hinge loss while the approximation error in VC bounds is measured with respect to the 0 1 loss. Since the hinge loss upper bounds the 0 1 loss, the approximation error with respect to the 0 1 loss will never exceed that of the hinge loss.

It is not possible to derive bounds that involve the estimation error term

p

2B2=m for the 0 1 loss. This follows from the fact that the 0 1 loss is scale

1. Support Vector Machines

insensitive, and therefore there is no meaning to the norm of w or its margin when we measure error with the 0 1 loss. However, it is possible to de ne a loss

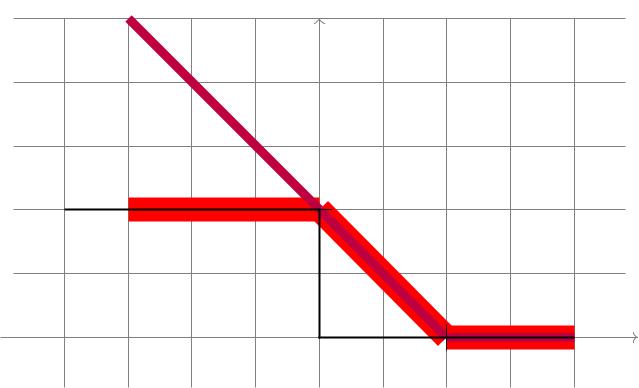
function that on one hand it is scale sensitive and thus enjoys the estimation

p

error 2B2=m while on the other hand it is more similar to the 0 1 loss. One option is the ramp loss, de ned as

`ramp(w; (x; y)) = minf1; `hinge(w; (x; y))g = minf1 ; maxf0; 1 yhw; xigg:

The ramp loss penalizes mistakes in the same way as the 0 1 loss and does not penalize examples that are separated with margin. The di erence between the ramp loss and the 0 1 loss is only with respect to examples that are correctly classi ed but not with a signi cant margin. Generalization bounds for the ramp loss are given in the advanced part of this book (see Appendix [26.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page383)).



`hinge

`0 1 `ramp

1

1 yhw; xi

The reason SVM relies on the hinge loss and not on the ramp loss is that the hinge loss is convex and, therefore, from the computational point of view, minimizing the hinge loss can be performed e ciently. In contrast, the problem of minimizing the ramp loss is computationally intractable.

15.3 Optimality Conditions and \Support Vectors"\*

The name \Support Vector Machine" stems from the fact that the solution of hard-SVM, w0, is supported by (i.e., is in the linear span of) the examples that are exactly at distance 1=kw0k from the separating hyperplane. These vectors are therefore called support vectors. To see this, we rely on Fritz John optimality conditions.

theorem 15.8 Let w0 be as de ned in Equation ([15.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page205)) and let I = fi :

jhw0; xiij = 1g. Then, there exist coe cients 1; : : : ; m such that

X

w0 = ixi:

i2I

The examples fxi : i 2 Ig are called support vectors.

The proof of this theorem follows by applying the following lemma to Equa-tion ([15.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page205)).

|  |  |
| --- | --- |
| 15.4 Duality\* | 211 |
|  |  |

lemma 15.9 (Fritz John) Suppose that

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w | ? | 2 | w | s.t. | 8 | i | 2 | i | (w) |  | 0; |
|  |  |  | argmin f(w) |  |  | [m]; g |  |
| where f; g1; : : : ; gm are di erentiable. Then, | | | | | | | | there exists 2 Rm such that | | | |
| rf(w?) + Pi2I irgi(w?) = 0, where I = fi : gi(w?) = 0g. | | | | | | | | | | |  |

15.4 Duality\*

Historically, many of the properties of SVM have been obtained by considering the dual of Equation ([15.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page205)). Our presentation of SVM does not rely on duality. For completeness, we present in the following how to derive the dual of Equa-tion ([15.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page205)).

We start by rewriting the problem in an equivalent form as follows. Consider the function

|  |  |
| --- | --- |
| 2Rm: 0 =1 i(1 yihw; xii) = | ( |
| m | 0 |
| Xi |
| 1 |
| g(w) = max |  |

We can therefore rewrite Equation ([15.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page205)) as

if 8i; yihw; xii 1 otherwise

:

min kwk2 + g(w) :

w

(15.7)

Rearranging the preceding we obtain that Equation ([15.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page205)) can be rewritten as the problem

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| w 2Rm: 0 2 kwk + | | | | =1 i(1 yihw; xii)! | : |  |
|  | 1 | |  | m |  |  |
| min max | 2 | Xi |  | (15.8) |
|  |  |  |
|  |  |  |  |
|  |  |  |  |  |

Now suppose that we ip the order of min and max in the above equation. This can only decrease the objective value (see Exercise [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page214)), and we have

!

m

min max 1 kwk2 + X i(1 yihw; xii)

w 2Rm: 0 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2Rm: 0 |  |  |  | i=1 | =1 i(1 yihw; xii)! | : |
| w 2 kwk + | | | |
|  |  | 1 | |  | m |  |
| max | min | 2 | Xi |  |
|  |  |  |
|  |  |  |  |
|  |  |  |  |  |

The preceding inequality is called weak duality. It turns out that in our case, strong duality also holds; namely, the inequality holds with equality. Therefore, the dual problem is

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2Rm: 0 w2 kwk + | | | | =1 i(1 yihw; xii)! | : |  |
|  | 1 | |  | m |  |  |
| max min | 2 | Xi |  | (15.9) |
|  |  |  |
|  |  |  |  |
|  |  |  |  |  |

We can simplify the dual problem by noting that once is xed, the optimization

1. Support Vector Machines

problem with respect to w is unconstrained and the objective is di erentiable; thus, at the optimum, the gradient equals zero:

m

m

X

X

w

iyixi

= 0

)

w =

iyixi:

i=1

i=1

This shows us that the solution must be in the linear span of the examples, a fact we will use later to derive SVM with kernels. Plugging the preceding into Equation ([15.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page211)) we obtain that the dual problem can be rewritten as

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rm: 0 02 | | | | | | iyixi | |  | 2 | +i 01 yi \* | | | jyjxj; xi+11 : |  |
|  |  | 1 | | |  | m |  |  | m | |  |  |  |
|  |  |  |  |  |  |  |
| 2 |  | i=1 |  |  | i=1 | |  | j |  |
|  |  |  |  |  | X | |  |  |  | X | |  | X |  |
| max | @ |  |  |  |  |  |  |  |  |  |  | @ | AA | (15.10) |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Rearranging | yields the dual problem | | | | | | | | | | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | i=1 j=1 i jyiyjhxj; xii1 : | |  |
|  |  |  | 2Rm: 0 | | | | 0 =1 | | | i 2 | |  |
|  |  |  |  |  |  |  |  | m | | 1 | | m m |  |  |
|  |  |  |  |  |  |  | @Xi | | |  |  | X X | A |  |
|  |  |  |  |  | max | |  | | (15.11) |
|  |  |  |  |  |  |  |  |  |  |

Note that the dual problem only involves inner products between instances and does not require direct access to speci c elements within an instance. This prop-erty is important when implementing SVM with kernels, as we will discuss in the next chapter.

15.5 Implementing Soft-SVM Using SGD

In this section we describe a very simple algorithm for solving the optimization problem of Soft-SVM, namely,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w 2 kwk | | |  | + m | | maxf0; 1 yhw; xiig! | : |  |
|  | | |  | 1 | | m |  |  |
| min |  |  | 2 |  |  | Xi |  | (15.12) |
|  |  |  |  |  |  |
|  |  |  |  |  | =1 |  |
|  |  |  |  |  |  |  |  |

We rely on the SGD framework for solving regularized loss minimization prob-lems, as described in Section [14.5.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page199).

Recall that, on the basis of Equation ([14.15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page200)), we can rewrite the update rule of SGD as

t

w(t+1) = 1 X v ; j

where vj is a subgradient of the loss function at w(j) on the random example chosen at iteration j. For the hinge loss, given an example (x; y), we can choose vj to be 0 if yhw(j); xi 1 and vj = y x otherwise (see Example [14.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page190)). Denoting (t) = Pj<t vj we obtain the following procedure.

|  |  |
| --- | --- |
| 15.6 Summary | 213 |
|  |  |

SGD for Solving Soft-SVM

goal: Solve Equation ([15.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page212))

parameter: T

initialize: (1) = 0

for t = 1; : : : ; T

Let w(t) = 1t (t)

Choose i uniformly at random from [m]

If (yihw(t); xii < 1)

Set (t+1) = (t) + yixi

Else

Set (t+1) = (t)

output: w = T1 PTt=1 w(t)

15.6 Summary

SVM is an algorithm for learning halfspaces with a certain type of prior knowl-edge, namely, preference for large margin. Hard-SVM seeks the halfspace that separates the data perfectly with the largest margin, whereas soft-SVM does not assume separability of the data and allows the constraints to be violated to some extent. The sample complexity for both types of SVM is di erent from the sample complexity of straightforward halfspace learning, as it does not depend on the dimension of the domain but rather on parameters such as the maximal norms of x and w.

The importance of dimension-independent sample complexity will be realized in the next chapter, where we will discuss the embedding of the given domain into some high dimensional feature space as means for enriching our hypothesis class. Such a procedure raises computational and sample complexity problems. The latter is solved by using SVM, whereas the former can be solved by using SVM with kernels, as we will see in the next chapter.

15.7 Bibliographic Remarks

SVMs have been introduced in (Cortes & Vapnik 1995, Boser, Guyon & Vapnik 1992). There are many good books on the theoretical and practical aspects of SVMs. For example, (Vapnik 1995, Cristianini & Shawe-Taylor 2000, Sch•olkopf

1. Smola 2002, Hsu, Chang & Lin 2003, Steinwart & Christmann 2008). Using SGD for solving soft-SVM has been proposed in Shalev-Shwartz et al. (2007).

1. Support Vector Machines

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15.8 | Exercises |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 1. Show that the hard-SVM rule, namely, | | | | | | | | | | |  |  | 8 |  | i h |  |  | ii |  |
|  | (w;b):kwk=1 | i2[m] | | jh | w; x | | ii | + b | | | j | s.t. |  |  |  |  | + b) > 0; |
|  | argmax | min | |  |  |  |  | i; y ( w; x | | | | |  |
|  | is equivalent to the following formulation: | | | | | | | | | | | |  |  |  |  |  |  |  |  |
|  |  |  | argmax | | | |  |  | min | | | yi(hw; xii + b): | | | | | |  |  | (15.13) |
|  |  |  | (w;b):kwk=1 i2[m] | | | | | | | | |  |  |  |
|  | Hint: De ne G = f(w; b) : 8i; yi(hw; xii + b) > 0g. | | | | | | | | | | | | | | | |  |  |  |  |
|  | 1. Show that |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | argmax | | | |  | min | | | | y ( w; x | | | ii | + b) | 2 G | | |  |
|  |  |  |  | i | 2 | [m] | | i h |  |  |  |  |
|  |  | (w;b):kwk=1 | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
|  | 2. Show that 8(w; b) 2 G; | | | |  |  |  | ii | |  |  | i [m] jh | | | |  | ii |  |  | j |
|  |  | i [m] | | i h | | w; x | |  |  | w; x | + b | |
|  |  | min y ( | | | |  |  | + b) = min | | | | |  |  |  |
|  |  | 2 |  |  |  |  |  |  |  |  |  |  | 2 |  |  |  |  |  |  |  |

1. Margin and the Perceptron Consider a training set that is linearly sep-arable with a margin and such that all the instances are within a ball of radius . Prove that the maximal number of updates the Batch Perceptron

algorithm given in Section [9.1.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page120) will make when running on this training set is ( = )2.

1. Hard versus soft SVM: Prove or refute the following claim:

There exists > 0 such that for every sample S of m > 1 examples, which is separable by the class of homogenous halfspaces, the hard-SVM and the soft-SVM (with parameter ) learning rules return exactly the same weight vector.

1. Weak duality: Prove that for any function f of two vector variables x 2 X ; y 2 Y, it holds that

min max f(x; y) max min f(x; y):

x2X y2Y y2Y x2X

1. Kernel Methods

In the previous chapter we described the SVM paradigm for learning halfspaces in high dimensional feature spaces. This enables us to enrich the expressive power of halfspaces by rst mapping the data into a high dimensional feature space, and then learning a linear predictor in that space. This is similar to the AdaBoost algorithm, which learns a composition of a halfspace over base hy-potheses. While this approach greatly extends the expressiveness of halfspace predictors, it raises both sample complexity and computational complexity chal-lenges. In the previous chapter we tackled the sample complexity issue using the concept of margin. In this chapter we tackle the computational complexity challenge using the method of kernels.

We start the chapter by describing the idea of embedding the data into a high dimensional feature space. We then introduce the idea of kernels. A kernel is a type of a similarity measure between instances. The special property of kernel similarities is that they can be viewed as inner products in some Hilbert space (or Euclidean space of some high dimension) to which the instance space is vir-tually embedded. We introduce the \kernel trick" that enables computationally e cient implementation of learning, without explicitly handling the high dimen-sional representation of the domain instances. Kernel based learning algorithms, and in particular kernel-SVM, are very useful and popular machine learning tools. Their success may be attributed both to being exible for accommodating domain speci c prior knowledge and to having a well developed set of e cient implementation algorithms.

16.1 Embeddings into Feature Spaces

The expressive power of halfspaces is rather restricted { for example, the follow-ing training set is not separable by a halfspace.

Let the domain be the real line; consider the domain points f 10; 9; 8; : : : ; 0;

1; : : : ; 9; 10g where the labels are +1 for all x such that jxj > 2 and 1 otherwise.

To make the class of halfspaces more expressive, we can rst map the original instance space into another space (possibly of a higher dimension) and then learn a halfspace in that space. For example, consider the example mentioned previously. Instead of learning a halfspace in the original representation let us

|  |  |  |
| --- | --- | --- |
| 216 | Kernel Methods |  |
|  |  |  |
|  | rst de ne a mapping : R ! R2 as follows: |  |
|  | (x) = (x; x2): |  |
|  | We use the term feature space to denote the range of . After applying the | |
|  | data can be easily explained using the halfspace h(x) = sign(hw; | (x)i b), |
|  | where w = (0; 1) and b = 5. |  |
|  | The basic paradigm is as follows: |  |
|  | 1. Given some domain set X and a learning task, choose a mapping | : X ! F, |

for some feature space F, that will usually be Rn for some n (however, the range of such a mapping can be any Hilbert space, including such spaces of in nite dimension, as we will show later).

2. Given a sequence of labeled examples, S = (x1; y1); : : : ; (xm; ym), create the

^

image sequence S = ( (x1); y1); : : : ; ( (xm); ym).

^

3. Train a linear predictor h over S.

4. Predict the label of a test point, x, to be h( (x)).

Note that, for every probability distribution D over X Y, we can readily de ne its image probability distribution D over F Y by setting, for every subset A F Y, D (A) = D( 1(A)).[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page216) It follows that for every predictor h over the feature space, LD (h) = LD(h ), where h is the composition of h onto .

The success of this learning paradigm depends on choosing a good for a given learning task: that is, a that will make the image of the data distribution (close to being) linearly separable in the feature space, thus making the resulting algorithm a good learner for a given task. Picking such an embedding requires prior knowledge about that task. However, often some generic mappings that enable us to enrich the class of halfspaces and extend its expressiveness are used. One notable example is polynomial mappings, which are a generalization of the

we have seen in the previous example.

Recall that the prediction of a standard halfspace classi er on an instance x is based on the linear mapping x 7! wh; xi. We can generalize linear mappings to a polynomial mapping, x 7!p(x), where p is a multivariate polynomial of degree k. For simplicity, consider rst the case in which x is 1 dimensional.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | k |  |  |  |  |  |
| In that case, p(x) = | j=0 wjxj, where w 2 Rk+1 is the vector of coe cients | | | | | |
| need to learn. We can rewrite p(x) = | | h | w; | (x) | i | where |
| of the polynomial weP |  |  |  |  |

1. R ! Rk+1 is the mapping x 7!(1; x; x2; x3; : : : ; xk). It follows that learning a k degree polynomial over R can be done by learning a linear mapping in the (k + 1) dimensional feature space.

More generally, a degree k multivariate polynomial from Rn to R can be writ-

ten as

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| J2[X |  |  | r |  |
|  | Y |  |
| p(x) = |  |  | wJxJi : | (16.1) |
| n]r:r |  | k | i=1 |  |

1 This is de ned for every A such that 1(A) is measurable with respect to D.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  | 16.2 The Kernel Trick | | 217 |
|  |  | | | | | | |  | | |
|  | As before, we can rewrite p(x) = hw; | | | | | | | (x)i where now : Rn ! Rd is such | | |
|  | that for | every J | | 2 | [n]r, r |  | k, the coordinate of | | (x) associated with J is the | |
|  | Q | r |  |  |  |  |  |
|  | monomial | | i=1 xJi . | | |  |  |  |  |  |
|  | Naturally, polynomial-based classi ers yield much richer hypothesis classes | | | | | | | | | |
|  | than halfspaces. We have seen at the beginning of this chapter an example in | | | | | | | | | |
|  | which the training set, in its original domain (X = R), cannot be separable | | | | | | | | | |
|  | by a halfspace, but after the embedding x 7!(x; | | | | | | | | x2) it is perfectly separable. | |
|  | So, while the classi er is always linear in the feature space, it can have highly | | | | | | | | | |
|  | nonlinear behavior on the original space from which instances were sampled. | | | | | | | | | |
|  | In general, we can choose any feature mapping | | | | | | | | that maps the original in- | |
|  | stances into some Hilbert space.[2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page217) The Euclidean space Rd is a Hilbert space for | | | | | | | | | |
|  | any nite d. But there are also in nite dimensional Hilbert spaces (as we shall | | | | | | | | | |
|  | see later on in this chapter). | | | | | |  |  |  |  |
|  | The bottom line of this discussion is that we can enrich the class of halfspaces | | | | | | | | | |
|  | by rst applying a nonlinear mapping, | | | | | | | , that maps the instance space into some | | |
|  | feature space, and then learning a halfspace in that feature space. However, if | | | | | | | | | |
|  | the range of | | is a high dimensional space we face two problems. First, the VC- | | | | | | | |
|  | dimension of halfspaces in Rn is n + 1, and therefore, if the range of | | | | | | | | | is very |
|  | large, we need many more samples in order to learn a halfspace in the range | | | | | | | | | |
|  | of . Second, from the computational point of view, performing calculations in | | | | | | | | | |
|  | the high dimensional space might be too costly. In fact, even the representation | | | | | | | | | |
|  | of the vector w in the feature space can be unrealistic. The rst issue can be | | | | | | | | | |
|  | tackled using the paradigm of large margin (or low norm predictors), as we | | | | | | | | | |
|  | already discussed in the previous chapter in the context of the SVM algorithm. | | | | | | | | | |
|  | In the following section we address the computational issue. | | | | | | | | |  |
| 16.2 | The Kernel Trick | | | | |  |  |  |  |  |
|  | We have seen that embedding the input space into some high dimensional feature | | | | | | | | | |
|  | space makes halfspace learning more expressive. However, the computational | | | | | | | | | |
|  | complexity of such learning may still pose a serious hurdle { computing linear | | | | | | | | | |
|  | separators over very high dimensional data may be computationally expensive. | | | | | | | | | |
|  | The common solution to this concern is kernel based learning. The term \kernels" | | | | | | | | | |
|  | is used in this context to describe inner products in the feature space. Given | | | | | | | | | |
|  | an embedding | | | of some domain space X into some Hilbert space, we de ne | | | | | | |
|  | the kernel function K(x; x0) = h (x); | | | | | | | (x0)i. One can think of K as specifying | | |
|  | similarity between instances and of the embedding | | | | | | | | as mapping the domain set | |

1. A Hilbert space is a vector space with an inner product, which is also complete. A space is complete if all Cauchy sequences in the space converge.

p

In our case, the norm kw k is de ned by the inner product hw; wi. The reason we require the range of to be in a Hilbert space is that projections in a Hilbert space are well de ned. In particular, if M is a linear subspace of a Hilbert space, then every x in the Hilbert space can be written as a sum x = u + v where u 2 M and hv; wi = 0 for all

w 2 M. We use this fact in the proof of the representer theorem given in the next section.

1. Kernel Methods

X into a space where these similarities are realized as inner products. It turns out that many learning algorithms for halfspaces can be carried out just on the basis of the values of the kernel function over pairs of domain points. The main advantage of such algorithms is that they implement linear separators in high

dimensional feature spaces without having to specify points in that space or expressing the embedding explicitly. The remainder of this section is devoted to constructing such algorithms.

In the previous chapter we saw that regularizing the norm of w yields a small sample complexity even if the dimensionality of the feature space is high. Inter-estingly, as we show later, regularizing the norm of w is also helpful in overcoming the computational problem. To do so, rst note that all versions of the SVM op-timization problem we have derived in the previous chapter are instances of the following general problem:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w | h | w; | 1 | i | h | w; | (x | m i | k | k | )); | (16.2) |
| min (f ( | | (x | ) | ; : : : ; | ) | ) + R( | w |

where f : Rm ! R is an arbitrary function and R : R+ ! R is a monotoni-cally nondecreasing function. For example, Soft-SVM for homogenous halfspaces (Equation ([15.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207))) can be derived from Equation ([16.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page218)) by letting R(a) = a2 and f(a1; : : : ; am) = m1 Pi maxf0; 1 yiaig. Similarly, Hard-SVM for nonhomogenous halfspaces (Equation ([15.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page204))) can be derived from Equation ([16.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page218)) by letting R(a) = a2 and letting f(a1; : : : ; am) be 0 if there exists b such that yi(ai +b) 1 for all i, and f(a1; : : : ; am) = 1 otherwise.

The following theorem shows that there exists an optimal solution of Equa-tion ([16.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page218)) that lies in the span of f (x1); : : : ; (xm)g.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| theorem 16.1 (Representer Theorem) Assume that is a | | | mapping from | | | | X | to | |
|  | m |  |  |  |  |
| a Hilbert space. Then, there exists a vector | 2 R | m such that w = | | Pi=1 |  |  | (x | | ) |
| is an optimal solution of Equation ([16.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page218)). |  |  |  | i |  | i |  |

Proof Let w? be an optimal solution of Equation ([16.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page218)). Because w? is an element of a Hilbert space, we can rewrite w? as

m

X

w? = i (xi) + u;

i=1

where hu; (xi)i = 0 for all i. Set w = w? u. Clearly, kw?k2 = kwk2 + kuk2, thus kwk kw?k. Since R is nondecreasing we obtain that R(kwk) R(kw?k). Additionally, for all i we have that

hw; (xi)i = hw? u; (xi)i = hw?; (xi)i;

hence

f (hw; (x1)i ; : : : ; hw; (xm)i) = f (hw?; (x1)i ; : : : ; hw?; (xm)i) :

We have shown that the objective of Equation ([16.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page218)) at w cannot be larger than the objective at w? and therefore w is also an optimal solution. Since

Pm

w = i=1 i (xi) we conclude our proof.

|  |  |
| --- | --- |
| 16.2 The Kernel Trick | 219 |
|  |  |

On the basis of the representer theorem we can optimize Equation ([16.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page218)) with respect to the coe cients instead of the coe cients w as follows. Writing

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m |  |  |  |  |  |  |  |  |  |
| w = Pj=1 j | (xj) we have that for all i | | | | |  | m |  |  |
| hw; | (xi)i = | | \* | j (xj); (xi)+ | | = |  | jh (xj); | (xi)i: |
|  |  |  | Xj |  |  | Xj | |  |  |
|  |  |  |  |  |  |  | =1 |  |  |
| Similarly, |  |  |  |  |  |  |  |  |  |
| kwk2 = \* | | j (xj); | | | j (xj)+ = m | | | i jh (xi); (xj)i: | |
|  |  | Xj |  |  | Xj | X | |  |  |
|  |  |  |  |  |  | i;j=1 | |  |  |
| Let K(x; x0) = h | | (x); | (x0)i be a function that implements the kernel function | | | | | | |
| with respect to the embedding | | | | | . Instead of solving Equation ([16.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page218)) we can solve | | | | |
| the equivalent problem | | | 0j=1 |  |  |  |  |  | 1 |
|  | 2Rm | | j | j 1 | =1 | j | j m |
|  |  |  | m |  |  | m |  |  | A |
|  | min f | | @X | K(x ; x ); : : : ; | | Xj | K(x ; x ) | |
|  |  |  |  |

0v 1

1. m

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | u | X |  |  |  |
| + R | @t | i jK(xj; xi) | A | : | (16.3) |
|  |  |  |  |

i;j=1

To solve the optimization problem given in Equation ([16.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page219)), we do not need any direct access to elements in the feature space. The only thing we should know is how to calculate inner products in the feature space, or equivalently, to calculate the kernel function. In fact, to solve Equation ([16.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page219)) we solely need to know the value of the m m matrix G s.t. Gi;j = K(xi; xj), which is often called the Gram matrix.

In particular, specifying the preceding to the Soft-SVM problem given in Equa-tion ([15.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page207)), we can rewrite the problem as

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2Rm | G + m | | =1 |  | 0; 1 yi(G )i | ! ; |  |
|  |  | 1 | m |  |  |  |  |
| min | T |  | Xi | max | (16.4) |
|  |  |  |  |
|  |  |  |  |  |

where (G )i is the i'th element of the vector obtained by multiplying the Gram matrix G by the vector . Note that Equation ([16.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page219)) can be written as quadratic programming and hence can be solved e ciently. In the next section we describe an even simpler algorithm for solving Soft-SVM with kernels.

Once we learn the coe cients we can calculate the prediction on a new instance by

|  |  |
| --- | --- |
| m | m |
| X | Xj |
| hw; (x)i =jh (xj); (x)i = | jK(xj; x): |
| j=1 | =1 |

The advantage of working with kernels rather than directly optimizing w in the feature space is that in some situations the dimension of the feature space

x2

e 2 xn. Then,

1. Kernel Methods

is extremely large while implementing the kernel function is very simple. A few examples are given in the following.

Example 16.1 (Polynomial Kernels) The k degree polynomial kernel is de ned to be

K(x; x0) = (1 + hx; x0i)k:

Now we will show that this is indeed a kernel function. That is, we will show

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| that there exists a mapping | h | from the original space to some higher dimensional | | | | |
| space for which K(x; x0) = | (x); | i |  | 0 | 0 |
|  | (x0) | . For simplicity, denote x | | = x0 = 1. |
| Then, we have |  |  |  |  |  |  |
| K(x; x0) = (1 + hx; x0i)k = (1 + hx; x0i)(1 + hx; x0i) | | | | | |  |
| = 0 n | xjxj01 | | 0 n | | xjxj01 |  |
| @X |  | A | @Xj | | A |  |
| j=0 |  |  |  | =0 |  |  |

k

* 1. Y

1. xJi x0Ji

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| J2f0;1;:::;ngk i=1 | | | | | |  |  |  |  |  |
| J2f0X; | | | | g | k |  | k |  |  |  |
| Y |  | Y |  |  |  |
| = |  |  |  |  |  | xJi | xJ0 | i : |  |  |
|  |  | 1;:::;n | | | k i=1 |  | i=1 |  |  |  |
| Now, if we de ne : R | n | ! R | (n+1)k | | |  |  |  | k |  |
|  | k |  | such that for J 2 f0; 1; : : : ; ng | | | |  | there is an |
|  |  | KQ(xi | | ; x0) = h (x); (x0)i: | | | | |  |  |
| element of (x) that equals | | | =1 xJi , we obtain that | | | | | |  |  |

Since contains all the monomials up to degree k, a halfspace over the range of corresponds to a polynomial predictor of degree k over the original space. Hence, learning a halfspace with a k degree polynomial kernel enables us to learn polynomial predictors of degree k over the original space.

Note that here the complexity of implementing K is O(n) while the dimension of the feature space is on the order of nk.

Example 16.2 (Gaussian Kernel) Let the original instance space be R and consider the mapping where for each nonnegative integer n 0 there exists

an element (x)n that equals p1

n!

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  | |  |  |  |  |  | x2 | | |  |  | p1n! | | | |
| h (x); (x0)i = n=0 | | p1n! e 2 | | | | | |  | xn | | |
| X | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | |  | | |  |  |  |  |  |  |  |  | |  |  |  |
| x2 | | | +(x0)2 | | | | | 1 | |  | | ( | xx )n | | |  | |
| = e | |  |  | 2 | |  |  | n=0 | |  | n0! | |
|  |  |  |  |  |  |  |  | X | |  |  |  |  |  |  |  |  |

(x0)2

e 2 (x0)n

kx x0k2

= e 2 :

Here the feature space is of in nite dimension while evaluating the kernel is very

|  |  |
| --- | --- |
| 16.2 The Kernel Trick | 221 |
|  |  |

simple. More generally, given a scalar > 0, the Gaussian kernel is de ned to be

|  |  |  |
| --- | --- | --- |
| K(x; x0) = e | kx x0k2 | |
| 2 | : |

Intuitively, the Gaussian kernel sets the inner product in the feature space between x; x0 to be close to zero if the instances are far away from each other (in the original domain) and close to 1 if they are close. is a parameter that controls the scale determining what we mean by \close." It is easy to verify that K implements an inner product in a space in which for any n and any monomial

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | kxk2 | n |
| of order k there exists an element of (x) that equals | p1n! | e | 2 | Qi=1 xJi . |

Hence, we can learn any polynomial predictor over the original space by using a Gaussian kernel.

Recall that the VC-dimension of the class of all polynomial predictors is in - nite (see Exercise [12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page81)). There is no contradiction, because the sample complexity required to learn with Gaussian kernels depends on the margin in the feature space, which will be large if we are lucky, but can in general be arbitrarily small.

The Gaussian kernel is also called the RBF kernel, for \Radial Basis Func-tions."

|  |  |  |  |
| --- | --- | --- | --- |
| 16.2.1 | Kernels as a Way to Express Prior Knowledge | |  |
|  | As we discussed previously, a feature mapping, | , may be viewed as expanding | |
|  | the class of linear classi ers to a richer class (corresponding to linear classi ers | | |
|  | over the feature space). However, as discussed in the book so far, the suitability | | |
|  | of any hypothesis class to a given learning task depends on the nature of that | | |
|  | task. One can therefore think of an embedding | as a way to express and utilize | |
|  | prior knowledge about the problem at hand. For example, if we believe that | | |
|  | positive examples can be distinguished by some ellipse, we can de ne | | to be all |
|  | the monomials up to order 2, or use a degree 2 polynomial kernel. | |  |
|  | As a more realistic example, consider the task of learning to nd a sequence of | | |
|  | characters (\signature") in a le that indicates whether it contains a virus or not. | | |
|  | Formally, let Xd be the set of all strings of length at most d over some alphabet | | |
|  | set . The hypothesis class that one wishes to learn is H = fhv : v 2 Xdg, where, | | |
|  | for a string x 2 Xd, hv(x) is 1 i v is a substring of x (and hv(x) = 1 otherwise). | | |
|  | Let us show how using an appropriate embedding this class can be realized by | | |
|  | linear classi ers over the resulting feature space. Consider a mapping | | to a space |
|  | Rs where s = jXdj, so that each coordinate of | (x) corresponds to some string v | |
|  | and indicates whether v is a substring of x (that is, for every x 2 Xd, | | (x) is a |

vector in f0; 1gjXdj). Note that the dimension of this feature space is exponential in d. It is not hard to see that every member of the class H can be realized by composing a linear classi er over (x), and, moreover, by such a halfspace whose norm is 1 and that attains a margin of 1 (see Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page225)). Furthermore, for every x 2 X , k (x)k = O(d). So, overall, it is learnable using SVM with a sample

1. Kernel Methods

complexity that is polynomial in d. However, the dimension of the feature space is exponential in d so a direct implementation of SVM over the feature space is problematic. Luckily, it is easy to calculate the inner product in the feature space

(i.e., the kernel function) without explicitly mapping instances into the feature

space. Indeed, K(x; x0) is simply the number of common substrings of x and x0, which can be easily calculated in time polynomial in d.

This example also demonstrates how feature mapping enables us to use halfspaces for nonvectorial domains.

16.2.2 Characterizing Kernel Functions\*

As we have discussed in the previous section, we can think of the speci cation of the kernel matrix as a way to express prior knowledge. Consider a given similarity function of the form K : X X ! R. Is it a valid kernel function? That is, does it represent an inner product between (x) and (x0) for some feature mapping

? The following lemma gives a su cient and necessary condition.

lemma 16.2 A symmetric function K : X X ! R implements an inner product in some Hilbert space if and only if it is positive semide nite; namely, for all x1; : : : ; xm, the Gram matrix, Gi;j = K(xi; xj), is a positive semide nite matrix.

Proof It is trivial to see that if K implements an inner product in some Hilbert space then the Gram matrix is positive semide nite. For the other direction, de ne the space of functions over X as RX = ff : X ! Rg. For each x 2 X let (x) be the function x 7!K( ; x). De ne a vector space by taking all linear combinations of elements of the form K( ; x). De ne an inner product on this vector space to be

\* +

X X

iK( ; xi); jK( ; x0j)

1. j

X

1. i jK(xi; x0j):

i;j

This is a valid inner product since it is symmetric (because K is symmetric), it is linear (immediate), and it is positive de nite (it is easy to see that K(x; x) 0 with equality only for (x) being the zero function). Clearly,

1. (x); (x0)i = hK( ; x); K( ; x0)i = K(x; x0);

which concludes our proof.

16.3 Implementing Soft-SVM with Kernels

Next, we turn to solving Soft-SVM with kernels. While we could have designed an algorithm for solving Equation ([16.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page219)), there is an even simpler approach that

|  |  |
| --- | --- |
| 16.3 Implementing Soft-SVM with Kernels | 223 |
|  |  |

directly tackles the Soft-SVM optimization problem in the feature space,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w 2 kwk | | |  | + m | | | maxf0; 1 yhw; (xi)ig! | ; |  |
|  |  | |  | 1 | |  | m |  |  |
| min |  |  | 2 |  |  |  | Xi |  | (16.5) |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  | =1 |  |
|  |  |  |  |  |  |  |  |  |

while only using kernel evaluations. The basic observation is that the vector w(t) maintained by the SGD procedure we have described in Section [15.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page212) is always in the linear span of f (x1); : : : ; (xm)g. Therefore, rather than maintaining w(t) we can maintain the corresponding coe cients .

Formally, let K be the kernel function, namely, for all x; x0, K(x; x0) =

1. (x); (x0)i. We shall maintain two vectors in Rm, corresponding to two vectors (t) and w(t) de ned in the SGD procedure of Section [15.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page212). That is, (t) will be

a vector such that

|  |  |  |
| --- | --- | --- |
|  | m |  |
| (t) = | j(t) (xj) | (16.6) |
|  | Xj |  |
|  | =1 |  |
| and (t) be such that |  |  |
|  | m |  |
| w(t) = | j(t) (xj): | (16.7) |
|  | Xj |  |
|  | =1 |  |

The vectors and are updated according to the following procedure.

SGD for Solving Soft-SVM with Kernels

Goal: Solve Equation ([16.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page223))

parameter: T

Initialize: (1) = 0

for t = 1; : : : ; T

Let (t) = 1t (t)

Choose i uniformly at random from [m]

For all j 6= i set j(t+1) = j(t)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| If (yi | m (t)K(xj; xi) < 1) | | | |  |  |  |  |
| Set | Pi | i |  | i |  |  |  |  |
|  | j=1 | j |  |  |  |  |  |  |
|  | (t+1) = (t) + y | | |  |  |  |  |  |
| Else |  |  |  |  |  |  |  |  |
| Set (t+1) | | = (t) |  |  |  |  |  |  |
|  | i | i |  |  |  |  |  |  |
|  |  | m |  |  | 1 |  | T |  |
| Output: w = Pj=1 | | | j | (xj) where = | P |  | (t) |
| T | t=1 |

The following lemma shows that the preceding implementation is equivalent to running the SGD procedure described in Section [15.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page212) on the feature space.

lemma 16.3 Let w^ be the output of the SGD procedure described in Sec-

Pm

tion [15.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page212), when applied on the feature space, and let w = j=1 j (xj) be the output of applying SGD with kernels. Then w = w^.

Proof We will show that for every t Equation ([16.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page223)) holds, where (t) is the result of running the SGD procedure described in Section [15.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page212) in the feature

1. Kernel Methods

space. By the de nition of (t) = 1t (t) and w(t) = 1t (t), this claim implies that Equation ([16.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page223)) also holds, and the proof of our lemma will follow. To prove that Equation ([16.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page223)) holds we use a simple inductive argument. For t = 1 the claim trivially holds. Assume it holds for t 1. Then,

\* +

m

1. E

|  |  |  |  |
| --- | --- | --- | --- |
| yi | w(t); (xi) = yi | j(t) (xj); (xi) = yi | j(t)K(xj; xi): |
|  | Xj |  | Xj |
|  |  |  | =1 |

Hence, the condition in the two algorithms is equivalent and if we update we have

|  |  |  |  |
| --- | --- | --- | --- |
|  | m |  | m |
| (t+1) = (t) + yi | (xi) =(t) | (xj) + yi (xi) = | (t+1) (xj); |
|  | X |  | Xj |
|  | j |  | j |
|  | j=1 |  | =1 |

which concludes our proof.

16.4 Summary

Mappings from the given domain to some higher dimensional space, on which a halfspace predictor is used, can be highly powerful. We bene t from a rich and complex hypothesis class, yet need to solve the problems of high sample and computational complexities. In Chapter [10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page130), we discussed the AdaBoost algo-rithm, which faces these challenges by using a weak learner: Even though we're in a very high dimensional space, we have an \oracle" that bestows on us a single good coordinate to work with on each iteration. In this chapter we intro-duced a di erent approach, the kernel trick. The idea is that in order to nd a halfspace predictor in the high dimensional space, we do not need to know the representation of instances in that space, but rather the values of inner products between the mapped instances. Calculating inner products between instances in the high dimensional space without using their representation in that space is done using kernel functions. We have also shown how the SGD algorithm can be implemented using kernels.

The ideas of feature mapping and the kernel trick allow us to use the framework of halfspaces and linear predictors for nonvectorial data. We demonstrated how kernels can be used to learn predictors over the domain of strings.

We presented the applicability of the kernel trick in SVM. However, the kernel trick can be applied in many other algorithms. A few examples are given as exercises.

This chapter ends the series of chapters on linear predictors and convex prob-lems. The next two chapters deal with completely di erent types of hypothesis classes.

|  |  |
| --- | --- |
| 16.5 Bibliographic Remarks | 225 |
|  |  |

16.5 Bibliographic Remarks

In the context of SVM, the kernel-trick has been introduced in Boser et al. (1992). See also Aizerman, Braverman & Rozonoer (1964). The observation that the kernel-trick can be applied whenever an algorithm only relies on inner products was rst stated by Sch•olkopf, Smola & M•uller (1998). The proof of the representer theorem is given in (Sch•olkopf, Herbrich, Smola & Williamson 2000, Sch•olkopf, Herbrich & Smola 2001). The conditions stated in Lemma [16.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page222) are simpli cation of conditions due to Mercer. Many useful kernel functions have been introduced in the literature for various applications. We refer the reader to Sch•olkopf & Smola (2002).

16.6 Exercises

1. Consider the task of nding a sequence of characters in a le, as described in Section [16.2.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page221). Show that every member of the class H can be realized by

composing a linear classi er over (x), whose norm is 1 and that attains a margin of 1.

1. Kernelized Perceptron: Show how to run the Perceptron algorithm while

only accessing the instances via the kernel function. Hint: The derivation is similar to the derivation of implementing SGD with kernels.

1. Kernel Ridge Regression: The ridge regression problem, with a feature

mapping , is the problem of nding a vector w that minimizes the function

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | | m |  |  |
|  |  | Xi |  |  |
| f(w) = kwk2 + 2m | | (16.8) | |
| (hw; (xi)i yi)2; |
|  |  | =1 |  |  |
| and then returning the predictor | |  |  |  |
| h(x) = hw; xi: | | |  |  |
| Show how to implement the ridge regression algorithm with kernels. | | |  |  |
| representer theorem tells us that there exists a vector | | | 2 R | m |
| Hint: The m | |  |  |
| such that Pi=1 i (xi) is a minimizer of Equation ([16.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page225)). | | |  |  |

1. Let G be the Gram matrix with regard to S and K. That is, Gij = K(xi; xj). De ne g : Rm ! R by

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | | m |  |
|  |  | Xi |  |
| g( ) =T G + 2m | | (16.9) |
| (h ; G ;ii yi)2; |
|  |  | =1 |  |

where G ;i is the i'th column of G. Show that if minimizes Equa-tion ([16.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page225)) then w = Pm i (xi) is a minimizer of f.

i=1

2. Find a closed form expression for .

1. Let N be any positive integer. For every x; x0 2 f1; : : : ; Ng de ne

K(x; x0) = minfx; x0g:

1. Kernel Methods

Prove that K is a valid kernel; namely, nd a mapping : f1; : : : ; Ng ! H where H is some Hilbert space, such that

8x; x0 2 f1; : : : ; Ng; K(x; x0) = h (x); (x0)i:

1. A supermarket manager would like to learn which of his customers have babies

on the basis of their shopping carts. Speci cally, he sampled i.i.d. customers, where for customer i, let xi f1; : : : ; dg denote the subset of items the customer bought, and let yi 2 f 1g be the label indicating whether this customer has a baby. As prior knowledge, the manager knows that there are k items such that the label is determined to be 1 i the customer bought at least one of these k items. Of course, the identity of these k items is not known (otherwise, there was nothing to learn). In addition, according to the store regulation, each customer can buy at most s items. Help the manager to

design a learning algorithm such that both its time complexity and its sample complexity are polynomial in s; k, and 1= .

6. Let X be an instance set and let be a feature mapping of X into some Hilbert feature space V . Let K : X X ! R be a kernel function that implements inner products in the feature space V .

Consider the binary classi cation algorithm that predicts the label of an unseen instance according to the class with the closest average. Formally, given a training sequence S = (x1; y1); : : : ; (xm; ym), for every y 2 f 1g we de ne

1 X

cy = my i:yi=y (xi):

where my = jfi : yi = ygj. We assume that m+ and m are nonzero. Then, the algorithm outputs the following decision rule:

(

h(x) =

1 k (x) c+k k (x) c k

* 1. otherwise:

1. Let w = c+ c and let b = 12 (kc k2 kc+k2). Show that

h(x) = sign(hw; (x)i + b):

2. Show how to express h(x) on the basis of the kernel function, and without accessing individual entries of (x) or w.

1. Multiclass, Ranking, and Complex Prediction Problems

Multiclass categorization is the problem of classifying instances into one of several possible target classes. That is, we are aiming at learning a predictor h : X ! Y, where Y is a nite set of categories. Applications include, for example, catego-rizing documents according to topic (X is the set of documents and Y is the set of possible topics) or determining which object appears in a given image (X is the set of images and Y is the set of possible objects).

The centrality of the multiclass learning problem has spurred the development of various approaches for tackling the task. Perhaps the most straightforward approach is a reduction from multiclass classi cation to binary classi cation. In Section [17.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page227) we discuss the most common two reductions as well as the main drawback of the reduction approach.

We then turn to describe a family of linear predictors for multiclass problems. Relying on the RLM and SGD frameworks from previous chapters, we describe several practical algorithms for multiclass prediction.

In Section [17.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page236) we show how to use the multiclass machinery for complex pre-diction problems in which Y can be extremely large but has some structure on it. This task is often called structured output learning. In particular, we demon-strate this approach for the task of recognizing handwritten words, in which Y is the set of all possible strings of some bounded length (hence, the size of Y is exponential in the maximal length of a word).

Finally, in Section [17.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page238) and Section [17.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page243) we discuss ranking problems in which the learner should order a set of instances according to their \relevance." A typ-ical application is ordering results of a search engine according to their relevance to the query. We describe several performance measures that are adequate for assessing the performance of ranking predictors and describe how to learn linear predictors for ranking problems e ciently.

17.1 One-versus-All and All-Pairs

The simplest approach to tackle multiclass prediction problems is by reduction to binary classi cation. Recall that in multiclass prediction we would like to learn

1. function h : X ! Y. Without loss of generality let us denote Y = f1; : : : ; kg. In the One-versus-All method (a.k.a. One-versus-Rest) we train k binary clas-

1. Multiclass, Ranking, and Complex Prediction Problems

si ers, each of which discriminates between one class and the rest of the classes. That is, given a training set S = (x1; y1); : : : ; (xm; ym), where every yi is in Y, we construct k binary training sets, S1; : : : ; Sk, where Si = (x1; ( 1)1[y16=i] ); : : : ; (xm; ( 1)1[ym6=i] ). In words, Si is the set of instances labeled 1 if their label in S was i, and 1 otherwise. For every i 2 [k] we train a binary predictor hi : X ! f 1g based on

Si, hoping that hi(x) should equal 1 if and only if x belongs to class i. Then, given h1; : : : ; hk, we construct a multiclass predictor using the rule

|  |  |
| --- | --- |
| h(x) 2 argmax hi(x): | (17.1) |
| i2[k] |  |

When more than one binary hypothesis predicts \1" we should somehow decide which class to predict (e.g., we can arbitrarily decide to break ties by taking the minimal index in argmaxi hi(x)). A better approach can be applied whenever each hi hides additional information, which can be interpreted as the con dence in the prediction y = i. For example, this is the case in halfspaces, where the actual prediction is sign(hw; xi), but we can interpret hw; xi as the con dence in the prediction. In such cases, we can apply the multiclass rule given in Equa-tion ([17.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page228)) on the real valued predictions. A pseudocode of the One-versus-All approach is given in the following.

One-versus-All

input:

training set S = (x1; y1); : : : ; (xm; ym)

algorithm for binary classi cation A

foreach i 2 Y

let Si = (x1; ( 1)1[y16=i] ); : : : ; (xm; ( 1)1[ym6=i] )

let hi = A(Si)

output:

the multiclass hypothesis de ned by h(x) 2 argmaxi2Y hi(x)

Another popular reduction is the All-Pairs approach, in which all pairs of

classes are compared to each other. Formally, given a training set S = (x1; y1); : : : ; (xm; ym), where every yi is in [k], for every 1 i < j k we construct a binary training sequence, Si;j, containing all examples from S whose label is either i or j. For

each such an example, we set the binary label in Si;j to be +1 if the multiclass label in S is i and 1 if the multiclass label in S is j. Next, we train a binary classi cation algorithm based on every Si;j to get hi;j. Finally, we construct a multiclass classi er by predicting the class that had the highest number of \wins." A pseudocode of the All-Pairs approach is given in the following.

|  |  |
| --- | --- |
| 17.1 One-versus-All and All-Pairs | 229 |
|  |  |

All-Pairs

input:

training set S = (x1; y1); : : : ; (xm; ym)

algorithm for binary classi cation A

foreach i; j 2 Y s.t. i < j

initialize Si;j to be the empty sequence

for t = 1; : : : ; m

If yt = i add (xt; 1) to Si;j

If yt = j add (xt; 1) to Si;j

let hi;j = A(Si;j)

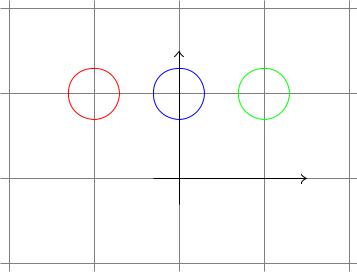
output:

the multiclass hypothesis de ned by

|  |  |  |
| --- | --- | --- |
| h(x) 2 argmaxi2Y |  | j2Y sign(j i) hi;j(x) |
|  |  | P |

Although reduction methods such as the One-versus-All and All-Pairs are simple and easy to construct from existing algorithms, their simplicity has a price. The binary learner is not aware of the fact that we are going to use its output hypotheses for constructing a multiclass predictor, and this might lead to suboptimal results, as illustrated in the following example.

Example 17.1 Consider a multiclass categorization problem in which the in-stance space is X = R2 and the label set is Y = f1; 2; 3g. Suppose that instances of the di erent classes are located in nonintersecting balls as depicted in the fol-lowing.



1 2 3

Suppose that the probability masses of classes 1; 2; 3 are 40%; 20%; and 40%, respectively. Consider the application of One-versus-All to this problem, and as-sume that the binary classi cation algorithm used by One-versus-All is ERM with respect to the hypothesis class of halfspaces. Observe that for the prob-lem of discriminating between class 2 and the rest of the classes, the optimal halfspace would be the all negative classi er. Therefore, the multiclass predic-tor constructed by One-versus-All might err on all the examples from class 2 (this will be the case if the tie in the de nition of h(x) is broken by the nu-merical value of the class label). In contrast, if we choose hi(x) = hwi; xi,

where w = 1 ; 1 , w = (0; 1), and w = 1 ; 1 , then the classi-

1 ~~p~~2 ~~p~~2 2 3 ~~p~~~~2~~~~p~~~~2~~

er de ned by h(x) = argmaxi hi(x) perfectly predicts all the examples. We see

1. Multiclass, Ranking, and Complex Prediction Problems

that even though the approximation error of the class of predictors of the form h(x) = argmaxihwi; xi is zero, the One-versus-All approach might fail to nd a good predictor from this class.

17.2 Linear Multiclass Predictors

In light of the inadequacy of reduction methods, in this section we study a more direct approach for learning multiclass predictors. We describe the family of linear multiclass predictors. To motivate the construction of this family, recall that a linear predictor for binary classi cation (i.e., a halfspace) takes the form

h(x) = sign(hw; xi):

An equivalent way to express the prediction is as follows:

h(x) = argmax hw; yxi;

y2f 1g

where yx is the vector obtained by multiplying each element of x by y.

This representation leads to a natural generalization of halfspaces to multiclass problems as follows. Let : X Y ! Rd be a class-sensitive feature mapping. That is, takes as input a pair (x; y) and maps it into a d dimensional feature vector. Intuitively, we can think of the elements of (x; y) as score functions that assess how well the label y ts the instance x. We will elaborate on later on. Given and a vector w 2 Rd, we can de ne a multiclass predictor, h : X ! Y, as follows:

h(x) = argmax hw; (x; y)i:

y2Y

That is, the prediction of h for the input x is the label that achieves the highest weighted score, where weighting is according to the vector w.

Let W be some set of vectors in Rd, for example, W = fw 2 Rd : kwk Bg, for some scalar B > 0. Each pair ( ; W ) de nes a hypothesis class of multiclass predictors:

H ;W = fx 7!argmax hw; (x; y)i : w 2 W g:

y2Y

Of course, the immediate question, which we discuss in the sequel, is how to construct a good . Note that if Y = f 1g and we set (x; y) = yx and

1. = Rd, then H ;W becomes the hypothesis class of homogeneous halfspace predictors for binary classi cation.

17.2.1 How to Construct

As mentioned before, we can think of the elements of (x; y) as score functions that assess how well the label y ts the instance x. Naturally, designing a good is similar to the problem of designing a good feature mapping (as we discussed in

|  |  |
| --- | --- |
| 17.2 Linear Multiclass Predictors | 231 |
|  |  |

Chapter [16](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page215) and as we will discuss in more detail in Chapter [25](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page357)). Two examples of useful constructions are given in the following.

The Multivector Construction:

Let Y = f1; : : : ; kg and let X = Rn. We de ne : X Y ! Rd, where d = nk, as follows

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (x; y) = [ 0; : : : ; 0 | | | | | | ; x1; : : : ; xn ; | | | | | | 0; : : : ; 0 | | | | | | ]: | (17.2) |
| | |  | | {z |  | } | | |  | {z | |  | } | | |  | | {z |  | } |  |  |
| 2R | | (y 1)n | | | | 2 | | | Rn | | | 2R | | (k y)n | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

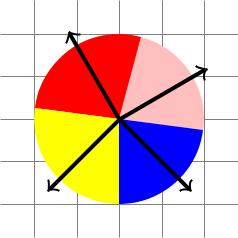
That is, (x; y) is composed of k vectors, each of which is of dimension n, where we set all the vectors to be the all zeros vector except the y'th vector, which is set to be x. It follows that we can think of w 2 Rnk as being composed of k weight vectors in Rn, that is, w = [w1; : : : ; wk], hence the name multivec-tor construction. By the construction we have that hw; (x; y)i = hwy; xi, and therefore the multiclass prediction becomes

h(x) = argmax hwy; xi:

y2Y

A geometric illustration of the multiclass prediction over X = R2 is given in the following.

w2



w1

w3 w4

TF-IDF:

The previous de nition of (x; y) does not incorporate any prior knowledge about the problem. We next describe an example of a feature function that does incorporate prior knowledge. Let X be a set of text documents and Y be a set of possible topics. Let d be a size of a dictionary of words. For each word in the dictionary, whose corresponding index is j, let T F (j; x) be the number of times the word corresponding to j appears in the document x. This quantity is called Term-Frequency. Additionally, let DF (j; y) be the number of times the word corresponding to j appears in documents in our training set that are not about topic y. This quantity is called Document-Frequency and measures whether word j is frequent in other topics. Now, de ne : X Y ! Rd to be such that

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j(x; y) = T F (j; x) log | DF (j;y) | | | ; |
|  |  | m |  |  |

where m is the total number of documents in our training set. The preced-ing quantity is called term-frequency-inverse-document-frequency or TF-IDF for

1. Multiclass, Ranking, and Complex Prediction Problems

short. Intuitively, j(x; y) should be large if the word corresponding to j ap-pears a lot in the document x but does not appear at all in documents that are not on topic y. If this is the case, we tend to believe that the document x is on topic y. Note that unlike the multivector construction described previously, in the current construction the dimension of does not depend on the number of topics (i.e., the size of Y).

17.2.2 Cost-Sensitive Classi cation

So far we used the zero-one loss as our performance measure of the quality of h(x). That is, the loss of a hypothesis h on an example (x; y) is 1 if h(x) 6= y and 0 otherwise. In some situations it makes more sense to penalize di erent levels of loss for di erent mistakes. For example, in object recognition tasks, it is less severe to predict that an image of a tiger contains a cat than predicting that the image contains a whale. This can be modeled by specifying a loss function,

: Y Y ! R+, where for every pair of labels, y0; y, the loss of predicting the label y0 when the correct label is y is de ned to be (y0; y). We assume

that (y; y) = 0. Note that the zero-one loss can be easily modeled by setting (y0; y) = 1[y06=y].

17.2.3 ERM

We have de ned the hypothesis class H ;W and speci ed a loss function . To learn the class with respect to the loss function, we can apply the ERM rule with respect to this class. That is, we search for a multiclass hypothesis h 2 H ;W , parameterized by a vector w, that minimizes the empirical risk with respect to ,

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | |  | m |
|  |  |  | Xi |
| LS(h) = m | | |
| (h(xi); yi): |
|  |  |  | =1 |

We now show that when W = Rd and we are in the realizable case, then it is possible to solve the ERM problem e ciently using linear programming. Indeed, in the realizable case, we need to nd a vector w 2 Rd that satis es

8i 2 [m]; yi = argmaxhw; (xi; y)i:

y2Y

Equivalently, we need that w will satisfy the following set of linear inequalities

8i 2 [m]; 8y 2 Y n fyig; hw; (xi; yi)i > hw; (xi; y)i:

Finding w that satis es the preceding set of linear equations amounts to solving a linear program.

As in the case of binary classi cation, it is also possible to use a generalization of the Perceptron algorithm for solving the ERM problem. See Exercise [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page248).

In the nonrealizable case, solving the ERM problem is in general computa-tionally hard. We tackle this di culty using the method of convex surrogate

|  |  |
| --- | --- |
| 17.2 Linear Multiclass Predictors | 233 |
|  |  |

loss functions (see Section [12.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page167)). In particular, we generalize the hinge loss to multiclass problems.

17.2.4 Generalized Hinge Loss

Recall that in binary classi cation, the hinge loss is de ned to be maxf0; 1 yhw; xig. We now generalize the hinge loss to multiclass predictors of the form

hw(x) = argmax hw; (x; y0)i:

y02Y

Recall that a surrogate convex loss should upper bound the original nonconvex loss, which in our case is (hw(x); y). To derive an upper bound on (hw(x); y) we rst note that the de nition of hw(x) implies that

hw; (x; y)i hw; (x; hw(x))i:

Therefore,

(hw(x); y) (hw(x); y) + hw; (x; hw(x)) (x; y)i:

Since hw(x) 2 Y we can upper bound the right-hand side of the preceding by

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| max ( (y0; y) + | h | w; (x; y0) |  | (x; y) | def | (17.3) |
| ) = `(w; (x; y)): |
| y02Y |  | i |  |  |

We use the term \generalized hinge loss" to denote the preceding expression. As we have shown, `(w; (x; y)) (hw(x); y). Furthermore, equality holds when-ever the score of the correct label is larger than the score of any other label, y0, by at least (y0; y), namely,

8y0 2 Y n fyg; hw; (x; y)i hw; (x; y0)i + (y0; y):

It is also immediate to see that `(w; (x; y)) is a convex function with respect to w since it is a maximum over linear functions of w (see Claim [12.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page160) in Chapter [12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page156)), and that `(w; (x; y)) is -Lipschitz with = maxy02Y k (x; y0) (x; y)k.

Remark 17.2 We use the name \generalized hinge loss" since in the binary case, when Y = f 1g, if we set (x; y) = y2x , then the generalized hinge loss becomes the vanilla hinge loss for binary classi cation,

`(w; (x; y)) = maxf0; 1 yhw; xig:

Geometric Intuition:

The feature function : X Y ! Rd maps each x into jYj vectors in Rd. The value of `(w; (x; y)) will be zero if there exists a direction w such that when projecting the jYj vectors onto this direction we obtain that each vector is represented by the scalar hw; (x; y)i, and we can rank the di erent points on the basis of these scalars so that

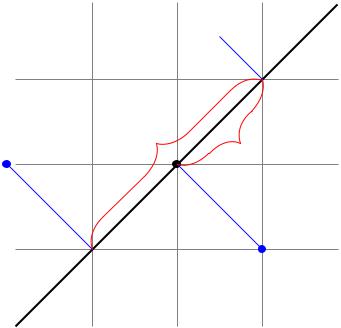
The point corresponding to the correct y is top-ranked

1. Multiclass, Ranking, and Complex Prediction Problems

For each y0 6= y, the di erence between hw; (x; y)i and hw; (x; y0)i is larger than the loss of predicting y0 instead of y. The di erence hw; (x; y)i hw; (x; y0)i is also referred to as the \margin" (see Section [15.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page202)).

This is illustrated in the following gure:

 w



(x; y) 

(x; y00)

|  |  |
| --- | --- |
| ( | y; |
|  |

1. )

y

|  |  |
| --- | --- |
|  | ( |
|  |
|  | y; |
|  | y0 |
|  | ) |
|  | (x; y0) |

17.2.5 Multiclass SVM and SGD

Once we have de ned the generalized hinge loss, we obtain a convex-Lipschitz learning problem and we can apply our general techniques for solving such prob-lems. In particular, the RLM technique we have studied in Chapter [13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page171) yields the multiclass SVM rule:

Multiclass SVM

input: (x1; y1); : : : ; (xm; ym)

parameters:

regularization parameter > 0

loss function : Y Y ! R+

class-sensitive feature mapping : X Y ! Rd

solve:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| w Rd | kwk + m | | | =1 y02Y | 0; yi) + hw; (xi; y0) (xi; yi)i)! |
| 2 |  | 1 |  | m |  |
| 2 |  |  | Xi |  |
| min |  |  | max ( (y |  |
|  |  |  |  |
| output the predictor hw(x) = argmaxy2Yhw; (x; y)i | | | | | |

We can solve the optimization problem associated with multiclass SVM us-ing generic convex optimization algorithms (or using the method described in Section [15.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page212)). Let us analyze the risk of the resulting hypothesis. The analysis seamlessly follows from our general analysis for convex-Lipschitz problems given in Chapter [13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page171). In particular, applying Corollary [13.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page179) and using the fact that the generalized hinge loss upper bounds the loss, we immediately obtain an analog of Corollary [15.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page208):

corollary 17.1 Let D be a distribution over X Y, let : X Y ! Rd, and assume that for all x 2 X and y 2 Y we have k (x; y)k =2. Let B > 0.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 17.2 Linear Multiclass Predictors | | | | | | | | 235 | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Consider running Multiclass SVM with = | | |  |  | 2 2 |  | on a training set S D | | | | | | m |
|  |  | B2m | |  |
|  |  |  | Then, | | | |  |  |  |  |  |  |  |
| and let hw be the output of Multiclass SVM. q | | | | | | |  | r |  |  |  |  |  |
| S Dm D | S Dm D | u:kuk B D | | | | | |  | m | | |  |
| E [L (hw)] | E [Lg hinge(w)] |  |  |  |  |  | Lg hinge |  | | 8 2B2 | | |  |
|  | min | | | | (u) + |  |  | ; | |  |
|  |  |  |

where LD(h) = E(x;y) D[ (h(x); y)] and LgD hinge(w) = E(x;y) D[`(w; (x; y))] with ` being the generalized hinge-loss as de ned in Equation ([17.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page233)).

We can also apply the SGD learning framework for minimizing LgD hinge(w) as described in Chapter [14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page184). Recall Claim [14.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page189), which dealt with subgradients of max functions. In light of this claim, in order to nd a subgradient of the generalized hinge loss all we need to do is to nd y 2 Y that achieves the maximum in the de nition of the generalized hinge loss. This yields the following algorithm:

SGD for Multiclass Learning

parameters:

Scalar > 0, integer T > 0

loss function : Y Y ! R+

class-sensitive feature mapping : X Y ! Rd

initialize: w(1) = 0 2 Rd

for t = 1; 2; : : : ; T

sample (x; y) D

nd y^ 2 argmaxy02Y (y0; y) + hw(t); (x; y0) (x; y)i

set vt = (x; y^) (x; y)

update w(t+1) = w(t) vt

output w = 1 PT w(t)

T t=1

Our general analysis of SGD given in Corollary [14.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page197) immediately implies:

corollary 17.2 Let D be a distribution over X Y, let : X Y ! Rd, and assume that for all x 2 X and y 2 Y we have k (x; y)k =2. Let B > 0. Then, for every > 0, if we run SGD for multiclass learning with a number of iterations (i.e., number of examples)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  | T |  | B2 2 | |  |  |  |  |  |  |  |
| and with = q | | | |  |  |  |  |  |  |  |  |  | 2 | |  |  |  |  |  |  |
|  | B2 | , then the output of SGD satis es | | | | | | | | | | | | |  |  |  |
|  | 2 T |  |  |  |
| S | E |  | (hw)] | | | | |  | S | E | g hinge | | |  | |  | min | | B | g hinge | (u) + : |
| m[LD | m[LD |  |  | (w)] | | u: u | k | LD |
|  | D |  |  |  |  |  |  |  |  | D |  |  |  |  |  |  | k |  |  |  |

Remark 17.3 It is interesting to note that the risk bounds given in Corol-lary [17.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page234) and Corollary [17.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page235) do not depend explicitly on the size of the label set Y, a fact we will rely on in the next section. However, the bounds may de-pend implicitly on the size of Y via the norm of (x; y) and the fact that the

bounds are meaningful only when there exists some vector u, kuk B, for which LgD hinge(u) is not excessively large.

1. Multiclass, Ranking, and Complex Prediction Problems

17.3 Structured Output Prediction

Structured output prediction problems are multiclass problems in which Y is very large but is endowed with a prede ned structure. The structure plays a key role in constructing e cient algorithms. To motivate structured learning problems, consider the problem of optical character recognition (OCR). Suppose we receive an image of some handwritten word and would like to predict which word is written in the image. To simplify the setting, suppose we know how to segment the image into a sequence of images, each of which contains a patch of the image corresponding to a single letter. Therefore, X is the set of sequences of images and Y is the set of sequences of letters. Note that the size of Y grows exponentially with the maximal length of a word. An example of an image x corresponding to the label y = \workable" is given in the following.



To tackle structure prediction we can rely on the family of linear predictors described in the previous section. In particular, we need to de ne a reasonable loss function for the problem, , as well as a good class-sensitive feature mapping, . By \good" we mean a feature mapping that will lead to a low approximation error for the class of linear predictors with respect to and . Once we do this, we can rely, for example, on the SGD learning algorithm de ned in the previous section.

However, the huge size of Y poses several challenges:

1. To apply the multiclass prediction we need to solve a maximization problem over Y. How can we predict e ciently when Y is so large?
2. How do we train w e ciently? In particular, to apply the SGD rule we again need to solve a maximization problem over Y.
3. How can we avoid over tting?

In the previous section we have already shown that the sample complexity of learning a linear multiclass predictor does not depend explicitly on the number of classes. We just need to make sure that the norm of the range of is not too large. This will take care of the over tting problem. To tackle the computational challenges we rely on the structure of the problem, and de ne the functions and

so that calculating the maximization problems in the de nition of hw and in the SGD algorithm can be performed e ciently. In the following we demonstrate one way to achieve these goals for the OCR task mentioned previously.

To simplify the presentation, let us assume that all the words in Y are of length r and that the number of di erent letters in our alphabet is q. Let y and y0 be two

|  |  |
| --- | --- |
| 17.3 Structured Output Prediction | 237 |
|  |  |

words (i.e., sequences of letters) in Y. We de ne the function (y0; y) to be the

average number of letters that are di erent in y0 and y, namely, 1 Pr 1[y 6=y0].

r i=1 i i

Next, let us de ne a class-sensitive feature mapping (x; y). It will be conve-nient to think about x as a matrix of size n r, where n is the number of pixels in each image, and r is the number of images in the sequence. The j'th column of x corresponds to the j'th image in the sequence (encoded as a vector of gray level values of pixels). The dimension of the range of is set to be d = n q + q2.

The rst nq feature functions are \type 1" features and take the form:

r

1 X

i;j;1(x; y) = r t=1 xi;t 1[yt=j]:

That is, we sum the value of the i'th pixel only over the images for which y assigns the letter j. The triple index (i; j; 1) indicates that we are dealing with feature (i; j) of type 1. Intuitively, such features can capture pixels in the image whose gray level values are indicative of a certain letter. The second type of features take the form

r

1 X

i;j;2(x; y) = r t=2 1[yt=i] 1[yt 1=j]:

That is, we sum the number of times the letter i follows the letter j. Intuitively, these features can capture rules like \It is likely to see the pair `qu' in a word" or \It is unlikely to see the pair `rz' in a word." Of course, some of these features will not be very useful, so the goal of the learning process is to assign weights to features by learning the vector w, so that the weighted score will give us a good prediction via

hw(x) = argmax hw; (x; y)i:

y2Y

It is left to show how to solve the optimization problem in the de nition of hw(x) e ciently, as well as how to solve the optimization problem in the de nition of y^ in the SGD algorithm. We can do this by applying a dynamic programming procedure. We describe the procedure for solving the maximization in the de nition of hw and leave as an exercise the maximization problem in the de nition of y^ in the SGD algorithm.

To derive the dynamic programming procedure, let us rst observe that we can write

r

X

(x; y) = (x; yt; yt 1);

t=1

for an appropriate : X [q] [q] [ f0g ! Rd, and for simplicity we assume that y0 is always equal to 0. Indeed, each feature function i;j;1 can be written in terms of

i;j;1(x; yt; yt 1) = xi;t 1[yt=j];

1. Multiclass, Ranking, and Complex Prediction Problems

while the feature function i;j;2 can be written in terms of

i;j;2(x; yt; yt 1) = 1[yt=i] 1[yt 1=j]:

Therefore, the prediction can be written as

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 2Y | r |  |  |  |  |  |  |
|  |  | Xt | w; (x; y | ; y |  | ) | : | (17.4) |
| hw(x) = | argmax | | h | t 1 |
| y |  | t |  | i |  |  |
|  |  |  | =1 |  |  |  |  |  |  |

In the following we derive a dynamic programming procedure that solves every problem of the form given in Equation ([17.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page238)). The procedure will maintain a matrix M 2 Rq;r such that

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | max | Xt |
| Ms; = | hw; (x; yt; yt 1)i: |
| (y1;:::;y ):y =s |
|  |  | =1 |

Clearly, the maximum of hw; (x; y)i equals maxs Ms;r. Furthermore, we can calculate M in a recursive manner:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| M |  | = max (M | s0; 1 | + | h | w; (x; s; s0) ) : | (17.5) |
|  | s; | s0 |  | i |  |

This yields the following procedure:

Dynamic Programming for Calculating hw(x) as Given

in Equation ([17.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page238))

input: a matrix x 2 Rn;r and a vector w

initialize:

foreach s 2 [q]

Ms;1 = hw; (x; s; 1)i

for = 2; : : : ; r

foreach s 2 [q]

set Ms; as in Equation ([17.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page238))

set Is; to be the s0 that maximizes Equation ([17.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page238))

set yt = argmaxs Ms;r

for = r; r 1; : : : ; 2

set y 1 = Iy ;

output: y = (y1; : : : ; yr)

17.4 Ranking

Ranking is the problem of ordering a set of instances according to their \rele-vance." A typical application is ordering results of a search engine according to their relevance to the query. Another example is a system that monitors elec-tronic transactions and should alert for possible fraudulent transactions. Such a system should order transactions according to how suspicious they are.

Formally, let X = S1n=1 X n be the set of all sequences of instances from

|  |  |
| --- | --- |
| 17.4 Ranking | 239 |
|  |  |

1. of arbitrary length. A ranking hypothesis, h, is a function that receives a sequence of instances x = (x1; : : : ; xr) 2 X , and returns a permutation of [r]. It is more convenient to let the output of h be a vector y 2 Rr, where by sorting the elements of y we obtain the permutation over [r]. We denote by (y) the permutation over [r] induced by y. For example, for r = 5, the vector

y = (2; 1; 6; 1; 0:5) induces the permutation (y) = (4; 3; 5; 1; 2). That is,

if we sort y in an ascending order, then we obtain the vector ( 1; 0:5; 1; 2; 6).

Now, (y)i is the position of yi in the sorted vector ( 1; 0:5; 1; 2; 6). This notation re ects that the top-ranked instances are those that achieve the highest values in (y).

In the notation of our PAC learning model, the examples domain is Z = S1 (X r Rr), and the hypothesis class, H, is some set of ranking hypotheses. r=1

to de ne such loss functions, and here we list a few examples. In all the examples we de ne `(h; (x; y)) = (h(x); y), for some function : S1r=1(Rr Rr) ! R+.

0{1 Ranking loss: (y0; y) is zero if y and y0 induce exactly the same

ranking and (y0; y) = 1 otherwise. That is, (y0; y) = 1[ (y0)6= (y)]. Such a loss function is almost never used in practice as it does not distinguish between the case in which (y0) is almost equal to (y) and the case in which (y0) is completely di erent from (y).

Kendall-Tau Loss: We count the number of pairs (i; j) that are in di erent order in the two permutations. This can be written as

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 2 | r 1 | r |  |  |  |  |
| (y0 |  |  | Xi | X |  |  |  |  |
| ; y) = |  |  | 1[sign(y0 |  | y0 )=sign(yi |  | yj)]: |
|  |  |
|  | r(r 1) | |  | i | j 6 |  |

=1 j=i+1

This loss function is more useful than the 0{1 loss as it re ects the level of similarity between the two rankings.

Normalized Discounted Cumulative Gain (NDCG): This measure em-phasizes the correctness at the top of the list by using a monotonically nondecreasing discount function D : N ! R+. We rst de ne a discounted cumulative gain measure:

r

X

G(y0; y) = D( (y0)i) yi:

i=1

In words, if we interpret yi as a score of the \true relevance" of item i, then we take a weighted sum of the relevance of the elements, while the weight of yi is determined on the basis of the position of i in (y0). Assuming that all elements of y are nonnegative, it is easy to verify that 0 G(y0; y) G(y; y). We can therefore de ne a normalized discounted cumulative gain by the ratio G(y0; y)=G(y; y), and the corresponding loss function would be

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| G(y0; y) | | | 1 | r |  |
| Xi |  |
|  |  |  |  |  |
| (y0; y) = 1 G(y; y) = G(y; y) | | | | (D( (y)i) D( (y0)i)) yi: |
| =1 |
|  |  |  |  |  |

1. Multiclass, Ranking, and Complex Prediction Problems

We can easily see that (y0; y) 2 [0; 1] and that (y0; y) = 0 whenever (y0) = (y).

A typical way to de ne the discount function is by

(

1 if i 2 fr k + 1; : : : ; rg

D(i) = log2(r i+2)

1. otherwise

where k < r is a parameter. This means that we care more about elements that are ranked higher, and we completely ignore elements that are not at the top-k ranked elements. The NDCG measure is often used to evaluate the performance of search engines since in such applications it makes sense completely to ignore elements that are not at the top of the ranking.

Once we have a hypothesis class and a ranking loss function, we can learn a ranking function using the ERM rule. However, from the computational point of view, the resulting optimization problem might be hard to solve. We next discuss how to learn linear predictors for ranking.

17.4.1 Linear Predictors for Ranking

A natural way to de ne a ranking function is by projecting the instances onto some vector w and then outputting the resulting scalars as our representation of the ranking function. That is, assuming that X Rd, for every w 2 Rd we de ne a ranking function

hw((x1; : : : ; xr)) = (hw; x1i; : : : ; hw; xri): (17.6)

As we discussed in Chapter [16](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page215), we can also apply a feature mapping that maps instances into some feature space and then takes the inner products with w in the feature space. For simplicity, we focus on the simpler form as in Equation ([17.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page240)).

Given some W Rd, we can now de ne the hypothesis class HW = fhw : w 2 W g. Once we have de ned this hypothesis class, and have chosen a ranking loss function, we can apply the ERM rule as follows: Given a training set, S = (x1; y1); : : : ; (xm; ym), where each (xi; yi) is in (X R)ri , for some ri 2 N, we

Pm

should search w 2 W that minimizes the empirical loss, i=1 (hw(xi); yi). As in the case of binary classi cation, for many loss functions this problem is computationally hard, and we therefore turn to describe convex surrogate loss functions. We describe the surrogates for the Kendall tau loss and for the NDCG loss.

A Hinge Loss for the Kendall Tau Loss Function:

We can think of the Kendall tau loss as an average of 0 1 losses for each pair. In particular, for every (i; j) we can rewrite

1[sign(yi0 yj0)6=sign(yi yj)] = 1[sign(yi yj)(yi0 yj0) 0]:

|  |  |
| --- | --- |
| 17.4 Ranking | 241 |
|  |  |

In our case, yi0 yj0 = hw; xi xji. It follows that we can use the hinge loss upper bound as follows:

1[sign(yi yj)(yi0 yj0) 0] max f0; 1 sign (yi yj) hw; xi xjig :

Taking the average over the pairs we obtain the following surrogate convex loss for the Kendall tau loss function:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | 2 |  | r 1 | r |
|  |  |  |  | Xi | X |
| (hw(x); y) |  |  |  |  | max f0; 1 sign(yi yj) hw; xi xjig : |
| r(r |  | 1) |  |

=1 j=i+1

The right-hand side is convex with respect to w and upper bounds the Kendall tau loss. It is also a -Lipschitz function with parameter maxi;j kxi xjk.

A Hinge Loss for the NDCG Loss Function:

The NDCG loss function depends on the predicted ranking vector y0 2 Rr via the permutation it induces. To derive a surrogate loss function we rst make the following observation. Let V be the set of all permutations of [r] encoded as vectors; namely, each v 2 V is a vector in [r]r such that for all i 6= j we have vi 6= vj. Then (see Exercise [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page249)),

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 2 | r |  |  |  |  |  |
|  |  |  |  |  | Xi | |  | yi0: |  |  |
|  | (y0) = argmax | | | | | | vi | |  | (17.7) |
|  |  |  |  | v V | | =1 | |  |  |  |  |
|  | = P |  |  |  |  |  |  |  |  |
| Let us denote (x; v) | r |  |  |  | r |  |  |  |  |  |
| i=1 vixi; it | |  |  |  |  |  |  |  |
|  |  |  | follows that | | | | |  |  |  |
|  |  |  |  | 2 |  | Xi | ih | w; x | | ii | |
|  |  | argmax | | | v |
|  | (hw(x)) = | | v V | | | =1 |  |  |
|  |  | = | v V | | | \* | r | | i |  | i+ |
|  |  |  |  | 2 |  | w; | Xi | | |  |  |
|  |  |  | argmax | | |  |  | v x | |  |
|  |  |  |  |  |  |  | =1 | | |  |  |

= argmaxhw; (x; v)i:

v2V

On the basis of this observation, we can use the generalized hinge loss for cost-sensitive multiclass classi cation as a surrogate loss function for the NDCG loss as follows:

(hw(x); y) (hw(x); y) + hw; (x; (hw(x)))i hw; (x; (y))i

max [ (v; y) + hw; (x; v)i hw; (x; (y))i]

v2V

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| = v V | " |  | (vi (y)i) hw; xii# | : | (17.8) |
|  |  |  | r |  |  |
| max |  | (v; y) + | Xi |  |  |
| 2 |  |  | =1 |  |  |

The right-hand side is a convex function with respect to w.

We can now solve the learning problem using SGD as described in Section [17.2.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page234). The main computational bottleneck is calculating a subgradient of the loss func-tion, which is equivalent to nding v that achieves the maximum in Equa-tion ([17.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page241)) (see Claim [14.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page189)). Using the de nition of the NDCG loss, this is

1. Multiclass, Ranking, and Complex Prediction Problems

equivalent to solving the problem

|  |  |
| --- | --- |
| 2 | r |
| Xi |
| argmin | ( ivi + i D(vi)); |
| v V | =1 |
|  |

where i = hw; xii and i = yi=G(y; y). We can think of this problem a little bit di erently by de ning a matrix A 2 Rr;r where

Ai;j = j i + D(j) i:

Now, let us think about each j as a \worker," each i as a \task," and Ai;j as the cost of assigning task i to worker j. With this view, the problem of nding v becomes the problem of nding an assignment of the tasks to workers of minimal cost. This problem is called \the assignment problem" and can be solved e ciently. One particular algorithm is the \Hungarian method" (Kuhn 1955). Another way to solve the assignment problem is using linear programming. To do so, let us rst write the assignment problem as

|  |  |  |
| --- | --- | --- |
| 2 | r |  |
| X |  |
| argmin | Ai;jBi;j | (17.9) |
| B R+r;r | i;j=1 |  |
|  | r |  |
|  | Xj | Bi;j = 1 |
| s.t. | 8i 2 [r]; |
|  | =1 |  |
|  | r |  |
|  | Xi | Bi;j = 1 |
|  | 8j 2 [r]; |
|  | =1 |  |

8i; j; Bi;j 2 f0; 1g

A matrix B that satis es the constraints in the preceding optimization problem is called a permutation matrix. This is because the constraints guarantee that there is at most a single entry of each row that equals 1 and a single entry of each column that equals 1. Therefore, the matrix B corresponds to the permutation v 2 V de ned by vi = j for the single index j that satis es Bi;j = 1.

The preceding optimization is still not a linear program because of the com-binatorial constraint Bi;j 2 f0; 1g. However, as it turns out, this constraint is redundant { if we solve the optimization problem while simply omitting the combinatorial constraint, then we are still guaranteed that there is an optimal solution that will satisfy this constraint. This is formalized later.

P

Denote hA; Bi = i;j Ai;jBi;j. Then, Equation ([17.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page242)) is the problem of mini-mizing hA; Bi such that B is a permutation matrix.

A matrix B 2 Rr;r is called doubly stochastic if all elements of B are non-negative, the sum of each row of B is 1, and the sum of each column of B is 1. Therefore, solving Equation ([17.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page242)) without the constraints Bi;j 2 f0; 1g is the

|  |  |
| --- | --- |
| problem |  |
| argminhA; Bi s.t. B is a doubly stochastic matrix: | (17.10) |

B2Rr;r

|  |  |
| --- | --- |
| 17.5 Bipartite Ranking and Multivariate Performance Measures | 243 |
|  |  |

The following claim states that every doubly stochastic matrix is a convex combination of permutation matrices.

claim 17.3 ((Birkho 1946, Von Neumann 1953)) The set of doubly stochastic matrices in Rr;r is the convex hull of the set of permutation matrices in Rr;r.

On the basis of the claim, we easily obtain the following:

lemma 17.4 There exists an optimal solution of Equation ([17.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page242)) that is also an optimal solution of Equation ([17.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page242)).

Proof Let B be a solution of Equation ([17.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page242)). Then, by Claim [17.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page243), we can

P

write B = i iCi, where each Ci is a permutation matrix, each i > 0, and

P

i i = 1. Since all the Ci are also doubly stochastic, we clearly have that hA; Bi hA; Cii for every i. We claim that there is some i for which hA; B i = hA; Cii. This must be true since otherwise, if for every i hA; Bi < hA; Cii, we would have that

1. +

X X X

hA; Bi = A; iCi = ihA; Cii > ihA; Bi = hA; Bi;

i i i

which cannot hold. We have thus shown that some permutation matrix, Ci, satis es hA; Bi = hA; Cii. But, since for every other permutation matrix C we have hA; B i hA; Ci we conclude that Ci is an optimal solution of both Equa-tion ([17.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page242)) and Equation ([17.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page242)). 

17.5 Bipartite Ranking and Multivariate Performance Measures

In the previous section we described the problem of ranking. We used a vector y 2 Rr for representing an order over the elements x1; : : : ; xr. If all elements in y are di erent from each other, then y speci es a full order over [r]. However, if two elements of y attain the same value, yi = yj for i 6= j, then y can only specify a partial order over [r]. In such a case, we say that xi and xj are of equal relevance according to y. In the extreme case, y 2 f 1gr, which means that each xi is either relevant or nonrelevant. This setting is often called \bipartite ranking." For example, in the fraud detection application mentioned in the previous section, each transaction is labeled as either fraudulent (yi = 1) or benign (yi = 1).

Seemingly, we can solve the bipartite ranking problem by learning a binary classi er, applying it on each instance, and putting the positive ones at the top of the ranked list. However, this may lead to poor results as the goal of a binary learner is usually to minimize the zero-one loss (or some surrogate of it), while the goal of a ranker might be signi cantly di erent. To illustrate this, consider again the problem of fraud detection. Usually, most of the transactions are benign (say 99:9%). Therefore, a binary classi er that predicts \benign" on all transactions will have a zero-one error of 0:1%. While this is a very small number, the resulting predictor is meaningless for the fraud detection application. The crux of the

1. Multiclass, Ranking, and Complex Prediction Problems

problem stems from the inadequacy of the zero-one loss for what we are really interested in. A more adequate performance measure should take into account the predictions over the entire set of instances. For example, in the previous section we have de ned the NDCG loss, which emphasizes the correctness of the top-ranked items. In this section we describe additional loss functions that are

speci cally adequate for bipartite ranking problems.

As in the previous section, we are given a sequence of instances, x = (x1; : : : ; xr), and we predict a ranking vector y0 2 Rr. The feedback vector is y 2 f 1gr. We de ne a loss that depends on y0 and y and depends on a threshold 2 R. This threshold transforms the vector y0 2 Rr into the vector (sign(yi0 ); : : : ; sign(yr0 )) 2 f 1gr. Usually, the value of is set to be 0. However, as we will see, we sometimes set while taking into account additional constraints on the problem.

The loss functions we de ne in the following depend on the following 4 num-bers:

True positives: a = jfi : yi = +1 ^ sign(yi0 ) = +1gj

False positives: b = jfi : yi = 1 ^ sign(yi0 ) = +1gj

(17.11)

False negatives: c = jfi : yi = +1 ^ sign(yi0 ) = 1gj

True negatives: d = jfi : yi = 1 ^ sign(yi0 ) = 1gj

The recall (a.k.a. sensitivity) of a prediction vector is the fraction of true positives y0 \catches," namely, a+ac . The precision is the fraction of correct predictions among the positive labels we predict, namely, a+ab . The speci city is the fraction of true negatives that our predictor \catches," namely, d+db .

Note that as we decrease the recall increases (attaining the value 1 when

= 1). On the other hand, the precision and the speci city usually decrease as we decrease . Therefore, there is a tradeo between precision and recall, and we can control it by changing . The loss functions de ned in the following use various techniques for combining both the precision and recall.

Averaging sensitivity and speci city: This measure is the average of the

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| sensitivity and speci city, namely, | 1 |  | a |  | + | d | . This is also the accuracy |
|  | a+c | | d+b |
| 2 | |  |  |  |

on positive examples averaged with the accuracy on negative examples. Here, we set = 0 and the corresponding loss function is (y0; y) =

1 1 a + d .

2 a+c d+b

F1-score: The F1 score is the harmonic mean of the precision and recall:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 2 | | |  |  | . Its maximal value (of 1) is obtained when both precision |
|  | 1 |  |  | + | 1 |  |
| Precision | | | Recall | | |
|  |  |

and recall are 1, and its minimal value (of 0) is obtained whenever one of them is 0 (even if the other one is 1). The F1 score can be written using

the numbers a; b; c as follows; F1 = 2a . Again, we set = 0, and the

2a+b+c

loss function becomes (y0; y) = 1 F1.

F -score: It is like F1 score, but we attach 2 times more importance to

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| recall than to precision, that is, |  |  | 1+ 2 | |  |  | . It can also be written as |
|  | 1 |  | + 2 | 1 |  |
|  | Precision | | | Recall | | |
|  |  |

|  |  |
| --- | --- |
| 17.5 Bipartite Ranking and Multivariate Performance Measures | 245 |
|  |  |

(1+ 2)a

F = (1+ 2)a+b+ 2c . Again, we set = 0, and the loss function becomes (y0; y) = 1 F .

Recall at k: We measure the recall while the prediction must contain at most k positive labels. That is, we should set so that a + b k. This is conve-nient, for example, in the application of a fraud detection system, where a bank employee can only handle a small number of suspicious transactions.

Precision at k: We measure the precision while the prediction must contain at least k positive labels. That is, we should set so that a + b k.

The measures de ned previously are often referred to as multivariate perfor-mance measures. Note that these measures are highly di erent from the average

zero-one loss, which in the preceding notation equals b+d . In the aforemen-

a+b+c+d

tioned example of fraud detection, when 99:9% of the examples are negatively labeled, the zero-one loss of predicting that all the examples are negatives is 0:1%. In contrast, the recall of such prediction is 0 and hence the F1 score is also 0, which means that the corresponding loss will be 1.

17.5.1 Linear Predictors for Bipartite Ranking

We next describe how to train linear predictors for bipartite ranking. As in the previous section, a linear predictor for ranking is de ned to be

hw(x) = (hw; x1i; : : : ; hw; xri):

The corresponding loss function is one of the multivariate performance measures described before. The loss function depends on y0 = hw(x) via the binary vector it induces, which we denote by

|  |  |
| --- | --- |
| b(y0) = (sign(y10 ); : : : ; sign(yr0 )) 2 f 1gr: | (17.12) |

As in the previous section, to facilitate an e cient algorithm we derive a convex surrogate loss function on . The derivation is similar to the derivation of the generalized hinge loss for the NDCG ranking loss, as described in the previous section.

Our rst observation is that for all the values of de ned before, there is some

1. f 1gr such that b(y0) can be rewritten as

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | | r |  |
| Xi |  |
| b(y0) = argmax | | viyi0: | (17.13) |
| v V | =1 |
|  |

This is clearly true for the case = 0 if we choose V = f 1gr. The two measures for which is not taken to be 0 are precision at k and recall at k. For precision at k we can take V to be the set V k, containing all vectors in f 1gr whose number of ones is at least k. For recall at k, we can take V to be V k, which is de ned analogously. See Exercise [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page249).

1. Multiclass, Ranking, and Complex Prediction Problems

Once we have de ned b as in Equation ([17.13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page245)), we can easily derive a convex surrogate loss as follows. Assuming that y 2 V , we have that

(hw(x); y) = (b(hw(x)); y)

r

X

(b(hw(x)); y) + (bi(hw(x)) yi)hw; xii

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| v V | " | i=1 | : |  |
| (vi yi) hw; xii# |  |
|  |  | r |  |  |
| max | (v; y) + | Xi |  | (17.14) |
| 2 |  | =1 |  |  |

The right-hand side is a convex function with respect to w.

We can now solve the learning problem using SGD as described in Section [17.2.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page234). The main computational bottleneck is calculating a subgradient of the loss func-tion, which is equivalent to nding v that achieves the maximum in Equa-tion ([17.14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page246)) (see Claim [14.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page189)).

In the following we describe how to nd this maximizer e ciently for any performance measure that can be written as a function of the numbers a; b; c; d given in Equation ([17.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page244)), and for which the set V contains all elements in f 1gr for which the values of a; b satisfy some constraints. For example, for \recall at k" the set V is all vectors for which a + b k.

The idea is as follows. For any a; b 2 [r], let

|  |  |  |  |
| --- | --- | --- | --- |
|  | jfi : vi = 1 ^ yi = 1gj = a ^ jfi : vi = 1 ^ yi = 1gj = b g : | |  |
| Ya;b = fv : |  |
| Any vector v 2 |  |  | \ V |
| V falls into Ya;b for some a; b 2 [r]. Furthermore, if Ya;b | |
|  |  |  |  |
| is not empty for some a; b 2 [r] then Ya;b \ V | | = Ya;b. Therefore, we can search | |

within each Ya;b that has a nonempty intersection with V separately, and then

take the optimal value. The key observation is that once we are searching only

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| within Ya;b, the value of is xed so we only need to maximize the expression | | | | | |
| 2Y | r |  |  |  |  |
| Xi | ih | w; x | ii | : |
| max | v |
| v a;b | =1 |  |  |

Suppose the examples are sorted so that hw; x1i hw; xri. Then, it is easy to verify that we would like to set vi to be positive for the smallest indices i. Doing this, with the constraint on a; b, amounts to setting vi = 1 for the a top ranked positive examples and for the b top-ranked negative examples. This yields the following procedure.

|  |  |
| --- | --- |
| 17.6 Summary | 247 |
|  |  |

Solving Equation ([17.14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page246))

input:

(x1; : : : ; xr); (y1; : : : ; yr); w; V;

assumptions:

is a function of a; b; c; d

1. contains all vectors for which f(a; b) = 1 for some function f initialize:
2. = jfi : yi = 1gj, N = jfi : yi = 1gj = (hw; x1i; : : : ; hw; xri), ? = 1 sort examples so that 1 2 r

let i1; : : : ; iP be the (sorted) indices of the positive examples

let j1; : : : ; jN be the (sorted) indices of the negative examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| for a = 0; 1; : : : ; P | |  |  |  |  |  |  |
| c = P a |  |  |  |  |  |  |  |
| for b = 0; 1; : : : ; N such that f(a; b) = 1 | | | | | |  |  |
| d = N b |  |  |  |  |  |  |  |
| calculate using a; b; c; d | | | | | |  |  |
| set v1; : : : ; vr s.t. vi1 | | | | = = via = vj1 = = vjb = 1 | | | |
| and the rest | | of the elements of v equal | | | |  | 1 |
| r | i |  | i |  |
| if? | Pi=1 | |  |  |  |
| set = + |  |  | v |  |  |  |

? = , v? = v

output v?

17.6 Summary

Many real world supervised learning problems can be cast as learning a multiclass predictor. We started the chapter by introducing reductions of multiclass learning to binary learning. We then described and analyzed the family of linear predictors for multiclass learning. We have shown how this family can be used even if the number of classes is extremely large, as long as we have an adequate structure on the problem. Finally, we have described ranking problems. In Chapter [29](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page402) we study the sample complexity of multiclass learning in more detail.

17.7 Bibliographic Remarks

The One-versus-All and All-Pairs approach reductions have been uni ed un-der the framework of Error Correction Output Codes (ECOC) (Dietterich & Bakiri 1995, Allwein, Schapire & Singer 2000). There are also other types of re-ductions such as tree-based classi ers (see, for example, Beygelzimer, Langford & Ravikumar (2007)). The limitations of reduction techniques have been studied

1. Multiclass, Ranking, and Complex Prediction Problems

in (Daniely et al. 2011, Daniely, Sabato & Shwartz 2012). See also Chapter [29](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page402), in which we analyze the sample complexity of multiclass learning.

Direct approaches to multiclass learning with linear predictors have been stud-ied in (Vapnik 1998, Weston & Watkins 1999, Crammer & Singer 2001). In par-ticular, the multivector construction is due to Crammer & Singer (2001).

Collins (2000) has shown how to apply the Perceptron algorithm for structured output problems. See also Collins (2002). A related approach is discriminative learning of conditional random elds; see La erty, McCallum & Pereira (2001). Structured output SVM has been studied in (Weston, Chapelle, Vapnik, Elissee & Sch•olkopf 2002, Taskar, Guestrin & Koller 2003, Tsochantaridis, Hofmann,

Joachims & Altun 2004).

The dynamic procedure we have presented for calculating the prediction hw(x) in the structured output section is similar to the forward-backward variables calculated by the Viterbi procedure in HMMs (see, for instance, (Rabiner & Juang 1986)). More generally, solving the maximization problem in structured output is closely related to the problem of inference in graphical models (see, for example, Koller & Friedman (2009)).

Chapelle, Le & Smola (2007) proposed to learn a ranking function with respect to the NDCG loss using ideas from structured output learning. They also ob-served that the maximization problem in the de nition of the generalized hinge loss is equivalent to the assignment problem.

Agarwal & Roth (2005) analyzed the sample complexity of bipartite ranking. Joachims (2005) studied the applicability of structured output SVM to bipartite ranking with multivariate performance measures.

17.8 Exercises

1. Consider a set S of examples in Rn [k] for which there exist vectors 1; : : : ; k such that every example (x; y) 2 S falls within a ball centered at y whose radius is r 1. Assume also that for every i 6= j, k i jk 4r. Con-sider concatenating each instance by the constant 1 and then applying the multivector construction, namely,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (x; y) = [ 0; : : : ; 0 | | | | | | ; x1; : : : ; xn; 1 ; | | | | | 0; : : : ; 0 | | | | | ]: |
| 2R| |  |  |  |  |  | | |  |  |  | } |  |  |  |  |  |  |
|  | {z } | | | | 2R{z | | | 2R|(k {zy)(n} | | | | |  |
| (y | |  | 1)(n+1) | | |  |  | n+1 | |  | +1) | | | | |  |

Show that there exists a vector w 2 Rk(n+1) such that `(w; (x; y)) = 0 for every (x; y) 2 S.

Hint: Observe that for every example (x; y) 2 S we can write x = y + v for

some kvk r. Now, take w = [w1; : : : ; wk], where wi = [ i ; k ik2=2].

2. Multiclass Perceptron: Consider the following algorithm:

|  |  |
| --- | --- |
| 17.8 Exercises | 249 |
|  |  |

Multiclass Batch Perceptron

Input:

1. training set (x1; y1); : : : ; (xm; ym)
2. class-sensitive feature mapping : X Y ! Rd Initialize: w(1) = (0; : : : ; 0) 2 Rd

For t = 1; 2; : : :

If (9 i and y 6= yi s.t. hw(t); (xi; yi)i hw(t); (xi; y)i) then

w(t+1) = w(t) + (xi; yi) (xi; y)

else

output w(t)

Prove the following:

theorem 17.5 Assume that there exists w? such that for all i and for all

y 6= yi it holds that hw?; (xi; yi)i hw?; (xi; y)i+1. Let R = maxi;y k (xi; yi) (xi; y)k. Then, the multiclass Perceptron algorithm stops after at most (Rkw?k)2 iterations, and when it stops it holds that 8i 2 [m]; yi = argmaxy hw(t); (xi; y)i.

1. Generalize the dynamic programming procedure given in Section [17.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page236) for solv-

^

ing the maximization problem given in the de nition of h in the SGD proce-

dure for multiclass prediction. You can assume that (y0; y) = Pr (yt0; yt)

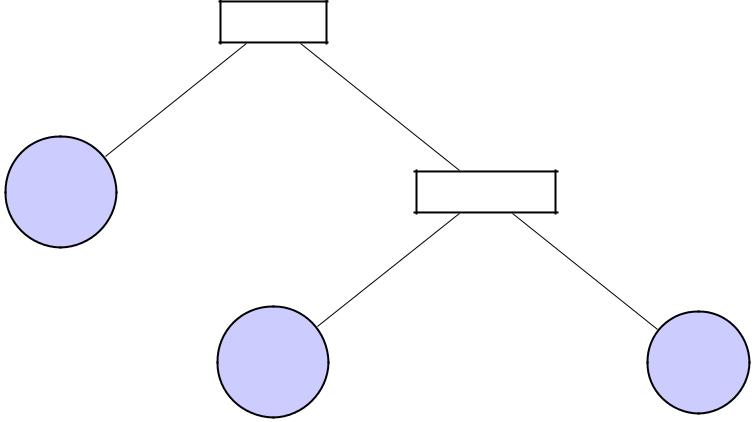
t=1

for some arbitrary function .

1. Prove that Equation ([17.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page241)) holds.
2. Show that the two de nitions of as de ned in Equation ([17.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page245)) and Equa-tion ([17.13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page245)) are indeed equivalent for all the multivariate performance mea-sures.

1. Decision Trees

A decision tree is a predictor, h : X ! Y, that predicts the label associated with an instance x by traveling from a root node of a tree to a leaf. For simplicity we focus on the binary classi cation setting, namely, Y = f0; 1g, but decision trees can be applied for other prediction problems as well. At each node on the root-to-leaf path, the successor child is chosen on the basis of a splitting of the input space. Usually, the splitting is based on one of the features of x or on a prede ned set of splitting rules. A leaf contains a speci c label. An example of a decision tree for the papayas example (described in Chapter [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page33)) is given in the following:



Color?

pale green to pale yellow

other

not-tasty Softness?

other gives slightly to palm pressure

not-tasty tasty

To check if a given papaya is tasty or not, the decision tree rst examines the color of the Papaya. If this color is not in the range pale green to pale yellow, then the tree immediately predicts that the papaya is not tasty without additional tests. Otherwise, the tree turns to examine the softness of the papaya. If the softness level of the papaya is such that it gives slightly to palm pressure, the decision tree predicts that the papaya is tasty. Otherwise, the prediction is \not-tasty." The preceding example underscores one of the main advantages of decision trees { the resulting classi er is very simple to understand and interpret.

|  |  |
| --- | --- |
| 18.1 Sample Complexity | 251 |
|  |  |

18.1 Sample Complexity

A popular splitting rule at internal nodes of the tree is based on thresholding the value of a single feature. That is, we move to the right or left child of the node on the basis of 1[xi< ], where i 2 [d] is the index of the relevant feature and 2 R is the threshold. In such cases, we can think of a decision tree as a splitting of the instance space, X = Rd, into cells, where each leaf of the tree corresponds to one cell. It follows that a tree with k leaves can shatter a set of k instances. Hence, if we allow decision trees of arbitrary size, we obtain a hypothesis class of in nite VC dimension. Such an approach can easily lead to over tting.

To avoid over tting, we can rely on the minimum description length (MDL) principle described in Chapter [7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page83), and aim at learning a decision tree that on one hand ts the data well while on the other hand is not too large.

For simplicity, we will assume that X = f0; 1gd. In other words, each instance is a vector of d bits. In that case, thresholding the value of a single feature corresponds to a splitting rule of the form 1[xi=1] for some i = [d]. For instance, we can model the \papaya decision tree" earlier by assuming that a papaya is parameterized by a two-dimensional bit vector x 2 f0; 1g2, where the bit x1 represents whether the color is pale green to pale yellow or not, and the bit x2 represents whether the softness is gives slightly to palm pressure or not. With this representation, the node Color? can be replaced with 1[x1=1], and the node Softness? can be replaced with 1[x2=1]. While this is a big simpli cation, the algorithms and analysis we provide in the following can be extended to more general cases.

With the aforementioned simplifying assumption, the hypothesis class becomes nite, but is still very large. In particular, any classi er from f0; 1gd to f0; 1g can be represented by a decision tree with 2d leaves and depth of d + 1 (see Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page256)). Therefore, the VC dimension of the class is 2d, which means that the number of examples we need to PAC learn the hypothesis class grows with 2d. Unless d is very small, this is a huge number of examples.

To overcome this obstacle, we rely on the MDL scheme described in Chapter [7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page83). The underlying prior knowledge is that we should prefer smaller trees over larger trees. To formalize this intuition, we rst need to de ne a description language for decision trees, which is pre x free and requires fewer bits for smaller decision trees. Here is one possible way: A tree with n nodes will be described in n + 1 blocks, each of size log2(d + 3) bits. The rst n blocks encode the nodes of the tree, in a depth- rst order (preorder), and the last block marks the end of the code. Each block indicates whether the current node is:

An internal node of the form 1[xi=1] for some i 2 [d] A leaf whose value is 1

A leaf whose value is 0 End of the code

1. Decision Trees

Overall, there are d + 3 options, hence we need log2(d + 3) bits to describe each block.

Assuming each internal node has two children,[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page252) it is not hard to show that this is a pre x-free encoding of the tree, and that the description length of a tree with n nodes is (n + 1) log2(d + 3).

By Theorem [7.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page90) we have that with probability of at least 1 over a sample

of size m, for every n and every decision tree h 2 H with n nodes it holds that

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| LD(h) LS(h) + | r |  |  |  |  | : | (18.1) |
|  | (n + 1) log2 | (d2m | |
|  |  |  |  | + 3) + log(2= ) | |  |  |
|  |  |  |  |  |  |  |  |

This bound performs a tradeo : on the one hand, we expect larger, more complex decision trees to have a smaller training risk, LS(h), but the respective value of n will be larger. On the other hand, smaller decision trees will have a smaller value of n, but LS(h) might be larger. Our hope (or prior knowledge) is that we can nd a decision tree with both low empirical risk, LS(h), and a number of nodes n not too high. Our bound indicates that such a tree will have low true risk, LD(h).

18.2 Decision Tree Algorithms

The bound on LD(h) given in Equation ([18.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page252)) suggests a learning rule for decision trees { search for a tree that minimizes the right-hand side of Equation ([18.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page252)). Unfortunately, it turns out that solving this problem is computationally hard.[2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page252) Consequently, practical decision tree learning algorithms are based on heuristics such as a greedy approach, where the tree is constructed gradually, and locally optimal decisions are made at the construction of each node. Such algorithms cannot guarantee to return the globally optimal decision tree but tend to work reasonably well in practice.

A general framework for growing a decision tree is as follows. We start with a tree with a single leaf (the root) and assign this leaf a label according to a majority vote among all labels over the training set. We now perform a series of iterations. On each iteration, we examine the e ect of splitting a single leaf. We de ne some \gain" measure that quanti es the improvement due to this split. Then, among all possible splits, we either choose the one that maximizes the gain and perform it, or choose not to split the leaf at all.

In the following we provide a possible implementation. It is based on a popular decision tree algorithm known as \ID3" (short for \Iterative Dichotomizer 3"). We describe the algorithm for the case of binary features, namely, X = f0; 1gd,

1. We may assume this without loss of generality, because if a decision node has only one child, we can replace the node by its child without a ecting the predictions of the decision

tree.

1. More precisely, if NP6=P then no algorithm can solve Equation ([18.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page252)) in time polynomial in n; d; and m.

|  |  |
| --- | --- |
| 18.2 Decision Tree Algorithms | 253 |
|  |  |

and therefore all splitting rules are of the form 1[xi=1] for some feature i 2 [d].

We discuss the case of real valued features in Section [18.2.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page255).

The algorithm works by recursive calls, with the initial call being ID3(S; [d]), and returns a decision tree. In the pseudocode that follows, we use a call to a procedure Gain(S; i), which receives a training set S and an index i and evaluates the gain of a split of the tree according to the ith feature. We describe several gain measures in Section [18.2.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page253).

ID3(S; A)

Input: training set S, feature subset A [d]

if all examples in S are labeled by 1, return a leaf 1 if all examples in S are labeled by 0, return a leaf 0

if A = ;, return a leaf whose value = majority of labels in S else :

Let j = argmaxi2A Gain(S; i)

if all examples in S have the same label

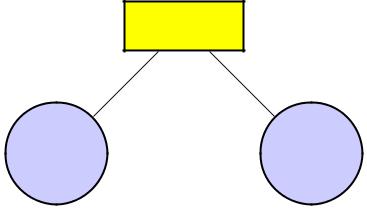
Return a leaf whose value = majority of labels in S else

Let T1 be the tree returned by ID3(f(x; y) 2 S : xj = 1g; A n fjg).

Let T2 be the tree returned by ID3(f(x; y) 2 S : xj = 0g; A n fjg).

Return the tree:

xj = 1?



T2 T1

18.2.1 Implementations of the Gain Measure

Di erent algorithms use di erent implementations of Gain(S; i). Here we present three. We use the notation PS[F ] to denote the probability that an event holds with respect to the uniform distribution over S.

Train Error: The simplest de nition of gain is the decrease in training error. Formally, let C(a) = minfa; 1 ag. Note that the training error before splitting on feature i is C(PS[y = 1]), since we took a majority vote among labels. Similarly, the error after splitting on feature i is

P[xi = 1] C(P[y = 1jxi = 1]) + P[xi = 0]C(P[y = 1jxi = 0]):

S S S S

Therefore, we can de ne Gain to be the di erence between the two, namely,

Gain(S; i) := C(P[y = 1])

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | S |  | j |  |  | j |  |
| S | S | S | S |
|  | P[xi = 1] C(P[y = 1 xi = 1]) + P[xi = 0]C(P[y = 1 xi = 0]) | | | | | | : |

1. Decision Trees

Information Gain: Another popular gain measure that is used in the ID3 and C4.5 algorithms of Quinlan (1993) is the information gain. The information gain is the di erence between the entropy of the label before and after the split, and is achieved by replacing the function C in the previous expression by the entropy function,

C(a) = a log(a) (1 a) log(1 a):

Gini Index: Yet another de nition of a gain, which is used by the CART algorithm of Breiman, Friedman, Olshen & Stone (1984), is the Gini index,

C(a) = 2a(1 a):

Both the information gain and the Gini index are smooth and concave upper bounds of the train error. These properties can be advantageous in some situa-tions (see, for example, Kearns & Mansour (1996)).

18.2.2 Pruning

The ID3 algorithm described previously still su ers from a big problem: The returned tree will usually be very large. Such trees may have low empirical risk, but their true risk will tend to be high { both according to our theoretical analysis, and in practice. One solution is to limit the number of iterations of ID3, leading to a tree with a bounded number of nodes. Another common solution is to prune the tree after it is built, hoping to reduce it to a much smaller tree, but still with a similar empirical error. Theoretically, according to the bound in Equation ([18.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page252)), if we can make n much smaller without increasing LS(h) by much, we are likely to get a decision tree with a smaller true risk.

Usually, the pruning is performed by a bottom-up walk on the tree. Each node might be replaced with one of its subtrees or with a leaf, based on some bound or estimate of LD(h) (for example, the bound in Equation ([18.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page252))). A pseudocode of a common template is given in the following.

Generic Tree Pruning Procedure

input:

function f(T; m) (bound/estimate for the generalization error of a decision tree T , based on a sample of size m),

tree T .

foreach node j in a bottom-up walk on T (from leaves to root):

nd T 0 which minimizes f(T 0; m), where T 0 is any of the following:

the current tree after replacing node j with a leaf 1.

the current tree after replacing node j with a leaf 0.

the current tree after replacing node j with its left subtree.

the current tree after replacing node j with its right subtree.

the current tree.

let T := T 0.

|  |  |
| --- | --- |
| 18.3 Random Forests | 255 |
|  |  |

18.2.3 Threshold-Based Splitting Rules for Real-Valued Features

In the previous section we have described an algorithm for growing a decision tree assuming that the features are binary and the splitting rules are of the form 1[xi=1]. We now extend this result to the case of real-valued features and threshold-based splitting rules, namely, 1[xi< ]. Such splitting rules yield decision stumps, and we have studied them in Chapter [10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page130).

The basic idea is to reduce the problem to the case of binary features as follows. Let x1; : : : ; xm be the instances of the training set. For each real-valued feature i, sort the instances so that x1;i xm;i. De ne a set of thresholds 0;i; : : : ; m+1;i such that j;i 2 (xj;i; xj+1;i) (where we use the convention x0;i = 1 and xm+1;i = 1). Finally, for each i and j we de ne the binary feature 1[xi< j;i]. Once we have constructed these binary features, we can run the ID3 procedure described in the previous section. It is easy to verify that for any decision tree with threshold-based splitting rules over the original real-valued features there exists a decision tree over the constructed binary features with the same training error and the same number of nodes.

If the original number of real-valued features is d and the number of examples is m, then the number of constructed binary features becomes dm. Calculating the Gain of each feature might therefore take O(dm2) operations. However, using a more clever implementation, the runtime can be reduced to O(dm log(m)). The idea is similar to the implementation of ERM for decision stumps as described in Section [10.1.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page133).

18.3 Random Forests

As mentioned before, the class of decision trees of arbitrary size has in nite VC dimension. We therefore restricted the size of the decision tree. Another way to reduce the danger of over tting is by constructing an ensemble of trees. In particular, in the following we describe the method of random forests, introduced by Breiman (2001).

A random forest is a classi er consisting of a collection of decision trees, where each tree is constructed by applying an algorithm A on the training set S and an additional random vector, , where is sampled i.i.d. from some distribution. The prediction of the random forest is obtained by a majority vote over the predictions of the individual trees.

To specify a particular random forest, we need to de ne the algorithm A and the distribution over . There are many ways to do this and here we describe one particular option. We generate as follows. First, we take a random subsample from S with replacements; namely, we sample a new training set S0 of size m0 using the uniform distribution over S. Second, we construct a sequence I1; I2; : : :, where each It is a subset of [d] of size k, which is generated by sampling uniformly at random elements from [d]. All these random variables form the vector . Then,

1. Decision Trees

the algorithm A grows a decision tree (e.g., using the ID3 algorithm) based on

the sample S0, where at each splitting stage of the algorithm, the algorithm is restricted to choosing a feature that maximizes Gain from the set It. Intuitively, if k is small, this restriction may prevent over tting.

18.4 Summary

Decision trees are very intuitive predictors. Typically, if a human programmer creates a predictor it will look like a decision tree. We have shown that the VC dimension of decision trees with k leaves is k and proposed the MDL paradigm for learning decision trees. The main problem with decision trees is that they are computationally hard to learn; therefore we described several heuristic pro-cedures for training them.

18.5 Bibliographic Remarks

Many algorithms for learning decision trees (such as ID3 and C4.5) have been derived by Quinlan (1986). The CART algorithm is due to Breiman et al. (1984). Random forests were introduced by Breiman (2001). For additional reading we refer the reader to (Hastie, Tibshirani & Friedman 2001, Rokach 2007).

The proof of the hardness of training decision trees is given in Hya l & Rivest (1976).

18.6 Exercises

1. 1. Show that any binary classi er h : f0; 1gd 7! 0f; 1g can be implemented as a decision tree of height at most d + 1, with internal nodes of the form (xi = 0?) for some i 2 f1; : : : ; dg.
   1. Conclude that the VC dimension of the class of decision trees over the domain f0; 1gd is 2d.
2. (Suboptimality of ID3)

Consider the following training set, where X = f0; 1g3 and Y = f0; 1g:

((1; 1; 1); 1)

((1; 0; 0); 1)

((1; 1; 0); 0)

((0; 0; 1); 0)

Suppose we wish to use this training set in order to build a decision tree of depth 2 (i.e., for each input we are allowed to ask two questions of the form (xi = 0?) before deciding on the label).

|  |  |
| --- | --- |
| 18.6 Exercises | 257 |
|  |  |

1. Suppose we run the ID3 algorithm up to depth 2 (namely, we pick the root node and its children according to the algorithm, but instead of keeping on with the recursion, we stop and pick leaves according to the majority label in each subtree). Assume that the subroutine used to measure the quality of each feature is based on the entropy function (so we measure the information gain), and that if two features get the same score, one of them is picked arbitrarily. Show that the training error of the resulting decision tree is at least 1=4.
2. Find a decision tree of depth 2 that attains zero training error.

1. Nearest Neighbor

Nearest Neighbor algorithms are among the simplest of all machine learning algorithms. The idea is to memorize the training set and then to predict the label of any new instance on the basis of the labels of its closest neighbors in the training set. The rationale behind such a method is based on the assumption that the features that are used to describe the domain points are relevant to their labelings in a way that makes close-by points likely to have the same label. Furthermore, in some situations, even when the training set is immense, nding a nearest neighbor can be done extremely fast (for example, when the training set is the entire Web and distances are based on links).

Note that, in contrast with the algorithmic paradigms that we have discussed so far, like ERM, SRM, MDL, or RLM, that are determined by some hypothesis class, H, the Nearest Neighbor method gures out a label on any test point without searching for a predictor within some prede ned class of functions.

In this chapter we describe Nearest Neighbor methods for classi cation and regression problems. We analyze their performance for the simple case of binary classi cation and discuss the e ciency of implementing these methods.

19.1 k Nearest Neighbors

Throughout the entire chapter we assume that our instance domain, X , is en-dowed with a metric function . That is, : X X ! R is a function that returns the distance between any two elements of X . For example, if X = Rd then can

q

|  |  |
| --- | --- |
| be the Euclidean distance, (x; x0) = kx x0k = | id=1(xi xi0)2. |
| Let S = (x1; y1); : : : ; (xm; ym) be a sequence of | P |
|  | training examples. For each |

1. 2 X , let 1(x); : : : ; m(x) be a reordering of f1; : : : ; mg according to their distance to x, (x; xi). That is, for all i < m,

(x; x i(x)) (x; x i+1(x)):

For a number k, the k-NN rule for binary classi cation is de ned as follows:

|  |  |
| --- | --- |
| 19.2 Analysis | 259 |
|  |  |

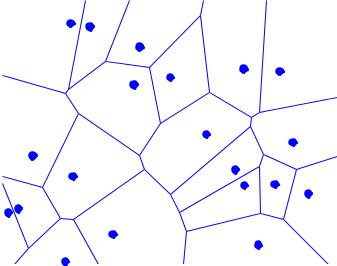


Figure 19.1 An illustration of the decision boundaries of the 1-NN rule. The points depicted are the sample points, and the predicted label of any new point will be the label of the sample point in the center of the cell it belongs to. These cells are called a Voronoi Tessellation of the space.

k-NN

input: a training sample S = (x1; y1); : : : ; (xm; ym)

output: for every point x 2 X ,

return the majority label among fy i(x) : i kg

When k = 1, we have the 1-NN rule:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| hS(x) = y 1(x): | |  |  |  |
| A geometric illustration of the 1-NN rule is given in Figure [19.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page259). | | | |  |
| For regression problems, namely, Y = R, one can de ne the | | prediction to be | | |
|  | 1 | k |
| the average target of the k nearest neighbors. That is, hS(x) = | | | i=1 y i(x). |
| k |
|  | k |  |  | with respect |
| More generally, for some function : (X Y) ! Y, the k-NN rule P | | | | |
| to is: |  |  |  |  |
| hS(x) = (x 1(x); y 1(x)); : : : ; (x k(x); y k(x)) : | | | | (19.1) |

It is easy to verify that we can cast the prediction by majority of labels (for classi cation) or by the averaged target (for regression) as in Equation ([19.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page259)) by an appropriate choice of . The generality can lead to other rules; for example, if Y = R, we can take a weighted average of the targets according to the distance from x:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| k |  |  | (x; x i(x)) | | | |
| Xi |  |  |
|  |  |  |  |  | y i(x): |
|  |  | k |  |  |
| hS(x) = | P | |  |  |
| =1 | j=1 | (x; x | (x)) | |
|  |  | j |  |  |

19.2 Analysis

Since the NN rules are such natural learning methods, their generalization prop-erties have been extensively studied. Most previous results are asymptotic con-sistency results, analyzing the performance of NN rules when the sample size, m,

1. Nearest Neighbor

goes to in nity, and the rate of convergence depends on the underlying distribu-tion. As we have argued in Section [7.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page92), this type of analysis is not satisfactory. One would like to learn from nite training samples and to understand the gen-eralization performance as a function of the size of such nite training sets and clear prior assumptions on the data distribution. We therefore provide a nite-sample analysis of the 1-NN rule, showing how the error decreases as a function of m and how it depends on properties of the distribution. We will also explain how the analysis can be generalized to k-NN rules for arbitrary values of k. In

particular, the analysis speci es the number of examples required to achieve a true error of 2LD(h?) + , where h? is the Bayes optimal hypothesis, assuming that the labeling rule is \well behaved" (in a sense we will de ne later).

19.2.1 A Generalization Bound for the 1-NN Rule

We now analyze the true error of the 1-NN rule for binary classi cation with the 0-1 loss, namely, Y = f0; 1g and `(h; (x; y)) = 1[h(x)6=y]. We also assume throughout the analysis that X = [0; 1]d and is the Euclidean distance.

We start by introducing some notation. Let D be a distribution over X Y. Let DX denote the induced marginal distribution over X and let : Rd ! R be the conditional probability[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page260) over the labels, that is,

(x) = P[y = 1jx]:

Recall that the Bayes optimal rule (that is, the hypothesis that minimizes LD(h) over all functions) is

h?(x) = 1[ (x)>1=2]:

We assume that the conditional probability function is c-Lipschitz for some c > 0: Namely, for all x; x0 2 X ; j (x) (x0)j c kx x0k. In other words, this assumption means that if two vectors are close to each other then their labels are likely to be the same.

The following lemma applies the Lipschitzness of the conditional probability function to upper bound the true error of the 1-NN rule as a function of the expected distance between each test instance and its nearest neighbor in the training set.

lemma 19.1 Let X = [0; 1]d; Y = f0; 1g, and D be a distribution over X Y for which the conditional probability function, , is a c-Lipschitz function. Let

1. = (x1; y1); : : : ; (xm; ym) be an i.i.d. sample and let hS be its corresponding 1-NN hypothesis. Let h? be the Bayes optimal rule for . Then,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | S | E | m | [LD(hS)] 2 LD(h?) + c S | | D | mE;x | | [kx |
|  |  | D |  |  |  |  |  | D |
| 1 | Formally, P[y = 1jx] = lim !0 | | | | D(f(x0;1):x02B(x; )g) | | | , where | |
|  | D(f(x0;y):x02B(x; );y2Yg) | | |

centered around x.

x 1(x)k]:

B(x; ) is a ball of radius

|  |  |
| --- | --- |
| 19.2 Analysis | 261 |
|  |  |

Proof Since LD(hS) = E(x;y) D[1[hS(x)6=y]], we obtain that ES[LD(hS)] is the probability to sample a training set S and an additional example (x; y), such

that the label of 1(x) is di erent from y. In other words, we can rst sample

1. unlabeled examples, Sx = (x1; : : : ; xm), according to DX , and an additional unlabeled example, x DX , then nd 1(x) to be the nearest neighbor of x in Sx, and nally sample y (x) and y 1(x) ( 1(x)). It follows that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E[L (hS)] = | |  | E | [1[y=y0 | | ]] |  |  |
| S | D | Sx DXm;x DX ;y (x);y0 ( 1(x)) | |  | 6 |  |  |  |
|  |  | Sx DXm;x DX | y (x);y0 ( 1(x)) | | 6 |  |  |
|  | = | E | P |  | [y = y0] | | : | (19.2) |

We next upper bound Py (x);y0 (x0)[y 6= y0] for any two domain points x; x0:

1. [y 6= y0] = (x0)(1 (x)) + (1 (x0)) (x)

y (x);y0 (x0)

1. ( (x) (x) + (x0))(1 (x))
   1. (1 (x) + (x) (x0)) (x)
2. 2 (x)(1 (x)) + ( (x) (x0))(2 (x) 1):

Using j2 (x) 1j 1 and the assumption that is c-Lipschitz, we obtain that the probability is at most:

1. [y 6= y0] 2 (x)(1 (x)) + c kx x0k:

y (x);y0 (x0)

Plugging this into Equation ([19.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page261)) we conclude that

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | D |  | x |  |  | S;x | k |  |  |  | k |  |
| E[L |  | (hS)] | E[2 (x)(1 |  | (x))] + c | E | [ | x |  | x 1(x) |  | ]: |

Finally, the error of the Bayes optimal classi er is

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| L | D | (h?) = | E[min | | (x); 1 |  | (x) | ] |  | E[ (x)(1 |  | (x))]: |
|  |  | x | f |  | g |  | x |  |

Combining the preceding two inequalities concludes our proof.

The next step is to bound the expected distance between a random x and its closest element in S. We rst need the following general probability lemma. The lemma bounds the probability weight of subsets that are not hit by a random sample, as a function of the size of that sample.

lemma 19.2 Let C1; : : : ; Cr be a collection of subsets of some domain set, X . Let S be a sequence of m points sampled i.i.d. according to some probability distribution, D over X . Then,

1. 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | 4 | X | P[Ci] | 5 |  |  | r |  | : |
| m e | | |
| S Dm |  |  |

i:Ci\S=;

1. Nearest Neighbor

Proof From the linearity of expectation, we can rewrite:

1. 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 4 | X |  | 5 | r |  |  |  |  |
| E | P[Ci] | X |  | S= |
|  |  |  | = | P[Ci] E 1[Ci | \ | ] : |
| S |  |  |  |  |  | S | ; |  |

i:Ci\S=; i=1

Next, for each i we have

E 1[Ci\S=;] = P[Ci \ S = ;] = (1 P[Ci])m e P[Ci] m:

S S

Combining the preceding two equations we get

1. 3

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 4 | X |  | 5 |  | r |  |  |
| E | P[Ci] |  | X |  | r max P[Ci] e P[Ci] m: |
|  |  |  |  | P[Ci] e P[Ci] m |  |
| S |  |  |  |  |  | i |

i:Ci\S=; i=1

Finally, by a standard calculus, maxa ae ma me1 and this concludes the proof.

Equipped with the preceding lemmas we are now ready to state and prove the main result of this section { an upper bound on the expected error of the 1-NN learning rule.

theorem 19.3 Let X = [0; 1]d; Y = f0; 1g, and D be a distribution over X Y for which the conditional probability function, , is a c-Lipschitz function. Let hS denote the result of applying the 1-NN rule to a sample S Dm. Then,

p 1

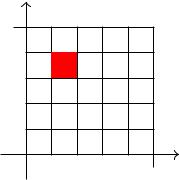
E [LD(hS)] 2 LD(h?) + 4 c d m d+1 :

S Dm

Proof Fix some = 1=T , for some integer T , let r = T d and let C1; : : : ; Cr be the

cover of the set X using boxes of length : Namely, for every ( 1; : : : ; d) 2 [T ]d,

there exists a set Ci of the form fx : 8j; xj 2 [( j 1)=T; j=T ]g. An illustration for d = 2, T = 5 and the set corresponding to = (2; 4) is given in the following.



1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | p | |  |  |  |  |  |  |  |  |  |  | p |  |  |
| For each x; x0 in the same box we have kx x0k | | | | | | | | | | | | | | | | | | | | | | | | d . Otherwise, kx x0kd. | | | | | | | | | | | | |
| Therefore, | |  |  |  |  |  | 2 2 | | | | | |  |  |  |  | 3 |  |  |  |  |  |  |  |  | 2 |  |  |  | 3 | |  | 3 |  |  |  |
| x;S | k |  |  | k |  | S | [ |  |  |  |  |  |  |  |  |  |  |  | [ | |  |  |  |  |  |
| x 1(x) | ] |  |  | Ci | pd + P | | | | | | | | Ci |  | ; |  |  |
| E | [ x |  |  | E | 4 | | P | | 4 | |  |  | 5 | 4 | 5 | pd | | 5 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | i:Ci\S=; | | |  |  |  |  |  |  |  |  |  |  | i:Ci\S6=; | | | |  |  |  |  |  |  |  |
| get that |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | P | S | i:Ci\S6=; Ci] 1 we | | | | | | | |
| and by combining Lemma [19.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page261) with the trivial bound | | | | | | | | | | | | | | | | | | | | | | | | | | |  | [ |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | E [ | |  | |  | |  | | k |  | p | |  |  | |  | r |  |  | | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | x | | x 1(x) | | ] | d | | |  |  | + : | | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | me | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | x;S k | |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| 19.2 Analysis | 263 |
|  |  |

Since the number of boxes is r = (1= )d we get that

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S;x | k |  |  | k |  | p |  | m e | |  |
| E |  |  |  |  |  |  |  | 2d d | + : |
| [ | x |  | x 1(x) ] |  | d |
|  |  |  |

Combining the preceding with Lemma [19.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page260) we obtain that

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | S D | |  | | D | | | |  | m e |  |
|  | E[L (hS)] | |  |  | 2 L (h?) + c p | |  |  |  | 2d d | + : |
|  |  |  | d |  |
|  |  |  |  |  |
| Finally, setting = 2 m 1=(d+1) and noting that | | | | | | | | |  |  |  |
|  | 2d d | + = |  | 2d 2 d md=(d+1) | | + 2 m 1=(d+1) | | | | | |
|  |  | |
|  | m e | |  |  | m e | | | |  |  |  |

1. m 1=(d+1)(1=e + 2) 4m 1=(d+1)

we conclude our proof.

The theorem implies that if we rst x the data-generating distribution and then let m go to in nity, then the error of the 1-NN rule converges to twice the Bayes error. The analysis can be generalized to larger values of k, showing that

p

the expected error of the k-NN rule converges to (1 + 8=k) times the error of the Bayes classi er. This is formalized in Theorem [19.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page265), whose proof is left as a guided exercise.

19.2.2 The \Curse of Dimensionality"

The upper bound given in Theorem [19.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page262) grows with c (the Lipschitz coe cient of ) and with d, the Euclidean dimension of the domain set X . In fact, it is easy

to see that a necessary condition for the last term in Theorem [19.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page262) to be smaller p

than is that m (4 c d= )d+1. That is, the size of the training set should increase exponentially with the dimension. The following theorem tells us that this is not just an artifact of our upper bound, but, for some distributions, this amount of examples is indeed necessary for learning with the NN rule.

theorem 19.4 For any c > 1, and every learning rule, L, there exists a distribution over [0; 1]d f0; 1g, such that (x) is c-Lipschitz, the Bayes error of the distribution is 0, but for sample sizes m (c + 1)d=2, the true error of the rule L is greater than 1=4.

Proof Fix any values of c and d. Let Gdc be the grid on [0; 1]d with distance of 1=c between points on the grid. That is, each point on the grid is of the form (a1=c; : : : ; ad=c) where ai is in f0; : : : ; c 1; cg. Note that, since any two distinct points on this grid are at least 1=c apart, any function : GDC ! [0; 1] is a c-Lipschitz function. It follows that the set of all c-Lipschitz functions over Gdc contains the set of all binary valued functions over that domain. We can therefore invoke the No-Free-Lunch result (Theorem [5.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page61)) to obtain a lower bound on the needed sample sizes for learning that class. The number of points on the grid is (c + 1)d; hence, if m < (c + 1)d=2, Theorem [5.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page61) implies the lower bound we are after. 

1. Nearest Neighbor

The exponential dependence on the dimension is known as the curse of di-mensionality. As we saw, the 1-NN rule might fail if the number of examples is smaller than ((c+1)d). Therefore, while the 1-NN rule does not restrict itself to a prede ned set of hypotheses, it still relies on some prior knowledge { its success depends on the assumption that the dimension and the Lipschitz constant of the underlying distribution, , are not too high.

19.3 E cient Implementation\*

Nearest Neighbor is a learning-by-memorization type of rule. It requires the entire training data set to be stored, and at test time, we need to scan the entire data set in order to nd the neighbors. The time of applying the NN rule is therefore (d m). This leads to expensive computation at test time.

When d is small, several results from the eld of computational geometry have proposed data structures that enable to apply the NN rule in time o(dO(1) log(m)). However, the space required by these data structures is roughly mO(d), which makes these methods impractical for larger values of d.

To overcome this problem, it was suggested to improve the search method by allowing an approximate search. Formally, an r-approximate search procedure is guaranteed to retrieve a point within distance of at most r times the distance to the nearest neighbor. Three popular approximate algorithms for NN are the kd-tree, balltrees, and locality-sensitive hashing (LSH). We refer the reader, for example, to (Shakhnarovich, Darrell & Indyk 2006).

19.4 Summary

The k-NN rule is a very simple learning algorithm that relies on the assumption that \things that look alike must be alike." We formalized this intuition using the Lipschitzness of the conditional probability. We have shown that with a suf-ciently large training set, the risk of the 1-NN is upper bounded by twice the risk of the Bayes optimal rule. We have also derived a lower bound that shows the \curse of dimensionality" { the required sample size might increase expo-nentially with the dimension. As a result, NN is usually performed in practice after a dimensionality reduction preprocessing step. We discuss dimensionality reduction techniques later on in Chapter [23](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page323).

19.5 Bibliographic Remarks

Cover & Hart (1967) gave the rst analysis of 1-NN, showing that its risk con-verges to twice the Bayes optimal error under mild conditions. Following a lemma due to Stone (1977), Devroye & Gy•or (1985) have shown that the k-NN rule

|  |  |
| --- | --- |
| 19.6 Exercises | 265 |
|  |  |

is consistent (with respect to the hypothesis class of all functions from Rd to f0; 1g). A good presentation of the analysis is given in the book of Devroye et al. (1996). Here, we give a nite sample guarantee that explicitly underscores the prior assumption on the distribution. See Section [7.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page92) for a discussion on con-sistency results. Finally, Gottlieb, Kontorovich & Krauthgamer (2010) derived another nite sample bound for NN that is more similar to VC bounds.

19.6 Exercises

In this exercise we will prove the following theorem for the k-NN rule.

theorem 19.5 Let X = [0; 1]d; Y = f0; 1g, and D be a distribution over X Y for which the conditional probability function, , is a c-Lipschitz function. Let hS denote the result of applying the k-NN rule to a sample S Dm, where k 10. Let h? be the Bayes optimal hypothesis. Then,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S D |  |  | r |  |  |  | ! |  | D |  | 6 c p |  |  |
|  | k | | |  |
| E[L | (hS)] |  | 1 + | 8 | |  |  | L |  | (h?) + | d | + k m 1=(d+1): |
|  |  |  |  |  |  |

1. Prove the following lemma.

lemma 19.6 Let C1; : : : ; Cr be a collection of subsets of some domain set, X . Let S be a sequence of m points sampled i.i.d. according to some probability

distribution, D over X . Then, for every k 2,

1. 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | E | 4 |  | X | P[Ci] | 5 |  | 2rk |  |
|  | j |  |  |  |
| S | m | | : |
| m |  |  |  |
|  | D |  |  |  |  |  |  |  |  |

i: Ci\Sj<k

Hints:

Show that

1. 3

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 4 |  | X | 5 | r |  |  |  |  |  |
| S | j | X | j | \ |  | j |  |
|  |  |  | P[Ci] = | S | S | < k] : |
| E |  |  |  | P[Ci] P [ Ci | |  |  |

i: Ci\Sj<k i=1

Fix some i and suppose that k < P[Ci] m=2. Use Cherno 's bound to show that

P [jCi \ Sj < k] P [jCi \ Sj < P[Ci]m=2] e P[Ci] m=8:

S S

Use the inequality maxa ae ma me1 to show that for such i we have

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | [Ci] P [ | Ci |  | S |  | < k] |  | P[Ci]e P[Ci] m=8 |  | 8 |  | : |
| \ | j |  | me | | |
| S j |  |  |  |  |  |

Conclude the proof by using the fact that for the case k P[Ci] m=2 we clearly have:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | [Ci] P [ | Ci |  | S |  | < k] |  | P[Ci] |  | 2k | : |
| \ | j |  | m | |
| S j |  |  |  |  |  |

1. Nearest Neighbor
   1. We use the notation y p as a shorthand for \y is a Bernoulli random variable with expected value p." Prove the following lemma:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lemma 19.7 | Let k 10 and let Z1; : : : ; Zk be independent | | | | | | | | | | | | | | | | | | | Bernoulli random | | | | |
|  | k | | k | i |
| that | P | i | |  | i |  |  | k Pi | | | | | |  | i |  |  | 0 |  |  | Pi=1 |
| variables with |  | [Z = 1] = p | | | | . Denote p = | | | 1 |  |  |  |  | p |  | and p | |  | = | |  | 1 |  | Z . Show |
|  |  |  |  |  |  |
| Z1;:::;Zk y | | |  | p 6 | 0 | |  |  | r | | k | | | ! y | | P | p 6 | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E P [y = 1[p >1=2]] | | | | | | |  | 8 | | | | |  |  |  |  | [y = 1[p>1=2]]: | | | | | | |  |
|  | 1 + | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Hints:

W.l.o.g. assume that p 1=2. Then, Py p[y 6= 1[p>1=2]] = p. Let y0 = 1[p0>1=2].

Show that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | P [y = y0] | | |  | p = | P [p0 > 1=2](1 |  | 2p): |
| Z1;:::;Zk y |  | p | 6 |  |  | Z1;:::;Zk |  |  |
|  |  |  |

Use Cherno 's bound (Lemma [B.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page424)) to show that

P[p0 > 1=2] e k p h( 21p 1);

where

h(a) = (1 + a) log(1 + a) a:

To conclude the proof of the lemma, you can rely on the following inequality (without proving it): For every p 2 [0; 1=2] and k 10:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | (1 2p) e k p + 2 | | | | | | (log(2p)+1) |  |  | | r | k | | | p: |
|  |  |  |  |  |  | k |  |  |  |  |  | 8 | |  |  |
| 3. Fix some p; p0 2 [0; 1] and y0 2 f0; 1g. Show that | | | | | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| y | [y = y0] | |  | y |  | [y = y0] + | | j | p |  | p0 | : | |  |  |
| Pp | 6 | Pp0 | | 6 |  | j |  |  |  |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |

1. Conclude the proof of the theorem according to the following steps:

As in the proof of Theorem [19.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page262), six some > 0 and let C1; : : : ; Cr be the

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| cover of the set | X | using boxes of length . For each x; x0 | | | | | in the same |
|  |  | p |  | p |  |  |  |
| box we have kx x0k | | | d . Otherwise, kx x0k 2 d. Show that | | | | |

1. 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | E[L |  | (hS)] |  | E | 4 | j | | X | [Ci] | | 5 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | D |  |  |  |  | P |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | S |  | S |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | i: Ci\Sj<k | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | i | S;(x;y) h | | | |  | 6 | j 8 |  |  | 2 |  | k |  |  |  | |  |  | k | | | |  |  |  | i |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | p |  |  |  |
|  |  |  | + max | | | P | |  | hS(x) = y | | |  | j | |  | [k]; |  | x | |  |  | x j(x) | |  |  |  |  | d : | | | (19.3) |
|  | Bound the rst summand using Lemma [19.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page265). | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | To bound the second summand, let us x Sjx | | | | | | | | | | | | | | | | | | |  | and x such that all the k | | | | | | | | | | | |
|  | neighbors of x in Sjx are at distance of at most | | | | | | | | | | | | | | | | | | | | | | p | |  |  |  |  |  |  |  |  |
|  |  | d from x. W.l.o.g | | | | | | | | |
|  | assume that the k NN are x1; : : : ; xk. Denote pi = (xi) and let p = | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | 1 | | Pi pi. Use | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | k |  |  | [h |  | ([x](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page266)) = y] | |  |  |  |  |  | [h | |  |  | (x) = y] + | | | | | |  | p |  | (x) : | | |
|  |  |  |  |  |  | Exercise [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page266) to show that | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | E | | P |  |  | S |  |  |  |  | E | | P |  |  | S | |  | 6 |  |  |  |  | j |  | |  |  | j |
|  |  |  | y1;:::;yj y (x) | | | | |  |  |  | 6y1;:::;yj y p | | | | | | |  |  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| 19.6 Exercises | 267 |
|  |  |

W.l.o.g. assume that p 1=2. Now use Lemma [19.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page266) to show that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y1 | ;:::;yj y |  | p | 6 |  |  | r |  | k ! | | | y | P | p | 6 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | P P [hS(x) = y] | | | |  | 1 + | 8 | | |  |  |  |  | [1[p>1=2] | = y]: |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Show that

P [1[p>1=2] 6= y] = p = minfp; 1 pg minf (x); 1 (x)g+ jp (x)j:

y p

Combine all the preceding to obtain that the second summand in Equa-

tion ([19.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page266)) is bounded by

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 1 + r | | |  |  |  |  | ! LD(h?) + 3 c pd: | | | | | | | | | | | | |  |  |
|  |  |  |  |  |  | k | | |  |  |
|  |  |  |  |  | 8 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | | |  | |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Use r = (2= )d to obtain that: | | | | | | | | | |  |  |  |  | ! |  |  |  |  |  |  |  |  |  |  |  |  |
| S | D |  |  | |  |  | r | | |  | | |  |  | D | (h?) + 3 c p | | | | |  |  |  | m | |
|  | k | | | |  |  |  |  |
| E[L |  |  |  |  |  |  |  |  |  | 8 | | |  |  |  |  |  | | + | 2(2= )d k | : |
|  | (hS)] |  |  | 1 + | | |  |  |  |  | L |  | d | |
|  |  |  |  |  |  |  |  |  |  |  |  |
| Set = 2m 1=(d+1) and use | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 c m 1=(d+1) p | | | |  | | + | 2k | | m 1=(d+1) | | | | | | | | | 6cp |  | + k m 1=(d+1) | | | | | | |
| d | | d |
| e | |
| to conclude the proof. | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. Neural Networks

An arti cial neural network is a model of computation inspired by the structure of neural networks in the brain. In simpli ed models of the brain, it consists of a large number of basic computing devices (neurons) that are connected to each other in a complex communication network, through which the brain is able to carry out highly complex computations. Arti cial neural networks are formal computation constructs that are modeled after this computation paradigm.

Learning with neural networks was proposed in the mid-20th century. It yields an e ective learning paradigm and has recently been shown to achieve cutting-edge performance on several learning tasks.

A neural network can be described as a directed graph whose nodes correspond to neurons and edges correspond to links between them. Each neuron receives as input a weighted sum of the outputs of the neurons connected to its incoming edges. We focus on feedforward networks in which the underlying graph does not contain cycles.

In the context of learning, we can de ne a hypothesis class consisting of neural network predictors, where all the hypotheses share the underlying graph struc-ture of the network and di er in the weights over edges. As we will show in Section [20.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page271), every predictor over n variables that can be implemented in time T (n) can also be expressed as a neural network predictor of size O(T (n)2), where the size of the network is the number of nodes in it. It follows that the family of hypothesis classes of neural networks of polynomial size can su ce for all practical learning tasks, in which our goal is to learn predictors which can be implemented e ciently. Furthermore, in Section [20.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page274) we will show that the sam-ple complexity of learning such hypothesis classes is also bounded in terms of the size of the network. Hence, it seems that this is the ultimate learning paradigm we would want to adapt, in the sense that it both has a polynomial sample com-plexity and has the minimal approximation error among all hypothesis classes consisting of e ciently implementable predictors.

The caveat is that the problem of training such hypothesis classes of neural net-work predictors is computationally hard. This will be formalized in Section [20.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page276). A widely used heuristic for training neural networks relies on the SGD frame-work we studied in Chapter [14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page184). There, we have shown that SGD is a successful learner if the loss function is convex. In neural networks, the loss function is highly nonconvex. Nevertheless, we can still implement the SGD algorithm and

|  |  |
| --- | --- |
| 20.1 Feedforward Neural Networks | 269 |
|  |  |

hope it will nd a reasonable solution (as happens to be the case in several practical tasks). In Section [20.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page277) we describe how to implement SGD for neural networks. In particular, the most complicated operation is the calculation of the gradient of the loss function with respect to the parameters of the network. We present the backpropagation algorithm that e ciently calculates the gradient.

20.1 Feedforward Neural Networks

The idea behind neural networks is that many neurons can be joined together by communication links to carry out complex computations. It is common to describe the structure of a neural network as a graph whose nodes are the neurons and each (directed) edge in the graph links the output of some neuron to the input of another neuron. We will restrict our attention to feedforward network structures in which the underlying graph does not contain cycles.

A feedforward neural network is described by a directed acyclic graph, G = (V; E), and a weight function over the edges, w : E ! R. Nodes of the graph correspond to neurons. Each single neuron is modeled as a simple scalar func-tion, : R ! R. We will focus on three possible functions for : the sign function, (a) = sign(a), the threshold function, (a) = 1[a>0], and the sig-moid function, (a) = 1=(1 + exp( a)), which is a smooth approximation to the threshold function. We call the \activation" function of the neuron. Each edge in the graph links the output of some neuron to the input of another neuron. The input of a neuron is obtained by taking a weighted sum of the outputs of all the neurons connected to it, where the weighting is according to w.

To simplify the description of the calculation performed by the network, we further assume that the network is organized in layers. That is, the set of nodes can be decomposed into a union of (nonempty) disjoint subsets, V = [Tt=0Vt, such that every edge in E connects some node in Vt 1 to some node in Vt, for some t 2 [T ]. The bottom layer, V0, is called the input layer. It contains n + 1 neurons, where n is the dimensionality of the input space. For every i 2 [n], the output of neuron i in V0 is simply xi. The last neuron in V0 is the \constant" neuron, which always outputs 1. We denote by vt;i the ith neuron of the tth layer and by ot;i(x) the output of vt;i when the network is fed with the input vector x. Therefore, for i 2 [n] we have o0;i(x) = xi and for i = n + 1 we have o0;i(x) = 1. We now proceed with the calculation in a layer by layer manner. Suppose we have calculated the outputs of the neurons at layer t. Then, we can calculate the outputs of the neurons at layer t + 1 as follows. Fix some vt+1;j 2 Vt+1. Let at+1;j(x) denote the input to vt+1;j when the network is fed with the input vector x. Then,

X

at+1;j(x) = w((vt;r; vt+1;j)) ot;r(x);

r: (vt;r;vt+1;j)2E

1. Neural Networks

and

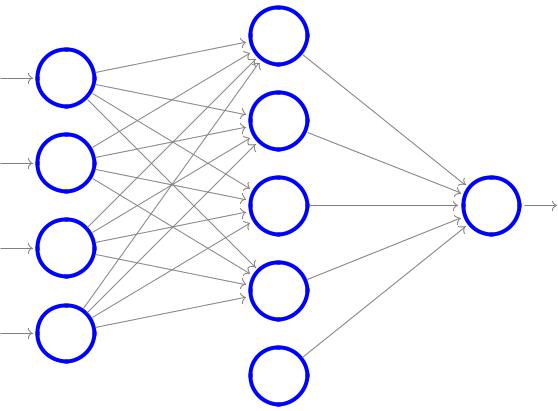
ot+1;j(x) = (at+1;j(x)) :

That is, the input to vt+1;j is a weighted sum of the outputs of the neurons in Vt that are connected to vt+1;j, where weighting is according to w, and the output of vt+1;j is simply the application of the activation function on its input.

Layers V1; : : : ; VT 1 are often called hidden layers. The top layer, VT , is called the output layer. In simple prediction problems the output layer contains a single neuron whose output is the output of the network.

We refer to T as the number of layers in the network (excluding V0), or the \depth" of the network. The size of the network is jV j. The \width" of the network is maxt jVtj. An illustration of a layered feedforward neural network of depth 2, size 10, and width 5, is given in the following. Note that there is a neuron in the hidden layer that has no incoming edges. This neuron will output the constant (0).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Input | Hidden | Output |  |
|  | layer | layer | layer |  |
|  | (V0) | (V1) | (V2) |  |
|  |  | v1;1 |  |  |
| x1 | v0;1 |  |  |  |
|  |  | v1;2 |  |  |
| x2 | v0;2 |  |  |  |
|  |  | v1;3 | v2;1 | Output |
| x3 | v0;3 |  |  |  |
|  |  | v1;4 |  |  |
| constant | v0;4 |  |  |  |
|  |  | v1;5 |  |  |



20.2 Learning Neural Networks

Once we have speci ed a neural network by (V; E; ; w), we obtain a function hV;E; ;w : RjV0j 1 ! RjVT j. Any set of such functions can serve as a hypothesis class for learning. Usually, we de ne a hypothesis class of neural network predic-tors by xing the graph (V; E) as well as the activation function and letting the hypothesis class be all functions of the form hV;E; ;w for some w : E ! R. The triplet (V; E; ) is often called the architecture of the network. We denote the hypothesis class by

|  |  |
| --- | --- |
| HV;E; = fhV;E; ;w : w is a mapping from E to Rg: | (20.1) |

|  |  |
| --- | --- |
| 20.3 The Expressive Power of Neural Networks | 271 |
|  |  |

That is, the parameters specifying a hypothesis in the hypothesis class are the weights over the edges of the network.

We can now study the approximation error, estimation error, and optimization error of such hypothesis classes. In Section [20.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page271) we study the approximation error of HV;E; by studying what type of functions hypotheses in HV;E; can implement, in terms of the size of the underlying graph. In Section [20.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page274) we study the estimation error of HV;E; , for the case of binary classi cation (i.e., VT = 1 and is the sign function), by analyzing its VC dimension. Finally, in Section [20.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page276) we show that it is computationally hard to learn the class HV;E; , even if the underlying graph is small, and in Section [20.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page277) we present the most commonly used heuristic for training HV;E; .

20.3 The Expressive Power of Neural Networks

In this section we study the expressive power of neural networks, namely, what type of functions can be implemented using a neural network. More concretely, we will x some architecture, V; E; , and will study what functions hypotheses in HV;E; can implement, as a function of the size of V .

We start the discussion with studying which type of Boolean functions (i.e., functions from f 1gn to f 1g) can be implemented by HV;E;sign. Observe that for every computer in which real numbers are stored using b bits, whenever we calculate a function f : Rn ! R on such a computer we in fact calculate a function g : f 1gnb ! f 1gb. Therefore, studying which Boolean functions can be implemented by HV;E;sign can tell us which functions can be implemented on a computer that stores real numbers using b bits.

We begin with a simple claim, showing that without restricting the size of the network, every Boolean function can be implemented using a neural network of depth 2.

claim 20.1 For every n, there exists a graph (V; E) of depth 2, such that HV;E;sign contains all functions from f 1gn to f 1g.

Proof We construct a graph with jV0j = n + 1; jV1j = 2n + 1; and jV2j = 1. Let E be all possible edges between adjacent layers. Now, let f : f 1gn ! f 1g be some Boolean function. We need to show that we can adjust the weights so that the network will implement f. Let u1; : : : ; uk be all vectors in f 1gn on which f outputs 1. Observe that for every i and every x 2 f 1gn, if x 6= ui then hx; uii n 2 and if x = ui then hx; uii = n. It follows that the function gi(x) = sign(hx; ui i n + 1) equals 1 if and only if x = ui. It follows that we can adapt the weights between V0 and V1 so that for every i 2 [k], the neuron v1;i implements the function gi(x). Next, we observe that f(x) is the disjunction of

1. Neural Networks

the functions gi(x), and therefore can be written as

k

!

X

f(x) = sign

gi(x) + k 1

;

i=1

which concludes our proof.

The preceding claim shows that neural networks can implement any Boolean function. However, this is a very weak property, as the size of the resulting network might be exponentially large. In the construction given at the proof of Claim [20.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page271), the number of nodes in the hidden layer is exponentially large. This is not an artifact of our proof, as stated in the following theorem.

theorem 20.2 For every n, let s(n) be the minimal integer such that there exists a graph (V; E) with jV j = s(n) such that the hypothesis class HV;E;sign contains all the functions from f0; 1gn to f0; 1g. Then, s(n) is exponential in n. Similar results hold for HV;E; where is the sigmoid function.

Proof Suppose that for some (V; E) we have that HV;E;sign contains all functions from f0; 1gn to f0; 1g. It follows that it can shatter the set of m = 2n vectors in f0; 1gn and hence the VC dimension of HV;E;sign is 2n. On the other hand, the VC dimension of HV;E;sign is bounded by O(jE j log(jEj)) O(jV j3), as we will show in the next section. This implies that jV j (2n=3), which concludes our proof for the case of networks with the sign activation function. The proof for the sigmoid case is analogous. 

Remark 20.1 It is possible to derive a similar theorem for HV;E; for any , as long as we restrict the weights so that it is possible to express every weight using a number of bits which is bounded by a universal constant. We can even con-sider hypothesis classes where di erent neurons can employ di erent activation functions, as long as the number of allowed activation functions is also nite.

Which functions can we express using a network of polynomial size? The pre-ceding claim tells us that it is impossible to express all Boolean functions using a network of polynomial size. On the positive side, in the following we show that all Boolean functions that can be calculated in time O(T (n)) can also be expressed by a network of size O(T (n)2).

theorem 20.3 Let T : N ! N and for every n, let Fn be the set of functions that can be implemented using a Turing machine using runtime of at most T (n). Then, there exist constants b; c 2 R+ such that for every n, there is a graph (Vn; En) of size at most c T (n)2 + b such that HVn;En;sign contains Fn.

The proof of this theorem relies on the relation between the time complexity of programs and their circuit complexity (see, for example, Sipser (2006)). In a nutshell, a Boolean circuit is a type of network in which the individual neurons

|  |  |
| --- | --- |
| 20.3 The Expressive Power of Neural Networks | 273 |
|  |  |

implement conjunctions, disjunctions, and negation of their inputs. Circuit com-plexity measures the size of Boolean circuits required to calculate functions. The relation between time complexity and circuit complexity can be seen intuitively as follows. We can model each step of the execution of a computer program as a simple operation on its memory state. Therefore, the neurons at each layer of the network will re ect the memory state of the computer at the corresponding time, and the translation to the next layer of the network involves a simple calculation that can be carried out by the network. To relate Boolean circuits to networks with the sign activation function, we need to show that we can implement the operations of conjunction, disjunction, and negation, using the sign activation function. Clearly, we can implement the negation operator using the sign activa-tion function. The following lemma shows that the sign activation function can also implement conjunctions and disjunctions of its inputs.

lemma 20.4 Suppose that a neuron v, that implements the sign activation function, has k incoming edges, connecting it to neurons whose outputs are in f 1g. Then, by adding one more edge, linking a \constant" neuron to v, and by adjusting the weights on the edges to v, the output of v can implement the conjunction or the disjunction of its inputs.

Proof Simply observe that if f : f 1gk ! f 1g is the conjunction func-

Pk

tion, f(x) = ^ixi, then it can be written as f(x) = sign 1 k + i=1 xi . Similarly, the disjunction function, f(x) = \_ixi, can be written as f(x) =

|  |  |  |
| --- | --- | --- |
| sign k 1 + | Pi=1 xi | . |
|  | k |  |

So far we have discussed Boolean functions. In Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page282) we show that neural networks are universal approximators. That is, for every xed precision param-eter, > 0, and every Lipschitz function f : [ 1; 1]n ! [ 1; 1], it is possible to construct a network such that for every input x 2 [ 1; 1]n, the network outputs a number between f(x) and f(x) + . However, as in the case of Boolean functions, the size of the network here again cannot be polynomial in n. This is formalized in the following theorem, whose proof is a direct corollary of Theo-rem [20.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page272) and is left as an exercise.

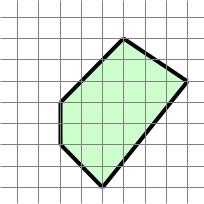
theorem 20.5 Fix some 2 (0; 1). For every n, let s(n) be the minimal integer such that there exists a graph (V; E) with jV j = s(n) such that the hypothesis class HV;E; , with being the sigmoid function, can approximate, to within precision of , every 1-Lipschitz function f : [ 1; 1]n ! [ 1; 1]. Then s(n) is exponential in n.

20.3.1 Geometric Intuition

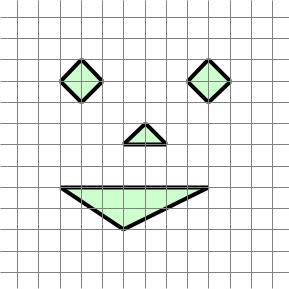
We next provide several geometric illustrations of functions f : R2 ! f 1g and show how to express them using a neural network with the sign activation function.

1. Neural Networks

Let us start with a depth 2 network, namely, a network with a single hidden layer. Each neuron in the hidden layer implements a halfspace predictor. Then, the single neuron at the output layer applies a halfspace on top of the binary outputs of the neurons in the hidden layer. As we have shown before, a halfspace can implement the conjunction function. Therefore, such networks contain all hypotheses which are an intersection of k 1 halfspaces, where k is the number of neurons in the hidden layer; namely, they can express all convex polytopes with k 1 faces. An example of an intersection of 5 halfspaces is given in the following.



We have shown that a neuron in layer V2 can implement a function that indicates whether x is in some convex polytope. By adding one more layer, and letting the neuron in the output layer implement the disjunction of its inputs, we get a network that computes the union of polytopes. An illustration of such a function is given in the following.



20.4 The Sample Complexity of Neural Networks

Next we discuss the sample complexity of learning the class HV;E; . Recall that the fundamental theorem of learning tells us that the sample complexity of learn-ing a hypothesis class of binary classi ers depends on its VC dimension. There-fore, we focus on calculating the VC dimension of hypothesis classes of the form HV;E; , where the output layer of the graph contains a single neuron.

We start with the sign activation function, namely, with HV;E;sign. What is the VC dimension of this class? Intuitively, since we learn jEj parameters, the VC dimension should be order of jEj. This is indeed the case, as formalized by the following theorem.

theorem 20.6 The VC dimension of HV;E;sign is O(jEj log(jEj)).

|  |  |
| --- | --- |
| 20.4 The Sample Complexity of Neural Networks | 275 |
|  |  |

Proof To simplify the notation throughout the proof, let us denote the hy-pothesis class by H. Recall the de nition of the growth function, H(m), from Section [6.5.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page73). This function measures maxC X:jC j=m jHC j, where HC is the re-striction of H to functions from C to f0; 1g. We can naturally extend the de - nition for a set of functions from X to some nite set Y, by letting HC be the restriction of H to functions from C to Y, and keeping the de nition of H(m) intact.

Our neural network is de ned by a layered graph. Let V0; : : : ; VT be the layers of the graph. Fix some t 2 [T ]. By assigning di erent weights on the edges between Vt 1 and Vt, we obtain di erent functions from RjVt 1j ! f 1gjVtj. Let H(t) be the class of all possible such mappings from RjVt 1j ! f 1gjVtj. Then,

1. can be written as a composition, H = H(T ) : : : H(1). In Exercise [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page282) we show that the growth function of a composition of hypothesis classes is bounded by the products of the growth functions of the individual classes. Therefore,

T

Y

H(m) H(t) (m):

t=1

In addition, each H(t) can be written as a product of function classes, H(t) = H(t;1) H(t;jVtj), where each H(t;j) is all functions from layer t 1 to f 1g that the jth neuron of layer t can implement. In Exercise [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page282) we bound product classes, and this yields

jVtj

Y

H(t) (m) H(t;i) (m):

i=1

Let dt;i be the number of edges that are headed to the ith neuron of layer t. Since the neuron is a homogenous halfspace hypothesis and the VC dimension of homogenous halfspaces is the dimension of their input, we have by Sauer's lemma that

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| H(t;i) (m) | dt;i | |  | dt;i | (em) t;i : |
|  |  | em |  | d |
|  |  |  |  |
| Overall, we obtained that |  |  |  |  |  |
| H(m) (em)Pt;i dt;i | | | | | = (em)jEj: |

Now, assume that there are m shattered points. Then, we must have H(m) = 2m, from which we obtain

2m (em)jEj ) m jEj log(em)= log(2):

The claim follows by Lemma [A.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419).

Next, we consider HV;E; , where is the sigmoid function. Surprisingly, it turns out that the VC dimension of HV;E; is lower bounded by (jEj2) (see Exercise [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page282).) That is, the VC dimension is the number of tunable parameters squared. It is also possible to upper bound the VC dimension by O(jV j2 jEj2), but the proof is beyond the scope of this book. In any case, since in practice

1. Neural Networks

we only consider networks in which the weights have a short representation as oating point numbers with O(1) bits, by using the discretization trick we easily

obtain that such networks have a VC dimension of O(jEj), even if we use the sigmoid activation function.

20.5 The Runtime of Learning Neural Networks

In the previous sections we have shown that the class of neural networks with an underlying graph of polynomial size can express all functions that can be imple-mented e ciently, and that the sample complexity has a favorable dependence on the size of the network. In this section we turn to the analysis of the time complexity of training neural networks.

We rst show that it is NP hard to implement the ERM rule with respect to HV;E;sign even for networks with a single hidden layer that contain just 4 neurons in the hidden layer.

theorem 20.7 Let k 3. For every n, let (V; E) be a layered graph with n input nodes, k + 1 nodes at the (single) hidden layer, where one of them is the constant neuron, and a single output node. Then, it is NP hard to implement the ERM rule with respect to HV;E;sign.

The proof relies on a reduction from the k-coloring problem and is left as Exercise [6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page283).

One way around the preceding hardness result could be that for the purpose of learning, it may su ce to nd a predictor h 2 H with low empirical error, not necessarily an exact ERM. However, it turns out that even the task of nd-ing weights that result in close-to-minimal empirical error is computationally infeasible (see (Bartlett & Ben-David 2002)).

One may also wonder whether it may be possible to change the architecture of the network so as to circumvent the hardness result. That is, maybe ERM with respect to the original network structure is computationally hard but ERM with respect to some other, larger, network may be implemented e ciently (see Chapter [8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page100) for examples of such cases). Another possibility is to use other acti-vation functions (such as sigmoids, or any other type of e ciently computable activation functions). There is a strong indication that all of such approaches are doomed to fail. Indeed, under some cryptographic assumption, the problem of learning intersections of halfspaces is known to be hard even in the repre-sentation independent model of learning (see Klivans & Sherstov (2006)). This implies that, under the same cryptographic assumption, any hypothesis class which contains intersections of halfspaces cannot be learned e ciently.

A widely used heuristic for training neural networks relies on the SGD frame-work we studied in Chapter [14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page184). There, we have shown that SGD is a successful learner if the loss function is convex. In neural networks, the loss function is highly nonconvex. Nevertheless, we can still implement the SGD algorithm and

|  |  |
| --- | --- |
| 20.6 SGD and Backpropagation | 277 |
|  |  |

hope it will nd a reasonable solution (as happens to be the case in several practical tasks).

20.6 SGD and Backpropagation

The problem of nding a hypothesis in HV;E; with a low risk amounts to the problem of tuning the weights over the edges. In this section we show how to apply a heuristic search for good weights using the SGD algorithm. Throughout this section we assume that is the sigmoid function, (a) = 1=(1 + e a), but the derivation holds for any di erentiable scalar function.

Since E is a nite set, we can think of the weight function as a vector w 2 RjEj. Suppose the network has n input neurons and k output neurons, and denote by hw : Rn ! Rk the function calculated by the network if the weight function is de ned by w. Let us denote by (hw(x); y) the loss of predicting hw(x) when the target is y 2 Y. For concreteness, we will take to be the squared loss, (hw(x); y) = 12 khw(x) yk2; however, similar derivation can be obtained for every di erentiable function. Finally, given a distribution D over the examples domain, Rn Rk, let LD(w) be the risk of the network, namely,

LD(w) = E [ (hw(x); y)] :

(x;y) D

Recall the SGD algorithm for minimizing the risk function LD(w). We repeat the pseudocode from Chapter [14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page184) with a few modi cations, which are relevant to the neural network application because of the nonconvexity of the objective function. First, while in Chapter [14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page184) we initialized w to be the zero vector, here we initialize w to be a randomly chosen vector with values close to zero. This is because an initialization with the zero vector will lead all hidden neurons to have the same weights (if the network is a full layered network). In addition, the hope is that if we repeat the SGD procedure several times, where each time we initialize the process with a new random vector, one of the runs will lead to a good local minimum. Second, while a xed step size, , is guaranteed to be good enough for convex problems, here we utilize a variable step size, t, as de ned in Section [14.4.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page194). Because of the nonconvexity of the loss function, the choice of the sequence t is more signi cant, and it is tuned in practice by a trial and error manner. Third, we output the best performing vector on a validation set. In addition, it is sometimes helpful to add regularization on the weights, with parameter . That is, we try to minimize LD(w) + 2 kwk2. Finally, the gradient does not have a closed form solution. Instead, it is implemented using the backpropagation algorithm, which will be described in the sequel.

1. Neural Networks

SGD for Neural Networks

parameters:

number of iterations

step size sequence 1; 2; : : : ;

regularization parameter > 0

input:

layered graph (V; E)

di erentiable activation function : R ! R

initialize:

choose w(1) 2 RjEj at random

(from a distribution s.t. w(1) is close enough to 0)

for i = 1; 2; : : : ;

sample (x; y) D

calculate gradient vi = backpropagation(x; y; w; (V; E); )

update w(i+1) = w(i) i(vi + w(i))

output:

w is the best performing w(i) on a validation set

Backpropagation

input:

example (x; y), weight vector w, layered graph (V; E), activation function : R ! R

initialize:

denote layers of the graph V0; : : : ; VT where Vt = fvt;1; : : : ; vt;kt g de ne Wt;i;j as the weight of (vt;j; vt+1;i)

(where we set Wt;i;j = 0 if (vt;j; vt+1;i) 2= E)

forward:

set o0 = x

for t = 1; : : : ; T

for i = 1; : : : ; kt

set a = Pkt 1 W o

t;i j=1 t 1;i;j t 1;j

set ot;i = (at;i)

backward:

set T = oT y

for t = T 1; T 2; : : : ; 1

for i = 1; : : : ; kt

t;i = Pkt+1 Wt;j;i t+1;j 0(at+1;j)

j=1

output:

foreach edge (vt 1;j; vt;i) 2 E

set the partial derivative to t;i 0(at;i) ot 1;j

|  |  |
| --- | --- |
| 20.6 SGD and Backpropagation | 279 |
|  |  |

Explaining How Backpropagation Calculates the Gradient:

We next explain how the backpropagation algorithm calculates the gradient of the loss function on an example (x; y) with respect to the vector w. Let us rst recall a few de nitions from vector calculus. Each element of the gradient is the partial derivative with respect to the variable in w corresponding to one of the edges of the network. Recall the de nition of a partial derivative. Given a function f : Rn ! R, the partial derivative with respect to the ith variable at w is obtained by xing the values of w1; : : : ; wi 1; wi+1; wn, which yields the scalar function g : R ! R de ned by g(a) = f((w1; : : : ; wi 1; wi + a; wi+1; : : : ; wn)), and then taking the derivative of g at 0. For a function with multiple outputs,

1. : Rn ! Rm, the Jacobian of f at w 2 Rn, denoted Jw(f), is the m n matrix whose i; j element is the partial derivative of fi : Rn ! R w.r.t. its jth variable at w. Note that if m = 1 then the Jacobian matrix is the gradient of the function (represented as a row vector). Two examples of Jacobian calculations, which we will later use, are as follows.

Let f(w) = Aw for A 2 Rm;n. Then Jw(f) = A.

For every n, we use the notation to denote the function from Rn to Rn

which applies the sigmoid function element-wise. That is, = ( ) means

that for every i we have i = ( i) = 1 . It is easy to verify

1+exp( i)

that J ( ) is a diagonal matrix whose (i; i) entry is 0( i), where 0 is the derivative function of the (scalar) sigmoid function, namely, 0( i) =

1 . We also use the notation diag( 0( )) to denote this

(1+exp( i))(1+exp( i))

matrix.

The chain rule for taking the derivative of a composition of functions can be written in terms of the Jacobian as follows. Given two functions f : Rn ! Rm and g : Rk ! Rn, we have that the Jacobian of the composition function, (f g) : Rk ! Rm, at w, is

Jw(f g) = Jg(w)(f)Jw(g):

For example, for g(w) = Aw, where A 2 Rn;k, we have that

Jw( g) = diag( 0(Aw)) A:

To describe the backpropagation algorithm, let us rst decompose V into the

layers of the graph, V = [Tt=0Vt. For every t, let us write Vt = fvt;1; : : : ; vt;kt g, where kt = jVtj. In addition, for every t denote Wt 2 Rkt+1;kt a matrix which

gives a weight to every potential edge between Vt and Vt+1. If the edge exists in

1. then we set Wt;i;j to be the weight, according to w, of the edge (vt;j; vt+1;i). Otherwise, we add a \phantom" edge and set its weight to be zero, Wt;i;j = 0. Since when calculating the partial derivative with respect to the weight of some edge we x all other weights, these additional \phantom" edges have no e ect

on the partial derivative with respect to existing edges. It follows that we can assume, without loss of generality, that all edges exist, that is, E = [t(Vt Vt+1).

1. Neural Networks

Next, we discuss how to calculate the partial derivatives with respect to the edges from Vt 1 to Vt, namely, with respect to the elements in Wt 1. Since we x all other weights of the network, it follows that the outputs of all the neurons in Vt 1 are xed numbers which do not depend on the weights in Wt 1. Denote the corresponding vector by ot 1. In addition, let us denote by `t : Rkt ! R the loss function of the subnetwork de ned by layers Vt; : : : ; VT as a function of the outputs of the neurons in Vt. The input to the neurons of Vt can be written as at = Wt 1ot 1 and the output of the neurons of Vt is ot = (at). That is, for every j we have ot;j = (at;j). We obtain that the loss, as a function of Wt 1, can be written as

gt(Wt 1) = `t(ot) = `t( (at)) = `t( (Wt 1ot 1)):

It would be convenient to rewrite this as follows. Let wt 1 2 Rkt 1kt be the column vector obtained by concatenating the rows of Wt 1 and then taking the transpose of the resulting long vector. De ne by Ot 1 the kt (kt 1kt) matrix

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | ot> 1 | 0 |  |  | 0 |  | 1 |  |  |
|  | 0 | ot> | 1 | 0 |  |  |  |
| Ot 1 = | B | .. | .. |  | .. | .. |  | C | : | (20.2) |
|  | B | . | . |  | . . | |  | C |  |  |
|  | B |  |  |  |  |  |  | C |  |  |
|  | B |  |  |  |  |  |  | C |  |  |
|  | B | 0 | 0 |  |  | ot> | 1 | C |  |  |
|  | @ |  |  |  |  |  |  | A |  |  |

Then, Wt 1ot 1 = Ot 1wt 1, so we can also write

gt(wt 1) = `t( (Ot 1 wt 1)):

Therefore, applying the chain rule, we obtain that

Jwt 1 (gt) = J (Ot 1wt 1)(`t) diag( 0(Ot 1wt 1)) Ot 1:

Using our notation we have ot = (Ot 1wt 1) and at = Ot 1wt 1, which yields

Jwt 1 (gt) = Jot (`t) diag( 0(at)) Ot 1:

Let us also denote t = Jot (`t). Then, we can further rewrite the preceding as

Jwt 1 (gt) = t;1 0(at;1) o>t 1 ; : : : ; t;kt 0(at;kt ) o>t 1 : (20.3)

It is left to calculate the vector t = Jot (`t) for every t. This is the gradient of `t at ot. We calculate this in a recursive manner. First observe that for the last layer we have that `T (u) = (u; y), where is the loss function. Since we assume that (u; y) = 12 ku yk2 we obtain that Ju(`T ) = (u y). In particular, T = JoT (`T ) = (oT y). Next, note that

`t(u) = `t+1( (Wtu)):

Therefore, by the chain rule,

Ju(`t) = J (Wtu)(`t+1)diag( 0(Wtu))Wt:

|  |  |
| --- | --- |
| 20.7 Summary | 281 |
|  |  |

In particular,

t = Jot (`t) = J (Wtot)(`t+1)diag( 0(Wtot))Wt

* 1. Jot+1 (`t+1)diag( 0(at+1))Wt
  2. t+1 diag( 0(at+1))Wt:

In summary, we can rst calculate the vectors fat; otg from the bottom of the network to its top. Then, we calculate the vectors f tg from the top of the network back to its bottom. Once we have all of these vectors, the partial derivatives are easily obtained using Equation ([20.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page280)). We have thus shown that the pseudocode of backpropagation indeed calculates the gradient.

20.7 Summary

Neural networks over graphs of size s(n) can be used to describe hypothesis

p

classes of all predictors that can be implemented in runtime of O( s(n)). We have also shown that their sample complexity depends polynomially on s(n) (speci cally, it depends on the number of edges in the network). Therefore, classes of neural network hypotheses seem to be an excellent choice. Regrettably, the problem of training the network on the basis of training data is computationally hard. We have presented the SGD framework as a heuristic approach for training neural networks and described the backpropagation algorithm which e ciently calculates the gradient of the loss function with respect to the weights over the edges.

20.8 Bibliographic Remarks

Neural networks were extensively studied in the 1980s and early 1990s, but with mixed empirical success. In recent years, a combination of algorithmic advance-ments, as well as increasing computational power and data size, has led to a breakthrough in the e ectiveness of neural networks. In particular, \deep net-works" (i.e., networks of more than 2 layers) have shown very impressive practical performance on a variety of domains. A few examples include convolutional net-works (Lecun & Bengio 1995), restricted Boltzmann machines (Hinton, Osindero & Teh 2006), auto-encoders (Ranzato, Huang, Boureau & Lecun 2007, Bengio & LeCun 2007, Collobert & Weston 2008, Lee, Grosse, Ranganath & Ng 2009, Le, Ranzato, Monga, Devin, Corrado, Chen, Dean & Ng 2012), and sum-product networks (Livni, Shalev-Shwartz & Shamir 2013, Poon & Domingos 2011). See also (Bengio 2009) and the references therein.

The expressive power of neural networks and the relation to circuit complexity have been extensively studied in (Parberry 1994). For the analysis of the sample complexity of neural networks we refer the reader to (Anthony & Bartlet 1999). Our proof technique of Theorem [20.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page274) is due to Kakade and Tewari lecture notes.

1. Neural Networks

Klivans & Sherstov (2006) have shown that for any c > 0, intersections of nc halfspaces over f 1gn are not e ciently PAC learnable, even if we allow repre-sentation independent learning. This hardness result relies on the cryptographic assumption that there is no polynomial time solution to the unique-shortest-vector problem. As we have argued, this implies that there cannot be an e cient algorithm for training neural networks, even if we allow larger networks or other activation functions that can be implemented e ciently.

The backpropagation algorithm has been introduced in Rumelhart, Hinton & Williams (1986).

20.9 Exercises

1. Neural Networks are universal approximators: Let f : [ 1; 1]n ! [ 1; 1] be a -Lipschitz function. Fix some > 0. Construct a neural net-work N : [ 1; 1]n ! [ 1; 1], with the sigmoid activation function, such that for every x 2 [ 1; 1]n it holds that jf(x) N(x)j .

Hint: Similarly to the proof of Theorem [19.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page262), partition [ 1; 1]n into small boxes. Use the Lipschitzness of f to show that it is approximately constant at each box. Finally, show that a neural network can rst decide which box the input vector belongs to, and then predict the averaged value of f at that box.

1. Prove Theorem [20.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page273).

Hint: For every f : f 1; 1gn ! f 1; 1g construct a 1-Lipschitz function

g : [ 1; 1]n ! [ 1; 1] such that if you can approximate g then you can express f.

1. Growth function of product: For i = 1; 2, let Fi be a set of functions from X to Yi. De ne H = F1 F2 to be the Cartesian product class. That is, for every f1 2 F1 and f2 2 F2, there exists h 2 H such that h(x) = (f1(x); f2(x)). Prove that H(m) F1 (m) F2 (m).
2. Growth function of composition: Let F1 be a set of functions from X to Z and let F2 be a set of functions from Z to Y. Let H = F2 F1 be the composition class. That is, for every f1 2 F1 and f2 2 F2, there exists h 2 H such that h(x) = f2(f1(x)). Prove that H(m) F2 (m) F1 (m).
3. VC of sigmoidal networks: In this exercise we show that there is a graph

(V; E) such that the VC dimension of the class of neural networks over these

graphs with the sigmoid activation function is (jEj2). Note that for every > 0, the sigmoid activation function can approximate the threshold activation function, 1[Pi xi], up to accuracy . To simplify the presentation, throughout the exercise we assume that we can exactly implement the activation function 1[Pi xi>0] using a sigmoid activation function.

Fix some n.

* 1. Construct a network, N1, with O(n) weights, which implements a function from R to f0; 1gn and satis es the following property. For every x 2 f0; 1gn,

|  |  |
| --- | --- |
| 20.9 Exercises | 283 |
|  |  |

if we feed the network with the real number 0:x1x2 : : : xn, then the output of the network will be x.

Hint: Denote = 0:x1x2 : : : xn and observe that 10k 0:5 is at least 0:5

if xk = 1 and is at most 0:3 if xk = 1.

* 1. Construct a network, N2, with O(n) weights, which implements a function from [n] to f0; 1gn such that N2(i) = ei for all i. That is, upon receiving the input i, the network outputs the vector of all zeros except 1 at the i'th neuron.
  2. Let 1; : : : ; n be n real numbers such that every i is of the form 0:a(1i)a(2i) : : : a(ni), with a(ji) 2 f0; 1g. Construct a network, N3, with O(n) weights, which im-plements a function from [n] to R, and satis es N2(i) = i for every i 2 [n].
  3. Combine N1; N3 to obtain a network that receives i 2 [n] and output a(i).
  4. Construct a network N4 that receives (i; j) 2 [n] [n] and outputs a(ji). Hint: Observe that the AND function over f0; 1g2 can be calculated using O(1) weights.
  5. Conclude that there is a graph with O(n) weights such that the VC di-mension of the resulting hypothesis class is n2.

1. Prove Theorem [20.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page276).

Hint: The proof is similar to the hardness of learning intersections of halfs-paces { see Exercise [32](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page111) in Chapter [8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page100).

Part III

Additional Learning Models

1. Online Learning

In this chapter we describe a di erent model of learning, which is called online learning. Previously, we studied the PAC learning model, in which the learner rst receives a batch of training examples, uses the training set to learn a hy-pothesis, and only when learning is completed uses the learned hypothesis for predicting the label of new examples. In our papayas learning problem, this means that we should rst buy a bunch of papayas and taste them all. Then, we use all of this information to learn a prediction rule that determines the taste of new papayas. In contrast, in online learning there is no separation between a training phase and a prediction phase. Instead, each time we buy a papaya, it is rst considered a test example since we should predict whether it is going to taste good. Then, after taking a bite from the papaya, we know the true label, and the same papaya can be used as a training example that can help us improve our prediction mechanism for future papayas.

Concretely, online learning takes place in a sequence of consecutive rounds. On each online round, the learner rst receives an instance (the learner buys a papaya and knows its shape and color, which form the instance). Then, the learner is required to predict a label (is the papaya tasty?). At the end of the round, the learner obtains the correct label (he tastes the papaya and then knows whether it is tasty or not). Finally, the learner uses this information to improve his future predictions.

To analyze online learning, we follow a similar route to our study of PAC learning. We start with online binary classi cation problems. We consider both the realizable case, in which we assume, as prior knowledge, that all the labels are generated by some hypothesis from a given hypothesis class, and the unrealizable case, which corresponds to the agnostic PAC learning model. In particular, we present an important algorithm called Weighted-Majority. Next, we study online learning problems in which the loss function is convex. Finally, we present the Perceptron algorithm as an example of the use of surrogate convex loss functions in the online learning model.

1. Online Learning

21.1 Online Classi cation in the Realizable Case

Online learning is performed in a sequence of consecutive rounds, where at round t the learner is given an instance, xt, taken from an instance domain X , and is required to provide its label. We denote the predicted label by pt. After predicting the label, the correct label, yt 2 f0; 1g, is revealed to the learner. The learner's goal is to make as few prediction mistakes as possible during this process. The learner tries to deduce information from previous rounds so as to improve its predictions on future rounds.

Clearly, learning is hopeless if there is no correlation between past and present rounds. Previously in the book, we studied the PAC model in which we assume that past and present examples are sampled i.i.d. from the same distribution source. In the online learning model we make no statistical assumptions regard-ing the origin of the sequence of examples. The sequence is allowed to be deter-ministic, stochastic, or even adversarially adaptive to the learner's own behavior (as in the case of spam e-mail ltering). Naturally, an adversary can make the number of prediction mistakes of our online learning algorithm arbitrarily large. For example, the adversary can present the same instance on each online round, wait for the learner's prediction, and provide the opposite label as the correct label.

To make nontrivial statements we must further restrict the problem. The real-izability assumption is one possible natural restriction. In the realizable case, we assume that all the labels are generated by some hypothesis, h? : X ! Y. Fur-thermore, h? is taken from a hypothesis class H, which is known to the learner. This is analogous to the PAC learning model we studied in Chapter [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page43). With this restriction on the sequence, the learner should make as few mistakes as possible, assuming that both h? and the sequence of instances can be chosen by an ad-versary. For an online learning algorithm, A, we denote by MA(H) the maximal number of mistakes A might make on a sequence of examples which is labeled by some h? 2 H. We emphasize again that both h? and the sequence of instances can be chosen by an adversary. A bound on MA(H) is called a mistake-bound and we will study how to design algorithms for which MA(H) is minimal. Formally:

definition 21.1 (Mistake Bounds, Online Learnability) Let H be a hypoth-esis class and let A be an online learning algorithm. Given any sequence S = (x1; h?(y1)); : : : ; (xT ; h?(yT )), where T is any integer and h? 2 H, let MA(S) be the number of mistakes A makes on the sequence S. We denote by MA(H) the supremum of MA(S) over all sequences of the above form. A bound of the form MA(H) B < 1 is called a mistake bound. We say that a hypothesis class H is online learnable if there exists an algorithm A for which MA(H) B < 1.

Our goal is to study which hypothesis classes are learnable in the online model, and in particular to nd good learning algorithms for a given hypothesis class.

Remark 21.1 Throughout this section and the next, we ignore the computa-

|  |  |
| --- | --- |
| 21.1 Online Classi cation in the Realizable Case | 289 |
|  |  |

tional aspect of learning, and do not restrict the algorithms to be e cient. In Section [21.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page300) and Section [21.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page301) we study e cient online learning algorithms.

To simplify the presentation, we start with the case of a nite hypothesis class, namely, jHj < 1.

In PAC learning, we identi ed ERM as a good learning algorithm, in the sense that if H is learnable then it is learnable by the rule ERMH. A natural learning rule for online learning is to use (at any online round) any ERM hypothesis, namely, any hypothesis which is consistent with all past examples.

Consistent

input: A nite hypothesis class H

initialize: V1 = H

for t = 1; 2; : : :

receive xt

choose any h 2 Vt

predict pt = h(xt)

receive true label yt = h?(xt)

update Vt+1 = fh 2 Vt : h(xt) = ytg

The Consistent algorithm maintains a set, Vt, of all the hypotheses which are consistent with (x1; y1); : : : ; (xt 1; yt 1). This set is often called the version space. It then picks any hypothesis from Vt and predicts according to this hy-pothesis.

Obviously, whenever Consistent makes a prediction mistake, at least one hypothesis is removed from Vt. Therefore, after making M mistakes we have jVtj jHj M. Since Vt is always nonempty (by the realizability assumption it contains h?) we have 1 jVtj jHj M. Rearranging, we obtain the following:

corollary 21.2 Let H be a nite hypothesis class. The Consistent algorithm enjoys the mistake bound MConsistent(H) jHj 1.

It is rather easy to construct a hypothesis class and a sequence of examples on which Consistent will indeed make jHj 1 mistakes (see Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page305)). Therefore, we present a better algorithm in which we choose h 2 Vt in a smarter way. We shall see that this algorithm is guaranteed to make exponentially fewer mistakes.

Halving

input: A nite hypothesis class H

initialize: V1 = H

for t = 1; 2; : : :

receive xt

predict pt = argmaxr2f0;1g jfh 2 Vt : h(xt) = rgj

(in case of a tie predict pt = 1)

receive true label yt = h?(xt)

update Vt+1 = fh 2 Vt : h(xt) = ytg

1. Online Learning

theorem 21.3 Let H be a nite hypothesis class. The Halving algorithm enjoys the mistake bound MHalving(H) log2(jHj).

Proof We simply note that whenever the algorithm errs we have jVt+1j jVtj=2, (hence the name Halving). Therefore, if M is the total number of mistakes, we have

1. jVT +1j jHj 2 M :

Rearranging this inequality we conclude our proof.

Of course, Halving's mistake bound is much better than Consistent's mistake bound. We already see that online learning is di erent from PAC learning|while in PAC, any ERM hypothesis is good, in online learning choosing an arbitrary ERM hypothesis is far from being optimal.

21.1.1 Online Learnability

We next take a more general approach, and aim at characterizing online learn-ability. In particular, we target the following question: What is the optimal online learning algorithm for a given hypothesis class H?

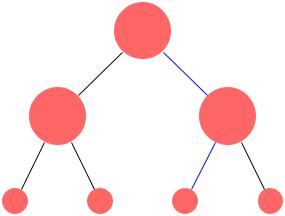
We present a dimension of hypothesis classes that characterizes the best achiev-able mistake bound. This measure was proposed by Nick Littlestone and we therefore refer to it as Ldim(H).

To motivate the de nition of Ldim it is convenient to view the online learning process as a game between two players: the learner versus the environment. On round t of the game, the environment picks an instance xt, the learner predicts a label pt 2 f0; 1g, and nally the environment outputs the true label, yt 2 f0; 1g. Suppose that the environment wants to make the learner err on the rst T rounds of the game. Then, it must output yt = 1 pt, and the only question is how it should choose the instances xt in such a way that ensures that for some h? 2 H we have yt = h?(xt) for all t 2 [T ].

A strategy for an adversarial environment can be formally described as a binary tree, as follows. Each node of the tree is associated with an instance from X . Initially, the environment presents to the learner the instance associated with the root of the tree. Then, if the learner predicts pt = 1 the environment will declare that this is a wrong prediction (i.e., yt = 0) and will traverse to the right child of the current node. If the learner predicts pt = 0 then the environment will set yt = 1 and will traverse to the left child. This process will continue and at each round, the environment will present the instance associated with the current node.

Formally, consider a complete binary tree of depth T (we de ne the depth of the tree as the number of edges in a path from the root to a leaf). We have 2T +1 1 nodes in such a tree, and we attach an instance to each node. Let v1; : : : ; v2T +1 1 be these instances. We start from the root of the tree, and set x1 = v1. At round t, we set xt = vit where it is the current node. At the end of

|  |  |
| --- | --- |
| 21.1 Online Classi cation in the Realizable Case | 291 |
|  |  |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | v1 |  |  |  |  |  |
|  |  | h1 | h2 | h3 | h4 |
|  |  |  |
|  |  |  |  |  |  |  |
| v2 | v3 | v1 | 0 | 0 | 1 | 1 |
| v2 | 0 | 1 |  |  |
|  |  |
|  |  | v3 |  |  | 0 | 1 |

Figure 21.1 An illustration of a shattered tree of depth 2. The dashed path corresponds to the sequence of examples ((v1; 1); (v3; 0)). The tree is shattered by H = fh1; h2; h3; h4g, where the predictions of each hypothesis in H on the instances v1; v2; v3 is given in the table (the '\*' mark means that hj(vi) can be either 1 or 0).

round t, we go to the left child of it if yt = 0 or to the right child if yt = 1. That

is, it+1 = 2it +yt. Unraveling the recursion we obtain it = 2t 1 +Pt 1 yj 2t 1 j.

j=1

The preceding strategy for the environment succeeds only if for every (y1; : : : ; yT ) there exists h 2 H such that yt = h(xt) for all t 2 [T ]. This leads to the following de nition.

definition 21.4 (H Shattered Tree) A shattered tree of depth d is a sequence of instances v1; : : : ; v2d 1 in X such that for every labeling (y1; : : : ; yd) 2 f0; 1gd

there exists h 2 H such that for all t 2 [d] we have h(vit ) = yt where it = 2t 1 + Pt 1 y 2t 1 j.

j=1 j

An illustration of a shattered tree of depth 2 is given in Figure [21.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page291).

definition 21.5 (Littlestone's Dimension (Ldim)) Ldim(H) is the maximal integer T such that there exists a shattered tree of depth T , which is shattered by H.

The de nition of Ldim and the discussion above immediately imply the fol-lowing:

lemma 21.6 No algorithm can have a mistake bound strictly smaller than Ldim(H); namely, for every algorithm, A, we have MA(H) Ldim(H).

Proof Let T = Ldim(H) and let v1; : : : ; v2T 1 be a sequence that satis es the requirements in the de nition of Ldim. If the environment sets xt = vit and yt = 1 pt for all t 2 [T ], then the learner makes T mistakes while the de nition of Ldim implies that there exists a hypothesis h 2 H such that yt = h(xt) for all t. 

Let us now give several examples.

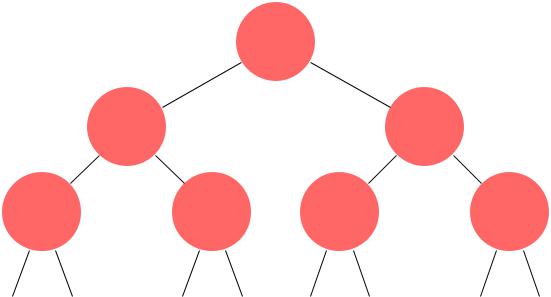
Example 21.2 Let H be a nite hypothesis class. Clearly, any tree that is shat-tered by H has depth of at most log2(jHj). Therefore, Ldim(H) log2(jHj). Another way to conclude this inequality is by combining Lemma [21.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page291) with The-orem [21.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page290).

Example 21.3 Let X = f1; : : : ; dg and H = fh1; : : : ; hdg where hj(x) = 1 i

1. Online Learning

x = j. Then, it is easy to show that Ldim(H) = 1 while jHj = d can be arbitrarily large. Therefore, this example shows that Ldim(H) can be signi cantly smaller than log2(jHj).

Example 21.4 Let X = [0; 1] and H = fx 7!1[x<a] : a 2 [0; 1]g; namely, H is the class of thresholds on the interval [0; 1]. Then, Ldim(H) = 1. To see this, consider the tree



1=2

1=4 3=4

1=8 3=8 5=8 7=8

This tree is shattered by H. And, because of the density of the reals, this tree can be made arbitrarily deep.

Lemma [21.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page291) states that Ldim(H) lower bounds the mistake bound of any algorithm. Interestingly, there is a standard algorithm whose mistake bound matches this lower bound. The algorithm is similar to the Halving algorithm. Recall that the prediction of Halving is made according to a majority vote of the hypotheses which are consistent with previous examples. We denoted this set by Vt. Put another way, Halving partitions Vt into two sets: Vt+ = fh 2 Vt : h(xt) = 1g and Vt = fh 2 Vt : h(xt) = 0g. It then predicts according to the larger of the two groups. The rationale behind this prediction is that whenever Halving makes a mistake it ends up with jVt+1j 0:5 jVtj.

The optimal algorithm we present in the following uses the same idea, but instead of predicting according to the larger class, it predicts according to the class with larger Ldim.

Standard Optimal Algorithm (SOA)

input: A hypothesis class H

initialize: V1 = H

for t = 1; 2; : : :

receive xt

(r)

for r 2 f0; 1g let Vt = fh 2 Vt : h(xt) = rg

(r)

predict pt = argmaxr2f0;1g Ldim(Vt )

(in case of a tie predict pt = 1)

receive true label yt

update Vt+1 = fh 2 Vt : h(xt) = ytg

The following lemma formally establishes the optimality of the preceding al-gorithm.

|  |  |
| --- | --- |
| 21.1 Online Classi cation in the Realizable Case | 293 |
|  |  |

lemma 21.7 SOA enjoys the mistake bound MSOA(H) Ldim(H).

Proof It su ces to prove that whenever the algorithm makes a prediction mis-take we have Ldim(Vt+1) Ldim(Vt) 1. We prove this claim by assuming the contrary, that is, Ldim(Vt+1) = Ldim(Vt). If this holds true, then the de nition of pt implies that Ldim(Vt(r)) = Ldim(Vt) for both r = 1 and r = 0. But, then we can construct a shaterred tree of depth Ldim(Vt) + 1 for the class Vt, which leads to the desired contradiction. 

Combining Lemma [21.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page292) and Lemma [21.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page291) we obtain:

corollary 21.8 Let H be any hypothesis class. Then, the standard optimal algorithm enjoys the mistake bound MSOA(H) = Ldim(H) and no other algorithm can have MA(H) < Ldim(H).

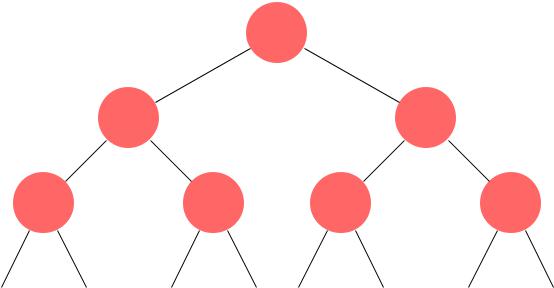
Comparison to VC Dimension

In the PAC learning model, learnability is characterized by the VC dimension of the class H. Recall that the VC dimension of a class H is the maximal number

1. such that there are instances x1; : : : ; xd that are shattered by H. That is, for any sequence of labels (y1; : : : ; yd) 2 f0; 1gd there exists a hypothesis h 2 H that gives exactly this sequence of labels. The following theorem relates the VC dimension to the Littlestone dimension.

theorem 21.9 For any class H, VCdim(H) Ldim(H), and there are classes for which strict inequality holds. Furthermore, the gap can be arbitrarily larger.

Proof We rst prove that VCdim(H) Ldim(H). Suppose VCdim(H) = d and let x1; : : : ; xd be a shattered set. We now construct a complete binary tree of instances v1; : : : ; v2d 1, where all nodes at depth i are set to be xi { see the following illustration:



x1

x2 x2

x3 x3 x3 x3

Now, the de nition of a shattered set clearly implies that we got a valid shattered tree of depth d, and we conclude that VCdim(H) Ldim(H). To show that the gap can be arbitrarily large simply note that the class given in Example [21.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page292) has VC dimension of 1 whereas its Littlestone dimension is in nite. 

1. Online Learning

21.2 Online Classi cation in the Unrealizable Case

In the previous section we studied online learnability in the realizable case. We now consider the unrealizable case. Similarly to the agnostic PAC model, we no longer assume that all labels are generated by some h? 2 H, but we require the learner to be competitive with the best xed predictor from H. This is captured by the regret of the algorithm, which measures how \sorry" the learner is, in retrospect, not to have followed the predictions of some hypothesis h 2 H. Formally, the regret of an algorithm A relative to h when running on a sequence of T examples is de ned as

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RegretA | (h; T ) = (x1;y1);:::;(xT ;yT ) | "t=1 jpt ytj | =1 jh(xt) ytj# | ; |  |
|  |  | T | T |  |  |
|  | sup | X | Xt |  | (21.1) |
|  |  |  |  |
| and the regret of the algorithm relative to a hypothesis class H is | | | |  |  |
|  | RegretA(H; T ) = sup RegretA(h; T ): | | |  | (21.2) |
|  |  | h2H |  |  |  |

We restate the learner's goal as having the lowest possible regret relative to H. An interesting question is whether we can derive an algorithm with low regret, meaning that RegretA(H; T ) grows sublinearly with the number of rounds, T , which implies that the di erence between the error rate of the learner and the best hypothesis in H tends to zero as T goes to in nity.

We rst show that this is an impossible mission|no algorithm can obtain a sublinear regret bound even if jHj = 2. Indeed, consider H = fh0; h1g, where h0 is the function that always returns 0 and h1 is the function that always returns

1. An adversary can make the number of mistakes of any online algorithm be equal to T , by simply waiting for the learner's prediction and then providing

the opposite label as the true label. In contrast, for any sequence of true labels, y1; : : : ; yT , let b be the majority of labels in y1; : : : ; yT , then the number of mistakes of hb is at most T =2. Therefore, the regret of any online algorithm

might be at least T T =2 = T =2, which is not sublinear in T . This impossibility result is attributed to Cover (Cover 1965).

To sidestep Cover's impossibility result, we must further restrict the power of the adversarial environment. We do so by allowing the learner to randomize his predictions. Of course, this by itself does not circumvent Cover's impossibil-ity result, since in deriving this result we assumed nothing about the learner's

strategy. To make the randomization meaningful, we force the adversarial envir-onment to decide on yt without knowing the random coins ipped by the learner on round t. The adversary can still know the learner's forecasting strategy and even the random coin ips of previous rounds, but it does not know the actual value of the random coin ips used by the learner on round t. With this (mild) change of game, we analyze the expected number of mistakes of the algorithm,

where the expectation is with respect to the learner's own randomization. That is, if the learner outputs y^t where P[^yt = 1] = pt, then the expected loss he pays

|  |  |
| --- | --- |
| 21.2 Online Classi cation in the Unrealizable Case | 295 |
|  |  |

on round t is

P[^yt 6= yt] = jpt ytj:

Put another way, instead of having the predictions of the learner being in f0; 1g we allow them to be in [0; 1], and interpret pt 2 [0; 1] as the probability to predict the label 1 on round t.

With this assumption it is possible to derive a low regret algorithm. In partic-ular, we will prove the following theorem.

theorem 21.10 For every hypothesis class H, there exists an algorithm for online classi cation, whose predictions come from [0; 1], that enjoys the regret bound

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T | T |  |  |  |
| Xt | X | p |  |  |
| 8h 2 H; | jpt ytj | jh(xt) ytj 2 minflog(jHj) ; Ldim(H) log(eT )g T : | | |
| =1 | t=1 |  |  |  |

Furthermore, no algorithm can achieve an expected regret bound smaller than

p

Ldim(H) T .

We will provide a constructive proof of the upper bound part of the preceding theorem. The proof of the lower bound part can be found in (Ben-David, Pal, & Shalev-Shwartz 2009).

The proof of Theorem [21.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page295) relies on the Weighted-Majority algorithm for learning with expert advice. This algorithm is important by itself and we dedicate the next subsection to it.

21.2.1 Weighted-Majority

Weighted-majority is an algorithm for the problem of prediction with expert ad-vice. In this online learning problem, on round t the learner has to choose the advice of d given experts. We also allow the learner to randomize his choice by de ning a distribution over the d experts, that is, picking a vector w(t) 2 [0; 1]d, with Pi wi(t) = 1, and choosing the ith expert with probability wi(t). After the learner chooses an expert, it receives a vector of costs, vt 2 [0; 1]d, where vt;i is the cost of following the advice of the ith expert. If the learner's predic-tions are randomized, then its loss is de ned to be the averaged cost, namely, Pi wi(t)vt;i = hw(t); vti. The algorithm assumes that the number of rounds T is given. In Exercise [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page305) we show how to get rid of this dependence using the doubling trick.

1. Online Learning

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Weighted-Majority | | | |  |  |
| input: number of experts, d ; | | | | | number of rounds, T | | |
|  | |  |  | |  |  |  |
| parameter: = | |  | 2 log(d)=T | |  |  |  |
| ~ (1) |  | (1; : : : ; 1) | |  |  |  |  |
| initialize: w | =p | | |  |  |  |  |
| for t = 1; 2; : : : |  |  |  |  |  |  |  |
| set w(t) = w~(t)=Zt where Zt | | | | | = i w~i(t) |  | [i] = w(t) |
|  |  |  |  | according to | | P |
| choose expert i at random | | | |  | P | i |

receive costs of all experts vt 2 [0; 1]d

pay cost hw(t); vti

update rule 8i; w~i(t+1) = w~i(t)e vt;i

The following theorem is key for analyzing the regret bound of Weighted-Majority.

theorem 21.11 Assuming that T > 2 log(d), the Weighted-Majority algo-rithm enjoys the bound

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | T |  |  |  | T |  |  |  |  |  |  |  |  |
|  | Xt | | (t) | 2 | | X |  |  |  | p |  |  |  |  |
|  |  |  | min | |  | v |  |  |  | 2 log(d) T : | | | |
|  |  | =1hw ; vti | |  | t;i |  |  |
|  |  | i [d] t=1 | | |  |  |  |  |  |  |
| Proof We have: |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | Zt+1 | = log Xi |  | w~(t) | e vt;i | | | = log Xi | | | (t) | e vt;i : | |
| log |  |  |  | i | wi |
|  | Zt | Zt |

Using the inequality e a 1 a + a2=2, which holds for all a 2 (0; 1), and the fact that Pi wi(t) = 1, we obtain

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Zt+1 | log Xi | (t) |  | 1 vt;i + 2vt;i2=2 |  |
| log |  | wi |
| Zt |

1. log 1 X wi(t) vt;i 2vt;i2=2 :

i

| {z }

def

= b

Next, note that b 2 (0; 1). Therefore, taking log of the two sides of the inequality 1 b e b we obtain the inequality log(1 b) b, which holds for all b 1, and obtain

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Zt+1 | Xi | (t) | vt;i 2vt;i2=2 |  |
| log |  | wi |
| Zt |

1. hw(t); vti + 2 X wi(t)vt;i2=2 i

hw(t); vti + 2=2:

|  |  |
| --- | --- |
| 21.2 Online Classi cation in the Unrealizable Case | 297 |
|  |  |

Summing this inequality over t we get

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| T | Zt+1 |  | T | T 2 |  |  |
| Xt |  | X |  |
| Zt | 2 : | | (21.3) |
| log(ZT +1) log(Z1) = log | hw(t); vti + |
| =1 |  |  | t=1 |  |  |  |

Next, we lower bound ZT +1. For each i, we can rewrite w~i(T +1) = e Pt vt;i and we get that

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| log ZT +1 = log | e | Pt vt;i | ! log maxi e P | t vt;i | = mini | vt;i: |
| i |  |  |  |  |  | t |
| X |  |  |  |  |  | X |

Combining the preceding with Equation ([21.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page297)) and using the fact that log(Z1) = log(d) we get that

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | T | |  |  | T 2 | | |
| mini | X |  | Xt | |  |  |
|  |  |  |  |  |
| vt;i log(d) | =1 | hw(t); vti + 2 ; | | | | | |
|  | t |  |  |  |  |  |  |  |
| which can be rearranged as follows: | |  |  |  |  |  |  |  |  |
| T |  |  | log(d) | | T | | | | |
| Xt | hw(t); vti mini | X |
|  |  |  |  |  |  |  |
|  | | + 2 : | | | |  |
| =1 | vt;i |  |
|  | t |  |  |  |  |  |  |  |

Plugging the value of into the equation concludes our proof.

Proof of Theorem [21.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page295)

Equipped with the Weighted-Majority algorithm and Theorem [21.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page296), we are ready to prove Theorem [21.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page295). We start with the simpler case, in which H is

1. nite class, and let us write H = fh1; : : : ; hdg. In this case, we can refer to each hypothesis, hi, as an expert, whose advice is to predict hi(xt), and whose

cost is vt;i = jhi(xt) ytj. The prediction of the algorithm will therefore be pt = Pi wi(t)hi(xt) 2 [0; 1], and the loss is

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| jpt ytj = |  | wi(t)hi(xt) yt = |  | wi(t)(hi(xt) yt) : |
|  | d |  | d |  |
|  | i=1 | i=1 |
|  | X |  | X |  |
|  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Now, if yt = 1, then for all i, hi(xt) yt 0. Therefore, the above equals to Pi wi(t)jhi(xt) ytj. If yt = 0 then for all i, hi(xt) yt 0, and the above also equals Pi wi(t)jhi(xt) ytj. All in all, we have shown that

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | d |
|  | jpt ytj = | | wi(t)jhi(xt) ytj = hw(t); vti: |
|  |  |  | Xi |
|  |  |  | =1 |
| Furthermore, for each i, | | t vt;i is exactly the number of mistakes hypothesis hi | |
| makes. Applying | Theorem [21.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page296) we obtain | | |
|  | P |  |

1. Online Learning

corollary 21.12 Let H be a nite hypothesis class. There exists an algorithm for online classi cation, whose predictions come from [0; 1], that enjoys the regret

bound

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T |  |  | T |  |  |  |  |  |  |  |  |
| X |  |  | Xt |  |  |  |  |  | p |  |  |
| jpt ytj h |  | t |  |  |  | tj | jHj | |
|  |  | j | ) | y |  |
|  | min | |  | h(x |  |  |  | 2 log( ) T : | |
| t=1 |  | 2H | =1 |  |  |  |  |  |  |  |  |

Next, we consider the case of a general hypothesis class. Previously, we con-structed an expert for each individual hypothesis. However, if H is in nite this leads to a vacuous bound. The main idea is to construct a set of experts in a more sophisticated way. The challenge is how to de ne a set of experts that, on one hand, is not excessively large and, on the other hand, contains experts that give accurate predictions.

We construct the set of experts so that for each hypothesis h 2 H and every sequence of instances, x1; x2; : : : ; xT , there exists at least one expert in the set which behaves exactly as h on these instances. For each L Ldim(H) and each sequence 1 i1 < i2 < < iL T we de ne an expert. The expert simulates the game between SOA (presented in the previous section) and the environment on the sequence of instances x1; x2; : : : ; xT assuming that SOA makes a mistake precisely in rounds i1; i2; : : : ; iL. The expert is de ned by the following algorithm.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Expert(i1; i2; : : : ; iL) | | | | | |
| input A hypothesis class H ; | | | | | | Indices i1 < i2 < < iL |
| initialize: V1 = H | |  |  |  |  |  |
| for | t = 1; 2; : : : ; T |  |  |  |  |  |
| receive xt | |  |  |  |  |  |
| for r 2 f0; 1g let Vt(r) = fh 2 Vt : h(xt) = rg | | | | | | |
| de ne y~ = argmax | | r | Ldim | V (r) | | |
|  | t |  | 0) | t |  |
|  | (in case of a tie set y~t = | | |
| if | t 2 fi1; i2; : : : ; iLg | | |  |  |  |

predict y^t = 1 y~t

else

predict y^t = y~t

update Vt+1 = Vt(^yt)

Note that each such expert can give us predictions at every round t while only observing the instances x1; : : : ; xt. Our generic online learning algorithm is now an application of the Weighted-Majority algorithm with these experts.

To analyze the algorithm we rst note that the number of experts is

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| d = | Ldim(H) |  | T | : | (21.4) |
| L=0 | L |
|  | X |  |  |  |  |

It can be shown that when T Ldim(H) + 2, the right-hand side of the equation is bounded by (eT =Ldim(H))Ldim(H) (the proof can be found in Lemma [A.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page420)).

|  |  |
| --- | --- |
| 21.2 Online Classi cation in the Unrealizable Case | 299 |
|  |  |

Theorem [21.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page296) tells us that the expected number of mistakes of Weighted-Majority

p

is at most the number of mistakes of the best expert plus 2 log(d) T . We will next show that the number of mistakes of the best expert is at most the number of mistakes of the best hypothesis in H. The following key lemma shows that, on any sequence of instances, for each hypothesis h 2 H there exists an expert with the same behavior.

lemma 21.13 Let H be any hypothesis class with Ldim(H) < 1. Let x1; x2; : : : ; xT be any sequence of instances. For any h 2 H, there exists L Ldim(H) and in-dices 1 i1 < i2 < < iL T such that when running Expert(i1; i2; : : : ; iL) on the sequence x1; x2; : : : ; xT , the expert predicts h(xt) on each online round t = 1; 2; : : : ; T .

Proof Fix h 2 H and the sequence x1; x2; : : : ; xT . We must construct L and the

indices i1; i2; : : : ; iL. Consider running SOA on the input (x1; h(x1)), (x2; h(x2)),

1. : :, (xT ; h(xT )). SOA makes at most Ldim(H) mistakes on such input. We de ne L to be the number of mistakes made by SOA and we de ne fi1; i2; : : : ; iLg to be the set of rounds in which SOA made the mistakes.

Now, consider the Expert(i1; i2; : : : ; iL) running on the sequence x1; x2; : : : ; xT . By construction, the set Vt maintained by Expert(i1; i2; : : : ; iL) equals the set Vt maintained by SOA when running on the sequence (x1; h(x1)); : : : ; (xT ; h(xT )).

The predictions of SOA di er from the predictions of h if and only if the round is in fi1; i2; : : : ; iLg. Since Expert(i1; i2; : : : ; iL) predicts exactly like SOA if t is not in fi1; i2; : : : ; iLg and the opposite of SOAs' predictions if t is in fi1; i2; : : : ; iLg, we conclude that the predictions of the expert are always the same as the pre-

dictions of h.

The previous lemma holds in particular for the hypothesis in H that makes the least number of mistakes on the sequence of examples, and we therefore obtain the following:

corollary 21.14 Let (x1; y1); (x2; y2); : : : ; (xT ; yT ) be a sequence of examples and let H be a hypothesis class with Ldim(H) < 1. There exists L Ldim(H) and indices 1 i1 < i2 < < iL T , such that Expert(i1; i2; : : : ; iL) makes at most as many mistakes as the best h 2 H does, namely,

T

X

min jh(xt) ytj

h2H

t=1

mistakes on the sequence of examples.

Together with Theorem [21.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page296), the upper bound part of Theorem [21.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page295) is proven.

1. Online Learning

21.3 Online Convex Optimization

In Chapter [12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page156) we studied convex learning problems and showed learnability results for these problems in the agnostic PAC learning framework. In this section we show that similar learnability results hold for convex problems in the online learning framework. In particular, we consider the following problem.

Online Convex Optimization

de nitions:

hypothesis class H ; domain Z ; loss function ` : H Z ! R assumptions:

H is convex

8z 2 Z, `( ; z) is a convex function

for t = 1; 2; : : : ; T

learner predicts a vector w(t) 2 H

environment responds with zt 2 Z

learner su ers loss `(w(t); zt)

As in the online classi cation problem, we analyze the regret of the algorithm. Recall that the regret of an online algorithm with respect to a competing hy-pothesis, which here will be some vector w? 2 H, is de ned as

|  |  |  |
| --- | --- | --- |
| T | T |  |
| Xt | X |  |
| RegretA(w?; T ) = | `(w(t); zt)`(w?; zt): | (21.5) |
| =1 | t=1 |  |

As before, the regret of the algorithm relative to a set of competing vectors, H, is de ned as

RegretA(H; T ) = sup RegretA(w?; T ):

w?2H

In Chapter [14](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page184) we have shown that Stochastic Gradient Descent solves convex learning problems in the agnostic PAC model. We now show that a very similar algorithm, Online Gradient Descent, solves online convex learning problems.

Online Gradient Descent

parameter: > 0

initialize: w(1) = 0

for t = 1; 2; : : : ; T

predict w(t)

receive zt and let ft( ) = `( ; zt)

choose vt 2 @ft(w(t))

update:

1. w(t+ 12 ) = w(t) vt
2. w(t+1) = argminw2H kw w(t+ 12 )k

|  |  |
| --- | --- |
| 21.4 The Online Perceptron Algorithm | 301 |
|  |  |

theorem 21.15 The Online Gradient Descent algorithm enjoys the following regret bound for every w? 2 H,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Regret | (w?; T ) | |  | kw?k2 | | | + | |  |  | |  | T | |  | v |  |  | 2: |  |  |  |  |
|  |  |  |  |  |  |  | Xt | | | k | tk | |  |  |  |
| A |  |  |  | 2 | |  |  |  | 2 | |  |  |  |  |  |  |
|  |  |  |  |  |  | =1 | | |  |  |  |  |  |  |
|  | is -Lipschitz for all t, then setting = 1=p | | | | | | | | | | | | | | | | | | | | |  | yields |
| If we further assume that ft | T |
|  | ? |  |  | 1 | |  | ? |  | 2 | |  | 2 | | | p | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| RegretA(w | | ; T ) | | |  | (kw | | k |  |  | + | | | | ) |  | T : | | | |  |  |  |
| 2 |  |  |  |  |  |  |
| If we further assume that H is B-bounded and we set = | | | | | | | | | | | | | | | | | | | B | | then | | |
| p |  |
| T |
| RegretA(H; T ) B | | | | | | | | | |  | p | |  |  |  |  |  |  |  |  |  |  |  |
|  | T : | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

Proof The analysis is similar to the analysis of Stochastic Gradient Descent with projections. Using the projection lemma, the de nition of w(t+ 12 ), and the de nition of subgradients, we have that for every t,

kw(t+1) w?k2 kw(t) w?k2

1. kw(t+1) w?k2 kw(t+ 12 ) w?k2 + kw(t+ 12 ) w?k2 kw(t) w?k2 kw(t+ 12 ) w?k2 kw(t) w?k2
2. kw(t) vt w?k2 kw(t) w?k2
3. 2 hw(t) w?; vti + 2kvtk2

2 (ft(w(t)) ft(w?)) + 2kvtk2:

Summing over t and observing that the left-hand side is a telescopic sum we obtain that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  | T |  |  |  | T | |  |
| kw(T +1) w?k2 kw(1) w?k2 2 | | | | | | | | X |  |  |  | Xt | | |
|  | (ft(w(t)) ft(w?)) + 2 | | | |  | kvtk2: |
|  |  |  |  |  |  |  |  | t=1 |  |  | =1 | | | |
| Rearranging the inequality and using the fact that w(1) = 0, we get that | | | | | | | | | | | | | | |
| T | (f (w(t)) | f (w?)) |  | kw(1) w?k2 kw(T +1) w?k2 | | | | | | + | T | |  | v 2 |
| X |  | |  |  | | |  | Xt |  |
| t | t |  | |  |  |  |  | 2 |  |  | k tk | |
| t=1 |  |  |  |  |  | 2 =1 | |
|  |  |  |  | kw?k2 | + |  |  | T | v 2: |  |  |  |  |  |
|  |  |  |  |  | Xt |  |  |  |  |  |
|  |  |  | 2 | | 2 | |  | k tk |  |  |  |  |  |
|  |  |  |  | =1 |  |  |  |  |  |

This proves the rst bound in the theorem. The second bound follows from the assumption that ft is -Lipschitz, which implies that kvtk . 

21.4 The Online Perceptron Algorithm

The Perceptron is a classic online learning algorithm for binary classi cation with the hypothesis class of homogenous halfspaces, namely, H = fx 7!sign(hw; xi) :

1. Online Learning

w 2 Rdg. In Section [9.1.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page120) we have presented the batch version of the Perceptron, which aims to solve the ERM problem with respect to H. We now present an online version of the Perceptron algorithm.

Let X = Rd, Y = f 1; 1g. On round t, the learner receives a vector xt 2 Rd. The learner maintains a weight vector w(t) 2 Rd and predicts pt = sign(hw(t); xti). Then, it receives yt 2 Y and pays 1 if pt 6= yt and 0 otherwise.

The goal of the learner is to make as few prediction mistakes as possible. In Section [21.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page288) we characterized the optimal algorithm and showed that the best achievable mistake bound depends on the Littlestone dimension of the class.

We show later that if d 2 then Ldim(H) = 1, which implies that we have no hope of making few prediction mistakes. Indeed, consider the tree for which v1 = ( 12 ; 1; 0; : : : ; 0), v2 = ( 14 ; 1; 0; : : : ; 0), v3 = ( 34 ; 1; 0; : : : ; 0), etc. Because of the density of the reals, this tree is shattered by the subset of H which contains all hypotheses that are parametrized by w of the form w = ( 1; a; 0; : : : ; 0), for a 2 [0; 1]. We conclude that indeed Ldim(H) = 1.

To sidestep this impossibility result, the Perceptron algorithm relies on the technique of surrogate convex losses (see Section [12.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page167)). This is also closely related to the notion of margin we studied in Chapter [15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page202).

A weight vector w makes a mistake on an example (x; y) whenever the sign of hw; xi does not equal y. Therefore, we can write the 0 1 loss function as follows

`(w; (x; y)) = 1[yhw;xi 0]:

On rounds on which the algorithm makes a prediction mistake, we shall use the hinge-loss as a surrogate convex loss function

ft(w) = maxf0; 1 ythw; xtig:

The hinge-loss satis es the two conditions:

ft is a convex function

For all w, ft(w) `(w; (xt; yt)). In particular, this holds for w(t).

On rounds on which the algorithm is correct, we shall de ne ft(w) = 0. Clearly, ft is convex in this case as well. Furthermore, ft(w(t)) = `(w(t); (xt; yt)) = 0.

Remark 21.5 In Section [12.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page167) we used the same surrogate loss function for all the examples. In the online model, we allow the surrogate to depend on the speci c round. It can even depend on w(t). Our ability to use a round speci c surrogate stems from the worst-case type of analysis we employ in online learning.

Let us now run the Online Gradient Descent algorithm on the sequence of functions, f1; : : : ; fT , with the hypothesis class being all vectors in Rd (hence, the projection step is vacuous). Recall that the algorithm initializes w(1) = 0 and its update rule is

w(t+1) = w(t) vt

for some vt 2 @ft(w(t)). In our case, if ythw(t); xti > 0 then ft is the zero

|  |  |
| --- | --- |
| 21.4 The Online Perceptron Algorithm | 303 |
|  |  |

function and we can take vt = 0. Otherwise, it is easy to verify that vt = ytxt is in @ft(w(t)). We therefore obtain the update rule

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| w |  | = | (w(t) | + ytxt | otherwise |
|  | (t+1) |  | w(t) |  | if ythw(t); xti > 0 |

Denote by M the set of rounds in which sign(hw(t); xti) 6= yt. Note that on round t, the prediction of the Perceptron can be rewritten as

|  |  |  |
| --- | --- | --- |
| pt = sign(hw(t); xti) = sign | yi hxi; xti! | : |
|  | X |  |
|  | i2M:i<t |  |

This form implies that the predictions of the Perceptron algorithm and the set M do not depend on the actual value of as long as > 0. We have therefore obtained the Perceptron algorithm:

Perceptron

initialize: w1 = 0

for t = 1; 2; : : : ; T

receive xt

predict pt = sign(hw(t); xti)

if ythw(t); xti 0

w(t+1) = w(t) + ytxt

else

w(t+1) = w(t)

To analyze the Perceptron, we rely on the analysis of Online Gradient De-scent given in the previous section. In our case, the subgradient of ft we use in the Perceptron is vt = 1[ythw(t);xti 0] yt xt. Indeed, the Perceptron's update is w(t+1) = w(t) vt, and as discussed before this is equivalent to w(t+1) = w(t) vt for every > 0. Therefore, Theorem [21.15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page300) tells us that

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| T | T | 1 | |  | | T |  |
| Xt | X |  |  |  |  | X |  |
|  | ft(w(t))ft(w?) |  |  | kw?k22 + |  |  | kvtk22: |
| =1 | 2 | 2 | t=1 |
| t=1 |  |  |  |  |  |
| Since ft(w(t)) is a surrogate for the 0 1 loss we know that | | | | | | | T |
| t=1 ft(w(t)) jMj. |
| Denote R = maxt kxtk; then we obtain | |  |  |  |  | P | |

T

jMj X ft(w?) 21 kw?k22 + 2 jMj R2

t=1

Setting = kpw?k and rearranging, we obtain

1. jMj

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | T |  |
| p |  |  | Xt |  |
| jMj Rkw?k jMj | | | ft(w?) 0: | (21.6) |
|  |  |  | =1 |  |

This inequality implies

1. Online Learning

theorem 21.16 Suppose that the Perceptron algorithm runs on a sequence (x1; y1); : : : ; (xT ; yT ) and let R = maxt kxtk. Let M be the rounds on which the Perceptron errs and let ft(w) = 1[t2M] [1 ythw; xti]+. Then, for every w?

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| jMj | t | ft(w?) + R kw?k | s |  |  |  |
| t | ft(w?) + R2 kw?k2 : | |
|  | X |  |  | X |  |  |

In particular, if there exists w? such that ythw?; xti 1 for all t then jMj R2 kw?k2:

Proof The theorem follows from Equation ([21.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page303)) and the following claim: Given x; b; c 2 R+, the inequality x b px c 0 implies that x c + b2 + b pc. The last claim can be easily derived by analyzing the roots of the convex parabola Q(y) = y2 by c. 

The last assumption of Theorem [21.16](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page303) is called separability with large margin (see Chapter [15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page202)). That is, there exists w? that not only satis es that the point xt lies on the correct side of the halfspace, it also guarantees that xt is not too close to the decision boundary. More speci cally, the distance from xt to the decision boundary is at least = 1=kw?k and the bound becomes (R= )2.

When the separability assumption does not hold, the bound involves the term [1 ythw?; xti]+ which measures how much the separability with margin require-ment is violated.

As a last remark we note that there can be cases in which there exists some w? that makes zero errors on the sequence but the Perceptron will make many errors. Indeed, this is a direct consequence of the fact that Ldim(H) = 1. The way we sidestep this impossibility result is by assuming more on the sequence of examples { the bound in Theorem [21.16](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page303) will be meaningful only if the cumulative surrogate loss, Pt ft(w?) is not excessively large.

21.5 Summary

In this chapter we have studied the online learning model. Many of the results we derived for the PAC learning model have an analog in the online model. First, we have shown that a combinatorial dimension, the Littlestone dimension, char-acterizes online learnability. To show this, we introduced the SOA algorithm (for the realizable case) and the Weighted-Majority algorithm (for the unrealizable case). We have also studied online convex optimization and have shown that online gradient descent is a successful online learner whenever the loss function is convex and Lipschitz. Finally, we presented the online Perceptron algorithm as a combination of online gradient descent and the concept of surrogate convex loss functions.

|  |  |
| --- | --- |
| 21.6 Bibliographic Remarks | 305 |
|  |  |

21.6 Bibliographic Remarks

The Standard Optimal Algorithm was derived by the seminal work of Lit-tlestone (1988). A generalization to the nonrealizable case, as well as other variants like margin-based Littlestone's dimension, were derived in (Ben-David et al. 2009). Characterizations of online learnability beyond classi cation have been obtained in (Abernethy, Bartlett, Rakhlin & Tewari 2008, Rakhlin, Srid-haran & Tewari 2010, Daniely et al. 2011). The Weighted-Majority algorithm is due to (Littlestone & Warmuth 1994) and (Vovk 1990).

The term \online convex programming" was introduced by Zinkevich (2003) but this setting was introduced some years earlier by Gordon (1999). The Per-ceptron dates back to Rosenblatt (Rosenblatt 1958). An analysis for the re-alizable case (with margin assumptions) appears in (Agmon 1954, Minsky & Papert 1969). Freund and Schapire (Freund & Schapire 1999) presented an anal-ysis for the unrealizable case with a squared-hinge-loss based on a reduction to the realizable case. A direct analysis for the unrealizable case with the hinge-loss was given by Gentile (Gentile 2003).

For additional information we refer the reader to Cesa-Bianchi & Lugosi (2006) and Shalev-Shwartz (2011).

21.7 Exercises

1. Find a hypothesis class H and a sequence of examples on which Consistent makes jHj 1 mistakes.
2. Find a hypothesis class H and a sequence of examples on which the mistake bound of the Halving algorithm is tight.
3. Let d 2, X = f1; : : : ; dg and let H = fhj : j 2 [d]g, where hj(x) = 1[x=j]. Calculate MHalving(H) (i.e., derive lower and upper bounds on MHalving(H), and prove that they are equal).
4. The Doubling Trick:

In Theorem [21.15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page300), the parameter depends on the time horizon T . In this

exercise we show how to get rid of this dependence by a simple trick. p

Consider an algorithm that enjoys a regret bound of the form T , but its parameters require the knowledge of T . The doubling trick, described in the following, enables us to convert such an algorithm into an algorithm that does not need to know the time horizon. The idea is to divide the time into periods of increasing size and run the original algorithm on each period.

The Doubling Trick

input: algorithm A whose parameters depend on the time horizon

for m = 0; 1; 2; : : :

run A on the 2m rounds t = 2m; : : : ; 2m+1 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 306 | Online Learning | | | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  | |  |  |  |  |
|  | Show that if the regret of A on each period of 2m rounds is at most p | | | | | | | | | |  |  |
|  | 2m, | |
|  | then the total regret is at most | | | | | | |  |  |  |  |  |
|  |  |  | p | |  |  |  | p |  |  |  |  |
|  |  | p |  | 2 | | |  |  | : |  |  |
|  |  |  |  | T |  |  |
|  |  | 2 1 | | | | | |  |  |  |  |  |

1. Online-to-batch Conversions: In this exercise we demonstrate how a suc-cessful online learning algorithm can be used to derive a successful PAC learner as well.

Consider a PAC learning problem for binary classi cation parameterized

by an instance domain, X , and a hypothesis class, H. Suppose that there exists an online learning algorithm, A, which enjoys a mistake bound MA(H) < 1. Consider running this algorithm on a sequence of T examples which are sam-

pled i.i.d. from a distribution D over the instance space X , and are labeled by some h? 2 H. Suppose that for every round t, the prediction of the algorithm is based on a hypothesis ht : X ! f0; 1g. Show that

E[LD(hr)] MA(H) ;

T

where the expectation is over the random choice of the instances as well as a random choice of r according to the uniform distribution over [T ].

Hint: Use similar arguments to the ones appearing in the proof of Theo-rem [14.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page192).

1. Clustering

Clustering is one of the most widely used techniques for exploratory data anal-ysis. Across all disciplines, from social sciences to biology to computer science, people try to get a rst intuition about their data by identifying meaningful groups among the data points. For example, computational biologists cluster genes on the basis of similarities in their expression in di erent experiments; re-tailers cluster customers, on the basis of their customer pro les, for the purpose of targeted marketing; and astronomers cluster stars on the basis of their spacial proximity.

The rst point that one should clarify is, naturally, what is clustering? In-tuitively, clustering is the task of grouping a set of objects such that similar objects end up in the same group and dissimilar objects are separated into dif-ferent groups. Clearly, this description is quite imprecise and possibly ambiguous. Quite surprisingly, it is not at all clear how to come up with a more rigorous de nition.

There are several sources for this di culty. One basic problem is that the two objectives mentioned in the earlier statement may in many cases contradict each other. Mathematically speaking, similarity (or proximity) is not a transi-tive relation, while cluster sharing is an equivalence relation and, in particular, it is a transitive relation. More concretely, it may be the case that there is a long sequence of objects, x1; : : : ; xm such that each xi is very similar to its two neighbors, xi 1 and xi+1, but x1 and xm are very dissimilar. If we wish to make sure that whenever two elements are similar they share the same cluster, then we must put all of the elements of the sequence in the same cluster. However, in that case, we end up with dissimilar elements (x1 and xm) sharing a cluster, thus violating the second requirement.

To illustrate this point further, suppose that we would like to cluster the points in the following picture into two clusters.

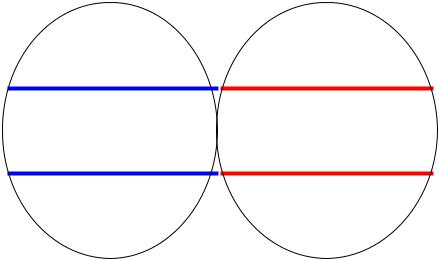


A clustering algorithm that emphasizes not separating close-by points (e.g., the Single Linkage algorithm that will be described in Section [22.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page310)) will cluster this input by separating it horizontally according to the two lines:

1. Clustering



In contrast, a clustering method that emphasizes not having far-away points share the same cluster (e.g., the 2-means algorithm that will be described in Section [22.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page310)) will cluster the same input by dividing it vertically into the right-hand half and the left-hand half:

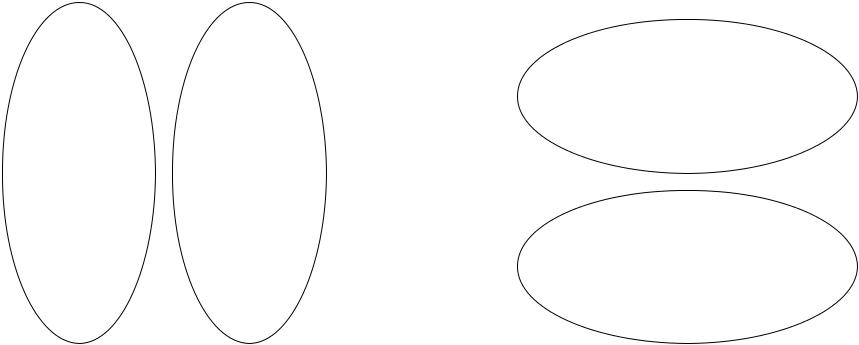


Another basic problem is the lack of \ground truth" for clustering, which is a common problem in unsupervised learning. So far in the book, we have mainly dealt with supervised learning (e.g., the problem of learning a classi er from labeled training data). The goal of supervised learning is clear { we wish to learn a classi er which will predict the labels of future examples as accurately as possible. Furthermore, a supervised learner can estimate the success, or the risk, of its hypotheses using the labeled training data by computing the empirical loss. In contrast, clustering is an unsupervised learning problem; namely, there are no labels that we try to predict. Instead, we wish to organize the data in some meaningful way. As a result, there is no clear success evaluation procedure for clustering. In fact, even on the basis of full knowledge of the underlying data distribution, it is not clear what is the \correct" clustering for that data or how to evaluate a proposed clustering.

Consider, for example, the following set of points in R2:

and suppose we are required to cluster them into two clusters. We have two highly justi able solutions:

Clustering 309



This phenomenon is not just arti cial but occurs in real applications. A given set of objects can be clustered in various di erent meaningful ways. This may be due to having di erent implicit notions of distance (or similarity) between objects, for example, clustering recordings of speech by the accent of the speaker versus clustering them by content, clustering movie reviews by movie topic versus clustering them by the review sentiment, clustering paintings by topic versus clustering them by style, and so on.

To summarize, there may be several very di erent conceivable clustering so-lutions for a given data set. As a result, there is a wide variety of clustering algorithms that, on some input data, will output very di erent clusterings.

A Clustering Model:

Clustering tasks can vary in terms of both the type of input they have and the type of outcome they are expected to compute. For concreteness, we shall focus on the following common setup:

Input | a set of elements, X , and a distance function over it. That is, a function d : X X ! R+ that is symmetric, satis es d(x; x) = 0 for all x 2 X and often also satis es the triangle inequality. Alternatively, the function could be a similarity function s : X X ! [0; 1] that is symmetric and satis es s(x; x) = 1 for all x 2 X . Additionally, some clustering algorithms also require an input parameter k (determining the number of required clusters).

Output | a partition of the domain set X into subsets. That is, C = (C1; : : : Ck) where Ski=1 Ci = X and for all i 6= j, Ci \Cj = ;. In some situations the clustering is \soft," namely, the partition of X into the di erent clusters is probabilistic where the output is a function assigning to each domain point, x 2 X , a vector (p1(x); : : : ; pk(x)), where pi(x) = P[x 2 Ci] is the probability that x belongs to cluster Ci. Another possible output is a clustering dendrogram (from Greek dendron = tree, gramma = draw-ing), which is a hierarchical tree of domain subsets, having the singleton sets in its leaves, and the full domain as its root. We shall discuss this formulation in more detail in the following.

1. Clustering

In the following we survey some of the most popular clustering methods. In the last section of this chapter we return to the high level discussion of what is clustering.

22.1 Linkage-Based Clustering Algorithms

Linkage-based clustering is probably the simplest and most straightforward paradigm of clustering. These algorithms proceed in a sequence of rounds. They start from the trivial clustering that has each data point as a single-point cluster. Then, repeatedly, these algorithms merge the \closest" clusters of the previous cluster-ing. Consequently, the number of clusters decreases with each such round. If kept going, such algorithms would eventually result in the trivial clustering in which all of the domain points share one large cluster. Two parameters, then, need to be determined to de ne such an algorithm clearly. First, we have to decide how to measure (or de ne) the distance between clusters, and, second, we have to determine when to stop merging. Recall that the input to a clustering algorithm

is a between-points distance function, d. There are many ways of extending d to a measure of distance between domain subsets (or clusters). The most common ways are

1. Single Linkage clustering, in which the between-clusters distance is de ned by the minimum distance between members of the two clusters, namely,

def

D(A; B) = minfd(x; y) : x 2 A; y 2 Bg

1. Average Linkage clustering, in which the distance between two clusters is de ned to be the average distance between a point in one of the clusters and a point in the other, namely,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| def | | 1 | j x2 | | X2 |
| j | | jj |
| D(A; B) = |  | A B | |  | d(x; y) |
|  |  |  |

A; y B

1. Max Linkage clustering, in which the distance between two clusters is de ned as the maximum distance between their elements, namely,

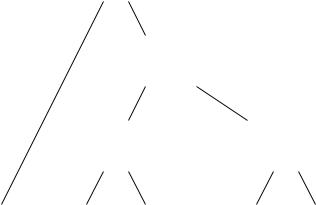
def

D(A; B) = maxfd(x; y) : x 2 A; y 2 Bg:

The linkage-based clustering algorithms are agglomerative in the sense that they start from data that is completely fragmented and keep building larger and larger clusters as they proceed. Without employing a stopping rule, the outcome of such an algorithm can be described by a clustering dendrogram: that is, a tree of domain subsets, having the singleton sets in its leaves, and the full domain as its root. For example, if the input is the elements X = fa; b; c; d; eg R2 with the Euclidean distance as depicted on the left, then the resulting dendrogram is the one depicted on the right:

|  |  |
| --- | --- |
| 22.2 k-Means and Other Cost Minimization Clusterings | 311 |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | fa; b; c; d; eg |  |
| a | fb; c; d; eg |  |
| e |  |  |
| d | fb; cg | fd; eg |
| c | fag fbg fcg fdg feg | |
| b |



The single linkage algorithm is closely related to Kruskal's algorithm for nding a minimal spanning tree on a weighted graph. Indeed, consider the full graph whose vertices are elements of X and the weight of an edge (x; y) is the distance d(x; y). Each merge of two clusters performed by the single linkage algorithm corresponds to a choice of an edge in the aforementioned graph. It is also possible to show that the set of edges the single linkage algorithm chooses along its run forms a minimal spanning tree.

If one wishes to turn a dendrogram into a partition of the space (a clustering), one needs to employ a stopping criterion. Common stopping criteria include

Fixed number of clusters { x some parameter, k, and stop merging clusters as soon as the number of clusters is k.

Distance upper bound { x some r 2 R+. Stop merging as soon as all the between-clusters distances are larger than r. We can also set r to be

maxfd(x; y) : x; y 2 X g for some < 1. In that case the stopping criterion is called \scaled distance upper bound."

22.2 k-Means and Other Cost Minimization Clusterings

Another popular approach to clustering starts by de ning a cost function over a parameterized set of possible clusterings and the goal of the clustering algorithm is to nd a partitioning (clustering) of minimal cost. Under this paradigm, the clustering task is turned into an optimization problem. The objective function is a function from pairs of an input, (X ; d), and a proposed clustering solution

1. = (C1; : : : ; Ck), to positive real numbers. Given such an objective function, which we denote by G, the goal of a clustering algorithm is de ned as nding, for

a given input (X ; d), a clustering C so that G((X ; d); C) is minimized. In order to reach that goal, one has to apply some appropriate search algorithm.

As it turns out, most of the resulting optimization problems are NP-hard, and some are even NP-hard to approximate. Consequently, when people talk about, say, k-means clustering, they often refer to some particular common approxima-tion algorithm rather than the cost function or the corresponding exact solution of the minimization problem.

Many common objective functions require the number of clusters, k, as a

1. Clustering

parameter. In practice, it is often up to the user of the clustering algorithm to choose the parameter k that is most suitable for the given clustering problem.

In the following we describe some of the most common objective functions.

The k-means objective function is one of the most popular clustering objectives. In k-means the data is partitioned into disjoint sets C1; : : : ; Ck where each Ci is represented by a centroid i. It is assumed that the input set X is embedded in some larger metric space (X 0; d) (so that X X 0) and centroids are members of X 0. The k-means objective function measures the squared distance between each point in X to the centroid of its cluster. The centroid of Ci is de ned to be

|  |  |  |  |
| --- | --- | --- | --- |
|  | X |  |  |
| i(Ci) = argmin | d(x; )2: | |  |
| 2X0 | x Ci |  |  |
|  | 2 |  |  |
| Then, the k-means objective is |  |  |  |
|  | k |  |  |
|  | X X | |  |
| Gk means((X ; d); (C1; : : : ; Ck)) = | | d(x; i(Ci))2: |  |
|  | i=1 x2Ci | |  |
| This can also be rewritten as |  |  |  |
|  |  | k |  |
|  |  | X X |  |
| Gk means((X ; d); (C1; : : : ; Ck)) = 1 | min | 2 | (22.1) |
| d(x; i) : |
| ;::: k2X0 |  |
|  |  | i=1 x2Ci |  |

The k-means objective function is relevant, for example, in digital com-munication tasks, where the members of X may be viewed as a collection of signals that have to be transmitted. While X may be a very large set of real valued vectors, digital transmission allows transmitting of only a nite number of bits for each signal. One way to achieve good transmis-sion under such constraints is to represent each member of X by a \close" member of some nite set 1; : : : k, and replace the transmission of any x 2 X by transmitting the index of the closest i. The k-means objective can be viewed as a measure of the distortion created by such a transmission representation scheme.

The k-medoids objective function is similar to the k-means objective, except that it requires the cluster centroids to be members of the input set. The objective function is de ned by

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | k |
|  |  |  |  | X X |
| GK medoid((X ; d); (C1; : : : ; Ck)) = |  | min | | 2 |
| 1 | d(x; i) : |
| ;::: k | 2X |

i=1 x2Ci

The k-median objective function is quite similar to the k-medoids objec-tive, except that the \distortion" between a data point and the centroid of its cluster is measured by distance, rather than by the square of the distance:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | k |
| GK median((X ; d); (C1; : : : ; Ck)) = |  |  |  | X X |
| 1 | min | | d(x; i): |
| ;::: k | 2X |

i=1 x2Ci

|  |  |
| --- | --- |
| 22.2 k-Means and Other Cost Minimization Clusterings | 313 |
|  |  |

An example where such an objective makes sense is the facility location problem. Consider the task of locating k re stations in a city. One can model houses as data points and aim to place the stations so as to minimize the average distance between a house and its closest re station.

The previous examples can all be viewed as center-based objectives. The so-lution to such a clustering problem is determined by a set of cluster centers, and the clustering assigns each instance to the center closest to it. More gener-ally, center-based objective is determined by choosing some monotonic function

1. : R+ ! R+ and then de ning

|  |  |  |
| --- | --- | --- |
|  |  | k |
| Gf ((X ; d); (C1; : : : Ck)) = |  | X X |
| min | f(d(x; i)); |
| 1;::: k2X0 |
|  |  | i=1 x2Ci |

where X 0 is either X or some superset of X .

Some objective functions are not center based. For example, the sum of in-

cluster distances (SOD)

|  |  |
| --- | --- |
|  | k |
| GSOD((X ; d); (C1; : : : Ck)) = | X X |
| d(x; y) |
|  | i=1 x;y2Ci |

and the MinCut objective that we shall discuss in Section [22.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page315) are not center-based objectives.

22.2.1 The k-Means Algorithm

The k-means objective function is quite popular in practical applications of clus-tering. However, it turns out that nding the optimal k-means solution is of-ten computationally infeasible (the problem is NP-hard, and even NP-hard to approximate to within some constant). As an alternative, the following simple iterative algorithm is often used, so often that, in many cases, the term k-means Clustering refers to the outcome of this algorithm rather than to the cluster-ing that minimizes the k-means objective cost. We describe the algorithm with respect to the Euclidean distance function d(x; y) = kx yk.

k-Means

input: X Rn ; Number of clusters k

initialize: Randomly choose initial centroids 1; : : : ; k repeat until convergence

8i 2 [k] set Ci = fx 2 X : i = argminj kx jkg

(break ties in some arbitrary manner)

8i 2 [k] update i = j 1 j Px2C x

Ci i

lemma 22.1 Each iteration of the k-means algorithm does not increase the k-means objective function (as given in Equation ([22.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page312))).

1. Clustering

Proof To simplify the notation, let us use the shorthand G(C1; : : : ; Ck) for the k-means objective, namely,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | k |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | X X |  |  |  |  | 2 |  |  |  |
|  | G(C | ; : : : ; C ) = |  |  |  | min | k | x |  |  | ik | : | (22.2) | |
|  | 1 | k | 1;:::; k2Rn | | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  | i=1 x2Ci |  |  |  |  |  |  |  |  |
| It is convenient to de ne (Ci) = | | |  | 1 |  |  | x and note that (Ci) = argmin | | | | | | | | n |
|  | Ci |  | x Ci |
|  |  |  | j | j |  |  |  |  |  |  |  | 2R |  |
|  |  |  |  | 2 |  |  |  |  |  |  |  |  |  |
| 2. Therefore, we can rewrite the k-means objective as | | | | | | | | |  |  |  |  |  |  |
| k |  |  |  |  |  | P |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | k |  |  |  |  |  |  |  |  |  |
|  | G(C1; : : : ; Ck) = X X kx (Ci)k2: | | | | | | | | | |  |  |  | (22.3) | |

i=1 x2Ci

P

x2Ci kx

Consider the update at iteration t of the k-means algorithm. Let C1(t 1); : : : ; Ck(t 1)

be the previous partition, let (it 1) = (Ci(t 1)), and let C1(t); : : : ; Ck(t) be the new partition assigned at iteration t. Using the de nition of the objective as given in Equation ([22.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page314)) we clearly have that

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | k |  |  |  | kxi(t 1)k2: | | | |  |
| G(C1(t); : : : ; Ck(t)) | | |  |  |  | (22.4) |
|  |  | X | | xXi | | |  |  |  |  |
|  |  | i=1 | | 2 | C | (t) |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| In addition, the de nition of the new partition (C(t) | | | | | | | | | ; : : : ; C(t)) implies that it | |
| (C1; : : : ; Ck). Hence, | Pi=1 | Px2Ci k |  |  |  | i | k | 1 | k |  |
| x |  | 2 over all possible partitions | | |
| minimizes the expression | k |  |  | (t 1) | |  |

k

X X

i=1 x2Ci(t)

kx (it 1)k2

|  |  |
| --- | --- |
| k |  |
| X X kxi(t 1)k2: | (22.5) |

i=1 x2Ci(t 1)

Using Equation ([22.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page314)) we have that the right-hand side of Equation ([22.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page314)) equals G(C1(t 1); : : : ; Ck(t 1)). Combining this with Equation ([22.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page314)) and Equation ([22.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page314)), we obtain that G(C1(t); : : : ; Ck(t)) G(C1(t 1); : : : ; Ck(t 1)), which concludes our proof. 

While the preceding lemma tells us that the k-means objective is monotonically nonincreasing, there is no guarantee on the number of iterations the k-means al-gorithm needs in order to reach convergence. Furthermore, there is no nontrivial lower bound on the gap between the value of the k-means objective of the al-gorithm's output and the minimum possible value of that objective function. In fact, k-means might converge to a point which is not even a local minimum (see Exercise [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page321)). To improve the results of k-means it is often recommended to repeat the procedure several times with di erent randomly chosen initial centroids (e.g., we can choose the initial centroids to be random points from the data).

|  |  |
| --- | --- |
| 22.3 Spectral Clustering | 315 |
|  |  |

22.3 Spectral Clustering

Often, a convenient way to represent the relationships between points in a data set X = fx1; : : : ; xmg is by a similarity graph; each vertex represents a data point xi, and every two vertices are connected by an edge whose weight is their similarity, Wi;j = s(xi; xj), where W 2 Rm;m. For example, we can set Wi;j = exp( d(xi; xj)2= 2), where d( ; ) is a distance function and is a parameter. The clustering problem can now be formulated as follows: We want to nd a partition of the graph such that the edges between di erent groups have low weights and the edges within a group have high weights.

In the clustering objectives described previously, the focus was on one side of our intuitive de nition of clustering { making sure that points in the same cluster are similar. We now present objectives that focus on the other requirement { points separated into di erent clusters should be nonsimilar.

22.3.1 Graph Cut

Given a graph represented by a similarity matrix W , the simplest and most direct way to construct a partition of the graph is to solve the mincut problem, which chooses a partition C1; : : : ; Ck that minimizes the objective

k

X X

cut(C1; : : : ; Ck) = Wr;s:

i=1 r2Ci;s2=Ci

For k = 2, the mincut problem can be solved e ciently. However, in practice it often does not lead to satisfactory partitions. The problem is that in many cases, the solution of mincut simply separates one individual vertex from the rest of the graph. Of course, this is not what we want to achieve in clustering, as clusters should be reasonably large groups of points.

Several solutions to this problem have been suggested. The simplest solution is to normalize the cut and de ne the normalized mincut objective as follows:

k

X 1 X

RatioCut(C1; : : : ; Ck) = Wr;s:

i=1 jCij r2Ci;s 2=Ci

The preceding objective assumes smaller values if the clusters are not too small. Unfortunately, introducing this balancing makes the problem computationally hard to solve. Spectral clustering is a way to relax the problem of minimizing RatioCut.

22.3.2 Graph Laplacian and Relaxed Graph Cuts

The main mathematical object for spectral clustering is the graph Laplacian matrix. There are several di erent de nitions of graph Laplacian in the literature, and in the following we describe one particular de nition.

1. Clustering

definition 22.2 (Unnormalized Graph Laplacian) The unnormalized graph Laplacian is the m m matrix L = D W where D is a diagonal matrix with

Pm

Di;i = j=1 Wi;j. The matrix D is called the degree matrix.

The following lemma underscores the relation between RatioCut and the Lapla-cian matrix.

lemma 22.3 Let C1; : : : ; Ck be a clustering and let H 2 Rm;k be the matrix such that

Hi;j = pj1 j 1[i2Cj]:

Cj

Then, the columns of H are orthonormal to each other and

RatioCut(C1; : : : ; Ck) = trace(H> L H):

Proof Let h1; : : : ; hk be the columns of H. The fact that these vectors are orthonormal is immediate from the de nition. Next, by standard algebraic ma-nipulations, it can be shown that trace(H> L H) = Pki=1 h>iLhi and that for any vector v we have

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| v>Lv = 2 | | |  | Dr;rvr2 2 | r;s | vrvsWr;s + | | | Ds;svs2! = 2 | | | r;s | Wr;s(vr vs)2: |
| 1 | | |  |  |  |  |  |  | 1 | | |  |  |
|  |  |  | Xr |  | X |  |  |  | Xs |  |  | X |  |
|  |  |  |  |  |  |  |  |  |
| Applying this with v = hi and noting that (hi;r hi;s)2 | | | | | | | | | | | | is nonzero only if | |
| r 2 Ci; s 2= Ci or the other way around, we obtain that | | | | | | | | | | | |  |  |
|  |  |  |  | hi>Lhi | | = | 1 | r2CXi2 i | | | |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  | j j |  |  |
|  |  |  |  | Ci | | Wr;s: | | |  |  |
|  |  |  |  |  |  |  | ;s =C | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

Therefore, to minimize RatioCut we can search for a matrix H whose columns

p

are orthonormal and such that each Hi;j is either 0 or 1= jCjj. Unfortunately, this is an integer programming problem which we cannot solve e ciently. Instead, we relax the latter requirement and simply search an orthonormal matrix H 2 Rm;k that minimizes trace(H> L H). As we will see in the next chapter about PCA (particularly, the proof of Theorem [23.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page325)), the solution to this problem is to set U to be the matrix whose columns are the eigenvectors corresponding to the k minimal eigenvalues of L. The resulting algorithm is called Unnormalized Spectral Clustering.

|  |  |
| --- | --- |
| 22.4 Information Bottleneck\* | 317 |
|  |  |

22.3.3 Unnormalized Spectral Clustering

Unnormalized Spectral Clustering

Input: W 2 Rm;m ; Number of clusters k

Initialize: Compute the unnormalized graph Laplacian L

Let U 2 Rm;k be the matrix whose columns are the eigenvectors of L corresponding to the k smallest eigenvalues

Let v1; : : : ; vm be the rows of U

Cluster the points v1; : : : ; vm using k-means

Output: Clusters C1; : : : ; CK of the k-means algorithm

The spectral clustering algorithm starts with nding the matrix H of the k eigenvectors corresponding to the smallest eigenvalues of the graph Laplacian matrix. It then represents points according to the rows of H. It is due to the properties of the graph Laplacians that this change of representation is useful. In many situations, this change of representation enables the simple k-means algorithm to detect the clusters seamlessly. Intuitively, if H is as de ned in Lemma [22.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page316) then each point in the new representation is an indicator vector whose value is nonzero only on the element corresponding to the cluster it belongs to.

22.4 Information Bottleneck\*

The information bottleneck method is a clustering technique introduced by Tishby, Pereira, and Bialek. It relies on notions from information theory. To illustrate the method, consider the problem of clustering text documents where each document is represented as a bag-of-words; namely, each document is a vector x = f0; 1gn, where n is the size of the dictionary and xi = 1 i the word corresponding to index i appears in the document. Given a set of m documents, we can interpret the bag-of-words representation of the m documents as a joint probability over a random variable x, indicating the identity of a document (thus taking values in [m]), and a random variable y, indicating the identity of a word in the dictionary (thus taking values in [n]).

With this interpretation, the information bottleneck refers to the identity of a clustering as another random variable, denoted C, that takes values in [k] (where k will be set by the method as well). Once we have formulated x; y; C as random variables, we can use tools from information theory to express a clustering objective. In particular, the information bottleneck objective is

min I(x; C) I(C; y) ;

p(Cjx)

where I( ; ) is the mutual information between two random variables,[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page317) is a

1 That is, given a probability function, p over the pairs (x; C),

1. Clustering

parameter, and the minimization is over all possible probabilistic assignments of points to clusters. Intuitively, we would like to achieve two contradictory goals. On one hand, we would like the mutual information between the identity of the document and the identity of the cluster to be as small as possible. This re ects the fact that we would like a strong compression of the original data. On the other hand, we would like high mutual information between the clustering variable and the identity of the words, which re ects the goal that the \relevant" information about the document (as re ected by the words that appear in the

document) is retained. This generalizes the classical notion of minimal su cient

statistics[2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page318) used in parametric statistics to arbitrary distributions.

Solving the optimization problem associated with the information bottleneck principle is hard in the general case. Some of the proposed methods are similar to the EM principle, which we will discuss in Chapter [24](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page342).

22.5 A High Level View of Clustering

So far, we have mainly listed various useful clustering tools. However, some fun-damental questions remain unaddressed. First and foremost, what is clustering? What is it that distinguishes a clustering algorithm from any arbitrary function that takes an input space and outputs a partition of that space? Are there any basic properties of clustering that are independent of any speci c algorithm or task?

One method for addressing such questions is via an axiomatic approach. There have been several attempts to provide an axiomatic de nition of clustering. Let us demonstrate this approach by presenting the attempt made by Kleinberg (2003).

Consider a clustering function, F , that takes as input any nite domain X with a dissimilarity function d over its pairs and returns a partition of X .

Consider the following three properties of such a function:

Scale Invariance (SI) For any domain set X , dissimilarity function d, and any > 0, the following should hold: F (X ; d) = F (X ; d) (where

def

( d)(x; y) = d(x; y)).

Richness (Ri) For any nite X and every partition C = (C1; : : : Ck) of X (into nonempty subsets) there exists some dissimilarity function d over X such that F (X ; d) = C.

|  |  |
| --- | --- |
| P P | p(a;b) |
| I(x; C) = a b p(a; b) log | p(a)p(b) , where the sum is over all values x can take and all |

values C can take.

1. A su cient statistic is a function of the data which has the property of su ciency with respect to a statistical model and its associated unknown parameter, meaning that \no other statistic which can be calculated from the same sample provides any additional information as to the value of the parameter." For example, if we assume that a variable is distributed normally with a unit variance and an unknown expectation, then the average function is a su cient statistic.

|  |  |
| --- | --- |
| 22.5 A High Level View of Clustering | 319 |
|  |  |

Consistency (Co) If d and d0 are dissimilarity functions over X , such that for every x; y 2 X , if x; y belong to the same cluster in F (X ; d) then d0(x; y) d(x; y) and if x; y belong to di erent clusters in F (X ; d) then d0(x; y) d(x; y), then F (X ; d) = F (X ; d0).

A moment of re ection reveals that the Scale Invariance is a very natural requirement { it would be odd to have the result of a clustering function depend on the units used to measure between-point distances. The Richness requirement basically states that the outcome of the clustering function is fully controlled by the function d, which is also a very intuitive feature. The third requirement, Consistency, is the only requirement that refers to the basic (informal) de nition of clustering { we wish that similar points will be clustered together and that dissimilar points will be separated to di erent clusters, and therefore, if points that already share a cluster become more similar, and points that are already separated become even less similar to each other, the clustering function should have even stronger \support" of its previous clustering decisions.

However, Kleinberg (2003) has shown the following \impossibility" result:

theorem 22.4 There exists no function, F , that satis es all the three proper-ties: Scale Invariance, Richness, and Consistency.

Proof Assume, by way of contradiction, that some F does satisfy all three properties. Pick some domain set X with at least three points. By Richness, there must be some d1 such that F (X ; d1) = ffxg : x 2 X g and there also exists some d2 such that F (X ; d2) 6= F (X ; d1).

Let 2 R+ be such that for every x; y 2 X , d2(x; y) d1(x; y). Let d3 = d2. Consider F (X ; d3). By the Scale Invariance property of F , we should have F (X ; d3) = F (X ; d2). On the other hand, since all distinct x; y 2 X reside in di erent clusters w.r.t. F (X ; d1), and d3(x; y) d1(x; y), the Consistency of F implies that F (X ; d3) = F (X ; d1). This is a contradiction, since we chose d1; d2 so that F (X ; d2) 6= F (X ; d1). 

It is important to note that there is no single \bad property" among the three properties. For every pair of the the three axioms, there exist natural clustering functions that satisfy the two properties in that pair (one can even construct such examples just by varying the stopping criteria for the Single Linkage clustering function). On the other hand, Kleinberg shows that any clustering algorithm that minimizes any center-based objective function inevitably fails the consis-tency property (yet, the k-sum-of-in-cluster-distances minimization clustering does satisfy Consistency).

The Kleinberg impossibility result can be easily circumvented by varying the properties. For example, if one wishes to discuss clustering functions that have a xed number-of-clusters parameter, then it is natural to replace Richness by k-Richness (namely, the requirement that every partition of the domain into k subsets is attainable by the clustering function). k-Richness, Scale Invariance and Consistency all hold for the k-means clustering and are therefore consistent.

1. Clustering

Alternatively, one can relax the Consistency property. For example, say that two clusterings C = (C1; : : : Ck) and C0 = (C10; : : : Cl0) are compatible if for every clusters Ci 2 C and Cj0 2 C0, either Ci Cj0 or Cj0 Ci or Ci \ Cj0 = ; (it is worthwhile noting that for every dendrogram, every two clusterings that are ob-tained by trimming that dendrogram are compatible). \Re nement Consistency"

is the requirement that, under the assumptions of the Consistency property, the

new clustering F (X ; d0) is compatible with the old clustering F (X ; d). Many common clustering functions satisfy this requirement as well as Scale Invariance and Richness. Furthermore, one can come up with many other, di erent, prop-erties of clustering functions that sound intuitive and desirable and are satis ed by some common clustering functions.

There are many ways to interpret these results. We suggest to view it as indi-cating that there is no \ideal" clustering function. Every clustering function will inevitably have some \undesirable" properties. The choice of a clustering func-tion for any given task must therefore take into account the speci c properties of that task. There is no generic clustering solution, just as there is no clas-si cation algorithm that will learn every learnable task (as the No-Free-Lunch theorem shows). Clustering, just like classi cation prediction, must take into account some prior knowledge about the speci c task at hand.

22.6 Summary

Clustering is an unsupervised learning problem, in which we wish to partition a set of points into \meaningful" subsets. We presented several clustering ap-proaches including linkage-based algorithms, the k-means family, spectral clus-tering, and the information bottleneck. We discussed the di culty of formalizing the intuitive meaning of clustering.

22.7 Bibliographic Remarks

The k-means algorithm is sometimes named Lloyd's algorithm, after Stuart Lloyd, who proposed the method in 1957. For a more complete overview of spectral clustering we refer the reader to the excellent tutorial by Von Luxburg (2007). The information bottleneck method was introduced by Tishby, Pereira

1. Bialek (1999). For an additional discussion on the axiomatic approach see Ackerman & Ben-David (2008).

22.8 Exercises

1. Suboptimality of k-Means: For every parameter t > 1, show that there exists an instance of the k-means problem for which the k-means algorithm

|  |  |
| --- | --- |
| 22.8 Exercises | 321 |
|  |  |

(might) nd a solution whose k-means objective is at least t OPT, where OPT is the minimum k-means objective.

1. k-Means Might Not Necessarily Converge to a Local Minimum: Show that the k-means algorithm might converge to a point which is not

a local minimum. Hint: Suppose that k = 2 and the sample points are

f1; 2; 3; 4g R suppose we initialize the k-means with the centers f2; 4g; and suppose we break ties in the de nition of Ci by assigning i to be the smallest value in argminj kx jk.

1. Given a metric space (X ; d), where jX j < 1, and k 2 N, we would like to nd a partition of X into C1; : : : ; Ck which minimizes the expression

Gk diam((X ; d); (C1; : : : ; Ck)) = max diam(Cj);

j2[d]

where diam(Cj) = maxx;x02Cj d(x; x0) (we use the convention diam(Cj) = 0 if jCjj < 2).

Similarly to the k-means objective, it is NP-hard to minimize the k-diam objective. Fortunately, we have a very simple approximation algorithm: Initially, we pick some x 2 X and set 1 = x. Then, the algorithm iteratively sets

8j 2 f2; : : : ; kg; j = argmax min d(x; i):

x2X i2[j 1]

Finally, we set

8i 2 [k]; Ci = fx 2 X : i = argmin d(x; j)g:

j2[k]

Prove that the algorithm described is a 2-approximation algorithm. That

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ^ |  |  |  | ^ |  |  |  |  |  |  |  |
| is, if we denote its output by C1 | | | | | | ; : : : ; Ck, and denote the optimal solution by | | | | | | | | | | |
| C1 ; : : : ; Ck , then, | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| G | k diam | (( | X | ; d); (C^ | ; : : : ; C^ |  | )) |  | 2 |  | G | k diam | (( | X | ; d); (C ; : : : ; C )): | |
|  |  | 1 | k | |  |  |  |  | 1 | k |

Hint: Consider the point k+1 (in other words, the next center we would have chosen, if we wanted k + 1 clusters). Let r = minj2[k] d( j; k+1). Prove the following inequalities

X ^ ^

Gk diam(( ; d); (C1; : : : ; Ck)) 2r

Gk diam((X; d); (C1 ; : : : ; Ck )) r:

1. Recall that a clustering function, F , is called Center-Based Clustering if, for some monotonic function f : R+ ! R+, on every given input (X ; d), F (X ; d) is a clustering that minimizes the objective

|  |  |  |
| --- | --- | --- |
|  |  | k |
| Gf ((X ; d); (C1; : : : Ck)) = |  | X X |
| min | f(d(x; i)); |
| 1;::: k2X0 |
|  |  | i=1 x2Ci |

where X 0 is either X or some superset of X .

1. Clustering

Prove that for every k > 1 the k-diam clustering function de ned in the previous exercise is not a center-based clustering function.

Hint: Given a clustering input (X ; d), with jX j > 2, consider the e ect of adding many close-by points to some (but not all) of the members of X , on either the k-diam clustering or any given center-based clustering.

1. Recall that we discussed three clustering \properties": Scale Invariance, Rich-ness, and Consistency. Consider the Single Linkage clustering algorithm.
   1. Find which of the three properties is satis ed by Single Linkage with the Fixed Number of Clusters (any xed nonzero number) stopping rule.
   2. Find which of the three properties is satis ed by Single Linkage with the Distance Upper Bound (any xed nonzero upper bound) stopping rule.
   3. Show that for any pair of these properties there exists a stopping criterion for Single Linkage clustering, under which these two axioms are satis ed.
2. Given some number k, let k-Richness be the following requirement:

For any nite X and every partition C = (C1; : : : Ck) of X (into nonempty subsets) there exists some dissimilarity function d over X such that F (X ; d) = C.

Prove that, for every number k, there exists a clustering function that satis es the three properties: Scale Invariance, k-Richness, and Consistency.

1. Dimensionality Reduction

Dimensionality reduction is the process of taking data in a high dimensional space and mapping it into a new space whose dimensionality is much smaller. This process is closely related to the concept of (lossy) compression in infor-mation theory. There are several reasons to reduce the dimensionality of the data. First, high dimensional data impose computational challenges. Moreover, in some situations high dimensionality might lead to poor generalization abili-ties of the learning algorithm (for example, in Nearest Neighbor classi ers the sample complexity increases exponentially with the dimension|see Chapter [19](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page258)). Finally, dimensionality reduction can be used for interpretability of the data, for nding meaningful structure of the data, and for illustration purposes.

In this chapter we describe popular methods for dimensionality reduction. In those methods, the reduction is performed by applying a linear transformation to the original data. That is, if the original data is in Rd and we want to embed it into Rn (n < d) then we would like to nd a matrix W 2 Rn;d that induces the mapping x 7!W x. A natural criterion for choosing W is in a way that will enable a reasonable recovery of the original x. It is not hard to show that in general, exact recovery of x from W x is impossible (see Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page339)).

The rst method we describe is called Principal Component Analysis (PCA). In PCA, both the compression and the recovery are performed by linear transfor-mations and the method nds the linear transformations for which the di erences between the recovered vectors and the original vectors are minimal in the least squared sense.

Next, we describe dimensionality reduction using random matrices W . We derive an important lemma, often called the \Johnson-Lindenstrauss lemma," which analyzes the distortion caused by such a random dimensionality reduction technique.

Last, we show how one can reduce the dimension of all sparse vectors using again a random matrix. This process is known as Compressed Sensing. In this case, the recovery process is nonlinear but can still be implemented e ciently using linear programming.

We conclude by underscoring the underlying \prior assumptions" behind PCA and compressed sensing, which can help us understand the merits and pitfalls of the two methods.

1. Dimensionality Reduction

23.1 Principal Component Analysis (PCA)

Let x1; : : : ; xm be m vectors in Rd. We would like to reduce the dimensional-ity of these vectors using a linear transformation. A matrix W 2 Rn;d, where n < d, induces a mapping x 7!W x, where W x 2 Rn is the lower dimensionality representation of x. Then, a second matrix U 2 Rd;n can be used to (approxi-mately) recover each original vector x from its compressed version. That is, for a compressed vector y = W x, where y is in the low dimensional space Rn, we can construct x~ = Uy, so that x~ is the recovered version of x and resides in the original high dimensional space Rd.

In PCA, we nd the compression matrix W and the recovering matrix U so that the total squared distance between the original and recovered vectors is minimal; namely, we aim at solving the problem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 2 | m |  |  |
| Xi |  |  |
|  | argmin | 2 |  |  |
|  | =1 kxi U W xik2 | : | (23.1) |
| W | Rn;d;U Rd;n |

To solve this problem we rst show that the optimal solution takes a speci c form.

lemma 23.1 Let (U; W ) be a solution to Equation ([23.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page324)). Then the columns of

1. are orthonormal (namely, U>U is the identity matrix of Rn) and W = U>.

Proof Fix any U; W and consider the mapping x 7!U W x. The range of this mapping, R = fU W x : x 2 Rdg, is an n dimensional linear subspace of Rd. Let

1. 2 Rd;n be a matrix whose columns form an orthonormal basis of this subspace, namely, the range of V is R and V >V = I. Therefore, each vector in R can be written as V y where y 2 Rn. For every x 2 Rd and y 2 Rn we have

kx V yk22 = kxk2 + y>V >V y 2y>V >x = kxk2 + kyk2 2y>(V >x);

where we used the fact that V >V is the identity matrix of Rn. Minimizing the preceding expression with respect to y by comparing the gradient with respect to y to zero gives that y = V >x. Therefore, for each x we have that

1. V >x = argmin kx x~k22:

x~2R

In particular this holds for x1; : : : ; xm and therefore we can replace U; W by V; V > and by that do not increase the objective

m

m

X

X

kxi UW xik22

kxi

V V >xik22:

i=1

i=1

Since this holds for every U; W the proof of the lemma follows.

On the basis of the preceding lemma, we can rewrite the optimization problem given in Equation ([23.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page324)) as follows:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 |  |  | m |  |  |  |  |  |  |
|  | argmin | | Xi | x |  | U U>x | 2 | : | (23.2) |
|  |  | k | i |
| U |  | R | d;n:U>U=I |  |  | ik2 |  |  |
|  |  |  | =1 |  |  |  |  |  |  |

|  |  |
| --- | --- |
| 23.1 Principal Component Analysis (PCA) | 325 |
|  |  |

We further simplify the optimization problem by using the following elementary algebraic manipulations. For every x 2 Rd and a matrix U 2 Rd;n such that U>U = I we have

|  |  |
| --- | --- |
| kx UU >xk2 = kxk2 2x>UU >x + x>U U>UU >x |  |
| = kxk2 x>UU >x |  |
| = kxk2 trace(U>xx>U); | (23.3) |

where the trace of a matrix is the sum of its diagonal entries. Since the trace is a linear operator, this allows us to rewrite Equation ([23.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page324)) as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| U Rd;n:U>U=I trace | | U> =1 xixi>U! | : | (23.4) |
| 2 |  | m |  |  |
| argmax | Xi |  |  |
|  |  |  |  |

Let A = Pmi=1 xix>i. The matrix A is symmetric and therefore it can be written using its spectral decomposition as A = VDV >, where D is diagonal and V >V = VV > = I. Here, the elements on the diagonal of D are the eigenvalues of A and the columns of V are the corresponding eigenvectors. We assume without loss of generality that D1;1 D2;2 Dd;d. Since A is positive semide nite it also holds that Dd;d 0. We claim that the solution to Equation ([23.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page325)) is the matrix U whose columns are the n eigenvectors of A corresponding to the largest n eigenvalues.

theorem 23.2 Let x1; : : : ; xm be arbitrary vectors in Rd, let A = Pmi=1 xix>i,

and let u1; : : : ; un be n eigenvectors of the matrix A corresponding to the largest n eigenvalues of A. Then, the solution to the PCA optimization problem given in Equation ([23.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page324)) is to set U to be the matrix whose columns are u1; : : : ; un and to set W = U>.

Proof Let VDV > be the spectral decomposition of A. Fix some matrix U 2 Rd;n with orthonormal columns and let B = V >U. Then, VB = VV >U = U. It follows that

U>AU = B>V >VDV >VB = B>DB;

and therefore

|  |  |  |
| --- | --- | --- |
| d | n |  |
| X | Xi | Bj;i2: |
| trace(U>AU ) = | Dj;j |
| j=1 | =1 |  |

Note that B>B = U>VV >U = U>U = I. Therefore, the columns of B are

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| also orthonormal, which implies that | | | | | d |  | n |  |  |  | 2 |  |  |  |  | ~ |
| j=1 | | i=1 Bj;i | | | | | = n. In addition, let B 2 | | | | |
| d;d | |  |  |  | n columns are the columns of B and in | | | | | | | | | | | |
| R be a matrix such that its rst P | | | | | |  | P |  | d |  | B~2 = 1, which implies that | | | | | |
| addition B~>B~ = I. Then, for every j we have | | | | | | | |  | i=1 | |
| P | n |  |  |  |  |  | P | |  | j;i |  |  |  |  |
|  |  |  |  |  |  |  | d |  |  |  |  |  |
| i=1 Bj;i2 | 1. It follows that: |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | Xj | | |  |  |  |  |
|  |  |  | trace(U>AU) | 2 |  | k | k |  |  |  |  |  | : |
|  |  |  |  | max | | |  |  |  |  | D |  |  |
|  |  |  |  | [0;1]d : | |  | 1 | n | |  | =1 | | j;j |  | j |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Pm

i=1

1. Dimensionality Reduction

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| It is not hard | to verify | | (see | Exercise | | [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page339)) | that | | |  | the | | right-hand | | | side equals to | |
| n |  |  |  |  |  | > |  |  |  | |  | j =1 | | j;j |  |  |  |
| P |  |  |  |  |  |  |  |  | 2 | R | d;n with or- |
| j=1 Dj;j. We have therefore shown that for | | | | | | | | | every matrix U | | | | | |
|  |  |  | n |  |  |  |
| thonormal columns it holds that trace(U AU) | | | | | | | | |  |  |  |  |  | D . On the other hand, | | | |
| if we set U to be the matrix whosen columns are | | | | | | | | | | the n leading eigenvectors of A | | | | | | | |
|  | P | |  |  |  |  |  |
| we obtain that trace(U>AU) = Pj=1 Dj;j, and this concludes our proof. | | | | | | | | | | | | | | | | | |
| Remark 23.1 | The proof of Theorem [23.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page325) also tells us that the value of the | | | | | | | | | | | | | | | | |
|  |  |  |  |  | n |  |  |  |  |  |  |  |  |  |  |  |  |
| objective of Equation ([23.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page325)) is | | | |  | i=1 Di;i. Combining this with Equation ([23.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page325)) | | | | | | | | | | | | |
|  | P | m | 2 |  |  | i=nP | | d |  | D | |  | we obtain that the optimal | | | | |
| and noting that |  |  |  | trace(A) = | | |  | i=1 | | i;i |
|  | i=1 kxik = P | | | P | d |  |  |  |  |  |  |  |  |
| objective value of Equation ([23.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page324)) is | | | | | |  | +1 Di;i. | | | | |  |  |  |  |  |  |

Remark 23.2 It is a common practice to \center" the examples before applying

PCA. That is, we rst calculate = m1 xi and then apply PCA on the

vectors (x1 ); : : : ; (xm ). This is also related to the interpretation of PCA as variance maximization (see Exercise [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page340)).

23.1.1 A More E cient Solution for the Case d m

In some situations the original dimensionality of the data is much larger than the number of examples m. The computational complexity of calculating the PCA solution as described previously is O(d3) (for calculating eigenvalues of A) plus O(md2) (for constructing the matrix A). We now show a simple trick that enables us to calculate the PCA solution more e ciently when d m.

Recall that the matrix A is de ned to be Pmi=1 xix>i. It is convenient to rewrite A = X>X where X 2 Rm;d is a matrix whose ith row is x>i. Consider the matrix B = XX>. That is, B 2 Rm;m is the matrix whose i; j element equals hxi; xji. Suppose that u is an eigenvector of B: That is, Bu = u for some

2 R. Multiplying the equality by X> and using the de nition of B we obtain X>XX>u = X>u. But, using the de nition of A, we get that A(X>u) =

(X>u). Thus, X>u is an eigenvector of A with eigenvalue of .

kX>uk

We can therefore calculate the PCA solution by calculating the eigenvalues of

1. instead of A. The complexity is O(m3) (for calculating eigenvalues of B) and m2d (for constructing the matrix B).

Remark 23.3 The previous discussion also implies that to calculate the PCA solution we only need to know how to calculate inner products between vectors. This enables us to calculate PCA implicitly even when d is very large (or even in nite) using kernels, which yields the kernel PCA algorithm.

23.1.2 Implementation and Demonstration

A pseudocode of PCA is given in the following.

|  |  |
| --- | --- |
| 23.1 Principal Component Analysis (PCA) | 327 |
|  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1.5 |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |
| 0.5 |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |
| −0.5 |  |  |  |  |  |  |
| −1 |  |  |  |  |  |  |
| −1.5 | −1 | −0.5 | 0 | 0.5 | 1 | 1.5 |
| −1.5 |

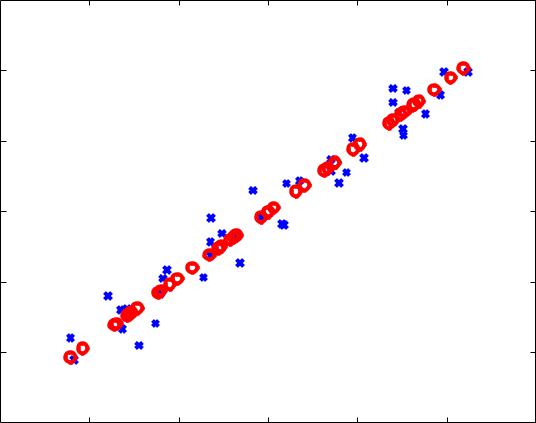


Figure 23.1 A set of vectors in R2 (blue x's) and their reconstruction after dimensionality reduction to R1 using PCA (red circles).

PCA

input

A matrix of m examples X 2 Rm;d

number of components n

if (m > d)

A = X>X

Let u1; : : : ; un be the eigenvectors of A with largest eigenvalues else

B = XX>

Let v1; : : : ; vn be the eigenvectors of B with largest eigenvalues

for i = 1; : : : ; n set ui = 1 X>vi

kX>vik

output: u1; : : : ; un

To illustrate how PCA works, let us generate vectors in R2 that approximately reside on a line, namely, on a one dimensional subspace of R2. For example, suppose that each example is of the form (x; x + y) where x is chosen uniformly at random from [ 1; 1] and y is sampled from a Gaussian distribution with mean 0 and standard deviation of 0:1. Suppose we apply PCA on this data. Then, the

eigenvector corresponding to the largest eigenvalue will be close to the vector p p

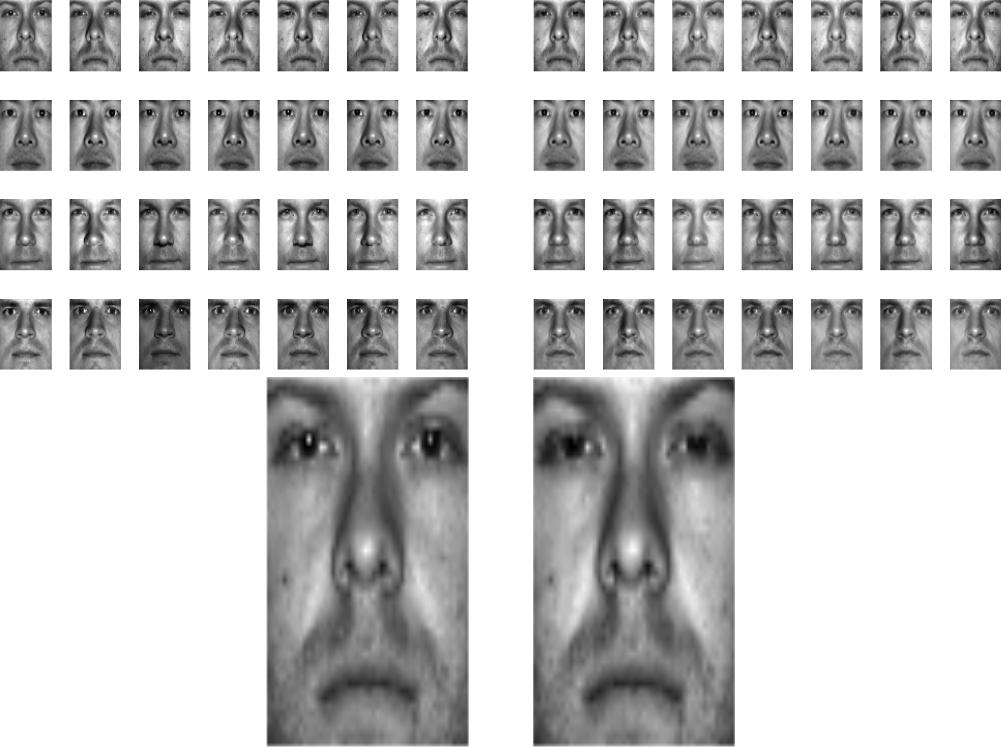
(1= 2; 1= 2). When projecting a point (x; x + y) on this principal component

2x+y

we will obtain the scalar p2 . The reconstruction of the original vector will be ((x + y=2); (x + y=2)). In Figure [23.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page327) we depict the original versus reconstructed data.

Next, we demonstrate the e ectiveness of PCA on a data set of faces. We extracted images of faces from the Yale data set (Georghiades, Belhumeur & Kriegman 2001). Each image contains 50 50 = 2500 pixels; therefore the original dimensionality is very high.

1. Dimensionality Reduction



|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | o |  |  | o | oo |  |  | o |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  | oo | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | x | x x | x xx | |  |  |  |  |  |
| + | |  |  |  | x |  |  |  |  |  |
| + |  |  |  |  |  |  |  |  |  |  |
| ++ + | | |  |  |  | \* |  |  |  |  | \* \* |  |
| ++ | | |  |  |  |  |  |  |  | \*\* |
|  |  |  |  |  |  |  |  |  |  |  | \* |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Figure 23.2 Images of faces extracted from the Yale data set. Top-Left: the original

images in R50x50. Top-Right: the images after dimensionality reduction to R10 and

reconstruction. Middle row: an enlarged version of one of the images before and after

PCA. Bottom: The images after dimensionality reduction to R2. The di erent marks indicate di erent individuals.

Some images of faces are depicted on the top-left side of Figure [23.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page328). Using PCA, we reduced the dimensionality to R10 and reconstructed back to the orig-inal dimension, which is 502. The resulting reconstructed images are depicted on the top-right side of Figure [23.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page328). Finally, on the bottom of Figure [23.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page328) we depict a 2 dimensional representation of the images. As can be seen, even from a 2 dimensional representation of the images we can still roughly separate di erent individuals.

|  |  |
| --- | --- |
| 23.2 Random Projections | 329 |
|  |  |

23.2 Random Projections

In this section we show that reducing the dimension by using a random linear transformation leads to a simple compression scheme with a surprisingly low distortion. The transformation x 7!W x, when W is a random matrix, is often referred to as a random projection. In particular, we provide a variant of a famous lemma due to Johnson and Lindenstrauss, showing that random projections do not distort Euclidean distances too much.

Let x1; x2 be two vectors in Rd. A matrix W does not distort too much the distance between x1 and x2 if the ratio

kW x1 W x2k

kx1 x2k

is close to 1. In other words, the distances between x1 and x2 before and after the transformation are almost the same. To show that kW x1 W x2k is not too far away from kx1 x2k it su ces to show that W does not distort the norm of the di erence vector x = x1 x2. Therefore, from now on we focus on the ratio

kW xk .

kxk

We start with analyzing the distortion caused by applying a random projection to a single vector.

lemma 23.3 Fix some x 2 Rd. Let W 2 Rn;d be a random matrix such that each Wi;j is an independent normal random variable. Then, for every 2 (0; 3) we have

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P " k(1= | | x | | | 2 | k |  | 1 > # | 2 e | n=6: |
|  | | p |  | | |  | 2 |  |  |  |
| n | )W x | |  | 2 |  |
| k | k | |  |  |  |  |  |
|  | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | |  |  |  |  |  |  |  |  |  |

Proof Without loss of generality we can assume that kxk2 = 1. Therefore, an equivalent inequality is

P (1 )n kW xk2 (1 + )n 1 2e 2n=6:

Let wi be the ith row of W . The random variable hwi; xi is a weighted sum of d independent normal random variables and therefore it is normally distributed with zero mean and variance P x2 = kxk2 = 1. Therefore, the random vari-

j j

able kW xk2 = Pni=1(hwi; xi)2 has a 2n distribution. The claim now follows directly from a measure concentration property of 2 random variables stated in Lemma [B.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page428) given in Section [B.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page428). 

The Johnson-Lindenstrauss lemma follows from this using a simple union bound argument.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| lemma 23.4 (Johnson-Lindenstrauss Lemma) | | | | | | Let Q be a nite set of vectors |
| in Rd. Let 2 (0; 1) and n be an integer such that | | | | | | |
| = r |  |  |  |  |  | 3: |
| 6 | | nj j | = ) | |
|  |  |  | log(2 Q |  |
|  |  |  |  |  |  |  |

1. Dimensionality Reduction

Then, with probability of at least 1 over a choice of a random matrix W 2 Rn;d such that each element of W is distributed normally with zero mean and variance of 1=n we have

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | W x | | | 2 |  |  |  |  |
| x | Q | k |  | x | k |  |  |  |  |
|  | k | 2 |  |  |  |
| 2 |  |  |  | k |  |  |  | 1 |  | < : |
| sup | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

Proof Combining Lemma [23.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page329) and the union bound we have that for every

2 (0; 3):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | x2Q | | k | W x | 2 |  |  |  |  |  |  | j | j |  |  | 2 |
| x 2k |  |  |  |  |  |
|  | sup |  |  | k k |  |  | 1 |  | > |  | 2 |  | Q | e |  | n=6: |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Let denote the right-hand side of the inequality; thus we obtain that

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| = r |  |  |  |  |  |  |
| 6 | | nj j | : | | |
|  |  |  | log(2 Q | = ) | | |
|  |  |  |  |  |  |  |

Interestingly, the bound given in Lemma [23.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page329) does not depend on the original dimension of x. In fact, the bound holds even if x is in an in nite dimensional Hilbert space.

23.3 Compressed Sensing

Compressed sensing is a dimensionality reduction technique which utilizes a prior assumption that the original vector is sparse in some basis. To motivate com-pressed sensing, consider a vector x 2 Rd that has at most s nonzero elements. That is,

def

kxk0 = jfi : xi 6= 0gj s:

Clearly, we can compress x by representing it using s (index,value) pairs. Fur-thermore, this compression is lossless { we can reconstruct x exactly from the s (index,value) pairs. Now, lets take one step forward and assume that x = U , where is a sparse vector, k k0 s, and U is a xed orthonormal matrix. That is, x has a sparse representation in another basis. It turns out that many nat-ural vectors are (at least approximately) sparse in some representation. In fact, this assumption underlies many modern compression schemes. For example, the JPEG-2000 format for image compression relies on the fact that natural images are approximately sparse in a wavelet basis.

Can we still compress x into roughly s numbers? Well, one simple way to do this is to multiply x by U>, which yields the sparse vector , and then represent

by its s (index,value) pairs. However, this requires us rst to \sense" x, to

store it, and then to multiply it by U>. This raises a very natural question: Why go to so much e ort to acquire all the data when most of what we get will be thrown away? Cannot we just directly measure the part that will not end up being thrown away?

|  |  |
| --- | --- |
| 23.3 Compressed Sensing | 331 |
|  |  |

Compressed sensing is a technique that simultaneously acquires and com-presses the data. The key result is that a random linear transformation can compress x without losing information. The number of measurements needed is order of s log(d). That is, we roughly acquire only the important information about the signal. As we will see later, the price we pay is a slower reconstruction phase. In some situations, it makes sense to save time in compression even at the price of a slower reconstruction. For example, a security camera should sense and compress a large amount of images while most of the time we do not need to decode the compressed data at all. Furthermore, in many practical applications, compression by a linear transformation is advantageous because it can be per-formed e ciently in hardware. For example, a team led by Baraniuk and Kelly has proposed a camera architecture that employs a digital micromirror array to perform optical calculations of a linear transformation of an image. In this case, obtaining each compressed measurement is as easy as obtaining a single raw measurement. Another important application of compressed sensing is medical imaging, in which requiring fewer measurements translates to less radiation for the patient.

Informally, the main premise of compressed sensing is the following three \sur-prising" results:

1. It is possible to reconstruct any sparse signal fully if it was compressed by

x 7!W x, where W is a matrix which satis es a condition called the Re-stricted Isoperimetric Property (RIP). A matrix that satis es this property is guaranteed to have a low distortion of the norm of any sparse representable vector.

1. The reconstruction can be calculated in polynomial time by solving a linear program.
2. A random n d matrix is likely to satisfy the RIP condition provided that n is greater than an order of s log(d).

Formally,

definition 23.5 (RIP) A matrix W 2 Rn;d is ( ; s)-RIP if for all x 6= 0 s.t.

kxk0 s we have

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | kW xk22 | | | |  | 1 |  |  | : |
| k | x | k | 22 |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

The rst theorem establishes that RIP matrices yield a lossless compression scheme for sparse vectors. It also provides a (none cient) reconstruction scheme.

theorem 23.6 Let < 1 and let W be a ( ; 2s)-RIP matrix. Let x be a vector s.t. kxk0 s, let y = W x be the compression of x, and let

x~ 2 argmin kvk0

v:W v=y

be a reconstructed vector. Then, x~ = x.

1. Dimensionality Reduction

Proof We assume, by way of contradiction, that x~ 6= x. Since x satis es the constraints in the optimization problem that de nes x~ we clearly have that

kx~k0 kxk0 s. Therefore, kx x~k0 2s and we can apply the RIP in-equality on the vector x x~. But, since W (x x~) = 0 we get that j0 1j , which leads to a contradiction. 

The reconstruction scheme given in Theorem [23.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page331) seems to be none cient because we need to minimize a combinatorial objective (the sparsity of v). Quite surprisingly, it turns out that we can replace the combinatorial objective, kvk0, with a convex objective, kvk1, which leads to a linear programming problem that can be solved e ciently. This is stated formally in the following theorem.

theorem 23.7 Assume that the conditions of Theorem [23.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page331) holds and that

|  |  |  |
| --- | --- | --- |
| < | 1 | . Then, |
| 1+p2 |

1. = argmin kvk0 = argmin kvk1:

v:W v=y v:W v=y

In fact, we will prove a stronger result, which holds even if x is not a sparse vector.

|  |  |  |
| --- | --- | --- |
| theorem 23.8 Let < | 1 | and let W be a ( ; 2s)-RIP matrix. Let x be an |
| 1+p2 |

arbitrary vector and denote

xs 2 argmin kx vk1:

v:kvk0 s

That is, xs is the vector which equals x on the s largest elements of x and equals 0 elsewhere. Let y = W x be the compression of x and let

x? 2 argmin kvk1

v:W v=y

be the reconstructed vector. Then,

kx? xk2 2 11 + s 1=2kx xsk1;

p

where = 2 =(1 ).

Note that in the special case that x = xs we get an exact recovery, x? = x, so Theorem [23.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page332) is a special case of Theorem [23.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page332). The proof of Theorem [23.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page332) is given in Section [23.3.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page333).

Finally, the third result tells us that random matrices with n (s log(d)) are likely to be RIP. In fact, the theorem shows that multiplying a random matrix by an orthonormal matrix also provides an RIP matrix. This is important for compressing signals of the form x = U where x is not sparse but is sparse. In that case, if W is a random matrix and we compress using y = W x then this is the same as compressing by y = (W U) and since W U is also RIP we can reconstruct (and thus also x) from y.

|  |  |
| --- | --- |
| 23.3 Compressed Sensing | 333 |
|  |  |

theorem 23.9 Let U be an arbitrary xed d d orthonormal matrix, let ; be scalars in (0; 1), let s be an integer in [d], and let n be an integer that satis es

1. 100 s log(40d=( )) : 2

Let W 2 Rn;d be a matrix s.t. each element of W is distributed normally with zero mean and variance of 1=n. Then, with proabability of at least 1 over the choice of W , the matrix W U is ( ; s)-RIP.

23.3.1 Proofs\*

Proof of Theorem [23.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page332)

We follow a proof due to Candes (2008).

Let h = x? x. Given a vector v and a set of indices I we denote by vI the vector whose ith element is vi if i 2 I and 0 otherwise.

The rst trick we use is to partition the set of indices [d] = f1; : : : ; dg into disjoint sets of size s. That is, we will write [d] = T0 [ T1 [ T2 : : : Td=s 1 where for all i, jTij = s, and we assume for simplicity that d=s is an integer. We de ne the partition as follows. In T0 we put the s indices corresponding to the s largest elements in absolute values of x (ties are broken arbitrarily). Let T0c = [d] n T0. Next, T1 will be the s indices corresponding to the s largest elements in absolute

value of hT0c . Let T0;1 = T0 [ T1 and T0c;1 = [d] n T0;1. Next, T2 will correspond to the s largest elements in absolute value of hT0c;1 . And, we will construct T3; T4; : : : in the same way.

To prove the theorem we rst need the following lemma, which shows that RIP also implies approximate orthogonality.

lemma 23.10 Let W be an ( ; 2s)-RIP matrix. Then, for any two disjoint sets I; J, both of size at most s, and for any vector u we have that hW uI ; W uJ i kuI k2 kuJ k2.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Proof W.l.o.g. assume kuI k2 = kuJ k2 = 1. | | | | | |  |  |
| h | W u | ; W u | J i | = | kW uI + W uJ k22 | kW uI W uJ k22 | : |
| I |  |  |  | 4 |  |

But, since jJ [ Ij 2s we get from the RIP condition that kW uI + W uJ k22 (1 + )(kuI k22 +kuJ k22) = 2(1 + ) and that kW uI W uJ k22 (1 )(kuI k22 + kuJ k22) = 2(1 ), which concludes our proof. 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| We are now ready to prove the theorem. Clearly, | | | |  |
|  |  | khk2 = khT0;1 + hT0c;1 k2 khT0;1 k2 + khT0c;1 k2: | | (23.5) |
| To prove the theorem we will show the following two claims: | | | |  |
| Claim 1:. khT0c;1 k2 | khT0 k2 + 2s 1=2kx xsk1. | | |  |
| Claim 2:. khT0;1 k2 |  | 2 | s 1=2kx xsk1. |  |
| 1 |  |

1. Dimensionality Reduction

Combining these two claims with Equation ([23.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page333)) we get that

khk2 khT0;1 k2 + khT0c;1 k2 2khT0;1 k2 + 2s 1=2kx xsk1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 + 1 | | | s 1=2kx xsk1 |
|  |  | 2 |  |  |

= 2 11 + s 1=2kx xsk1;

and this will conclude our proof.

Proving Claim 1:

To prove this claim we do not use the RIP condition at all but only use the fact that x? minimizes the `1 norm. Take j > 1. For each i 2 Tj and i0 2 Tj 1 we have that jhij jhi0j. Therefore, khTj k1 khTj 1 k1=s. Thus,

khTj k2 s1=2khTj k1 s 1=2khTj 1 k1:

Summing this over j = 2; 3; : : : and using the triangle inequality we obtain that

|  |  |
| --- | --- |
| X |  |
| khT0c;1 k2khTj k2 s 1=2khT0c k1 | (23.6) |

j 2

Next, we show that khT0c k1 cannot be large. Indeed, from the de nition of x? we have that kxk1 kx?k1 = kx + hk1. Thus, using the triangle inequality we obtain that

X X

kxk1 kx+hk1 = jxi+hij+ jxi+hij kxT0 k1 khT0 k1+khT0c k1 kxT0c k1

i2T0 i2T0c

(23.7)

|  |  |  |
| --- | --- | --- |
| and since kxT0c k1 = kx xsk1 = kxk1 kxT0 k1 we get that | |  |
| khT0c k1 khT0 k1 + 2kxT0c k1: | | (23.8) |
| Combining this with Equation ([23.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page334)) we get that | |  |
| khT0c;1 k2 s 1=2 khT0 k1 + 2kxT0c k1 | khT0 k2 + 2s 1=2kxT0c k1; |  |
| which concludes the proof of claim 1. |  |  |

Proving Claim 2:

For the second claim we use the RIP condition to get that

|  |  |  |  |
| --- | --- | --- | --- |
|  | (1 )khT0;1 k22 kW hT0;1 k22: | | (23.9) |
| Since W hT0;1 =2 | W h Pj 2 W hTj | = Pj 2 W hTj we have that |  |
|  | X | X |  |
| kW hT0;1 k2 =hW hT0;1 ; W hTj i =hW hT0 + W hT1 ; W hTj i: | | | |
|  | j 2 | j 2 |  |

From the RIP condition on inner products we obtain that for all i 2 f1; 2g and j 2 we have

jhW hTi ; W hTj ij khTi k2khTj k2:

|  |  |
| --- | --- |
| 23.3 Compressed Sensing | 335 |
|  |  |

p

Since khT0 k2 + khT1 k2 2khT0;1 k2 we therefore get that

1. X

kW hT0;1 k22 2 khT0;1 k2 khTj k2:

j 2

Combining this with Equation ([23.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page334)) and Equation ([23.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page334)) we obtain

p

(1 )khT0;1 k22 2 khT0;1 k2s 1=2khT0c k1:

Rearranging the inequality gives

p

khT0;1 k2 1 2 s 1=2khT0c k1:

Finally, using Equation ([23.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page334)) we get that

khT0;1 k2 s 1=2 (khT0 k1 + 2kxT0c k1) khT0 k2 + 2 s 1=2kxT0c k1; but since khT0 k2 khT0;1 k2 this implies

khT0;1 k2 12 s 1=2kxT0c k1;

which concludes the proof of the second claim.

Proof of Theorem [23.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page332)

To prove the theorem we follow an approach due to (Baraniuk, Davenport, De-Vore & Wakin 2008). The idea is to combine the Johnson-Lindenstrauss (JL) lemma with a simple covering argument.

We start with a covering property of the unit ball.

lemma 23.11 Let 2 (0; 1). There exists a nite set Q Rd of size jQj 3 d such that

sup min kx vk :

x:kxk 1 v2Q

Proof Let k be an integer and let

Q0 = fx 2 Rd : 8j 2 [d]; 9i 2 f k; k + 1; : : : ; kg s:t: xj = ki g:

Clearly, jQ0j = (2k + 1)d. We shall set Q = Q0 \ B2(1), where B2(1) is the unit `2 ball of Rd. Since the points in Q0 are distributed evenly on the unit `1 ball, the size of Q is the size of Q0 times the ratio between the volumes of the unit `2 and `1 balls. The volume of the `1 ball is 2d and the volume of B2(1) is

d=2

(1 + d=2) :

For simplicity, assume that d is even and therefore

d=2 d=2

(1 + d=2) = (d=2)! ;

e

:k k=1 v2Q

1. Dimensionality Reduction

where in the last inequality we used Stirling's approximation. Overall we obtained that

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q | j | (2k + 1)d ( =e)d=2 | (d=2) d=2 | 2 d: | (23.10) |
| j |  |  |  |  |

Now lets specify k. For each x 2 B2(1) let v 2 Q be the vector whose ith element is sign(xi) bjxij kc=k. Then, for each element we have that jxi vij 1=k and

thus

p

kx vk kd :

p

To ensure that the right-hand side will be at most we shall set k = d d= e. Plugging this value into Equation ([23.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page336)) we conclude that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | d |  |  |  |  |  |
| jQj (3p |  |  | d |  | d=2 | (d=2) | d=2 |  |  | | q2e | | | | |  | |  | d |  |
| d=(2 )) | | ( =e) | = |  | : |
|  |  |  |  |  |  | 3 | |  |  |  | | |  | 3 | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Let x be a vector that can be written as x = U with U being some orthonor-mal matrix and k k0 s. Combining the earlier covering property and the JL lemma (Lemma [23.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page329)) enables us to show that a random W will not distort any such x.

lemma 23.12 Let U be an orthonormal d d matrix and let I [d] be a set of indices of size jIj = s. Let S be the span of fUi : i 2 Ig, where Ui is the ith column of U. Let 2 (0; 1), 2 (0; 1), and n 2 N such that

1. 24 log(2= ) + s log(12= ) : 2

Then, with probability of at least 1 over a choice of a random matrix W 2 Rn;d such that each element of W is independently distributed according to N(0; 1=n), we have

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x S |  | k | k | x | k |  |  |  |  |
| 2 |  |  |  | k |  | 1 |  | < : |
| sup |  |  | W x | | |  |  |
|  |  |  |  |  |  |  |  |  |  |

Proof It su ces to prove the lemma for all x 2 S with kxk = 1. We can write x = UI where 2 Rs, k k2 = 1, and UI is the matrix whose columns are fUi : i 2 Ig. Using Lemma [23.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page335) we know that there exists a set Q of size jQj (12= )s such that

sup min k vk ( =4):

:k k=1 v2Q

But since U is orthogonal we also have that

sup min kUI UI vk ( =4):

Applying Lemma [23.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page329) on the set fUI v : v 2 Qg we obtain that for n satisfying

|  |  |
| --- | --- |
| 23.3 Compressed Sensing | 337 |
|  |  |

the condition given in the lemma, the following holds with probability of at least 1 :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sup | | | | |  |  |  | W UI v 2 | | | | | | | |  | 1 |  |  | =2; |
| k | | | k | UI v | | | k | | 2k | |  |  |  |
| v | 2 | | Q | |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| This also implies that |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sup | | | | |  |  |  | W UI v | | | | | | |  |  | 1 |  |  | =2: |
|  |  | k | |  | UI v | | |  | k | |  |  |  |
| v | | 2 | | Q | | |  |  |  | |  |
|  |  |  |  | |  | k | |  |  |  | k | | |  |  | |  |  |
|  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  | |  |  |
| Let a be the smallest number such | | | | | | | | | | | that | | | | |  |  | |  |  |
|  |  | x | | | 2 | | | S; | | | kW xk | | | | | |  | | 1 + a: | |
| 8 | | |  |  |  |  |  |  | kxk | | | | |  |  |

Clearly a < 1. Our goal is to show that a . This follows from the fact that for any x 2 S of unit norm there exists v 2 Q such that kx UI vk =4 and therefore

kW xk kW UI vk + kW (x UI v)k 1 + =2 + (1 + a) =4:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Thus, |  |  |  |  |  |  |
| 8 |  | 2 |  | kxk |  |  |
|  | x |  | S; | kW xk |  | 1 + ( =2 + (1 + a) =4) : |
|  |  |  |  |

But the de nition of a implies that

a =2 + (1 + a) =4

This proves that for all x 2 S we have this as well since

kW xk kW UI vk kW (x UI

1. a =2 + =4 : 1 =4

kW xk 1 . The other side follows from

kxk

v)k 1 =2 (1 + ) =4 1 :

The preceding lemma tells us that for x 2 S of unit norm we have

(1 ) kW xk (1 + );

which implies that

(1 2 ) kW xk2 (1 + 3 ):

The proof of Theorem [23.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page332) follows from this by a union bound over all choices of I.

1. Dimensionality Reduction

23.4 PCA or Compressed Sensing?

Suppose we would like to apply a dimensionality reduction technique to a given set of examples. Which method should we use, PCA or compressed sensing? In this section we tackle this question, by underscoring the underlying assumptions behind the two methods.

It is helpful rst to understand when each of the methods can guarantee per-fect recovery. PCA guarantees perfect recovery whenever the set of examples is contained in an n dimensional subspace of Rd. Compressed sensing guarantees perfect recovery whenever the set of examples is sparse (in some basis). On the basis of these observations, we can describe cases in which PCA will be better than compressed sensing and vice versa.

As a rst example, suppose that the examples are the vectors of the standard basis of Rd, namely, e1; : : : ; ed, where each ei is the all zeros vector except 1 in the ith coordinate. In this case, the examples are 1-sparse. Hence, compressed sensing will yield a perfect recovery whenever n (log(d)). On the other hand, PCA will lead to poor performance, since the data is far from being in an n dimensional subspace, as long as n < d. Indeed, it is easy ro verify that in such a case, the averaged recovery error of PCA (i.e., the objective of Equation ([23.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page324)) divided by m) will be (d n)=d, which is larger than 1=2 whenever n d=2.

We next show a case where PCA is better than compressed sensing. Consider m examples that are exactly on an n dimensional subspace. Clearly, in such a case, PCA will lead to perfect recovery. As to compressed sensing, note that the examples are n-sparse in any orthonormal basis whose rst n vectors span the subspace. Therefore, compressed sensing would also work if we will reduce the dimension to (n log(d)). However, with exactly n dimensions, compressed sensing might fail. PCA has also better resilience to certain types of noise. See (Chang, Weiss & Freeman 2009) for a discussion.

23.5 Summary

We introduced two methods for dimensionality reduction using linear transfor-mations: PCA and random projections. We have shown that PCA is optimal in the sense of averaged squared reconstruction error, if we restrict the reconstruc-tion procedure to be linear as well. However, if we allow nonlinear reconstruction, PCA is not necessarily the optimal procedure. In particular, for sparse data, ran-dom projections can signi cantly outperform PCA. This fact is at the heart of the compressed sensing method.

|  |  |
| --- | --- |
| 23.6 Bibliographic Remarks | 339 |
|  |  |

23.6 Bibliographic Remarks

PCA is equivalent to best subspace approximation using singular value decom-position (SVD). The SVD method is described in Appendix [C](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page430). SVD dates back to Eugenio Beltrami (1873) and Camille Jordan (1874). It has been rediscovered many times. In the statistical literature, it was introduced by Pearson (1901). Be-sides PCA and SVD, there are additional names that refer to the same idea and are being used in di erent scienti c communities. A few examples are the Eckart-Young theorem (after Carl Eckart and Gale Young who analyzed the method in 1936), the Schmidt-Mirsky theorem, factor analysis, and the Hotelling transform.

Compressed sensing was introduced in Donoho (2006) and in (Candes & Tao 2005). See also Candes (2006).

23.7 Exercises

1. In this exercise we show that in the general case, exact recovery of a linear compression scheme is impossible.
   1. let A 2 Rn;d be an arbitrary compression matrix where n d 1. Show that there exists u; v 2 Rn, u 6= v such that Au = Av.
   2. Conclude that exact recovery of a linear compression scheme is impossible.
2. Let 2 Rd such that 12d 0. Show that

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | k k | d |  | n |
| X | j j = | Xj |
|  | max |  | j: |
| [0;1]d: 1 | | n |  | =1 |
|  |  | j=1 |  |

Hint: Take every vector 2 [0; 1]d such that k k1 n. Let i be the minimal index for which i < 1. If i = n + 1 we are done. Otherwise, show that we can increase i, while possibly decreasing j for some j > i, and obtain a better solution. This will imply that the optimal solution is to set i = 1 for i n and i = 0 for i > n.

1. Kernel PCA: In this exercise we show how PCA can be used for construct-ing nonlinear dimensionality reduction on the basis of the kernel trick (see Chapter [16](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page215)).

Let X be some instance space and let S = fx1; : : : ; xmg be a set of points

in X . Consider a feature mapping : X ! V , where V is some Hilbert space

(possibly of in nite dimension). Let K : X X be a kernel function, that is,

k(x; x0) = h (x); (x0)i. Kernel PCA is the process of mapping the elements in S into V using , and then applying PCA over f (x1); : : : ; (xm)g into

Rn. The output of this process is the set of reduced elements.

Show how this process can be done in polynomial time in terms of m and n, assuming that each evaluation of K( ; ) can be calculated in a con-stant time. In particular, if your implementation requires multiplication of two matrices A and B, verify that their product can be computed. Similarly,

1. Dimensionality Reduction

if an eigenvalue decomposition of some matrix C is required, verify that this decomposition can be computed.

1. An Interpretation of PCA as Variance Maximization:

Let x1; : : : ; xm be m vectors in Rd, and let x be a random vector distributed

according to the uniform distribution over x1; : : : ; xm. Assume that E[x] = 0.

1. Consider the problem of nding a unit vector, w 2 Rd, such that the random variable hw; xi has maximal variance. That is, we would like to solve the problem

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 1 | m |
| k k | k k |  | Xi |
| argmax Var[hw; xi] = argmax | | m | (hw; xii)2: |
| w: w =1 | w: w =1 | | =1 |
|  |  |  |

Show that the solution of the problem is to set w to be the rst principle vector of x1; : : : ; xm.

1. Let w1 be the rst principal component as in the previous question. Now, suppose we would like to nd a second unit vector, w2 2 Rd, that maxi-mizes the variance of hw2; xi, but is also uncorrelated to hw1; xi. That is, we would like to solve:

argmax Var[hw; xi]:

w:kwk=1; E[(hw1;xi)(hw;xi)]=0

Show that the solution to this problem is to set w to be the second principal component of x1; : : : ; xm.

Hint: Note that

E[(hw1; xi)(hw; xi)] = w1> E[xx>]w = mw1>Aw;

where A = Pi xix>i. Since w is an eigenvector of A we have that the constraint E[(hw1; xi)(hw; xi)] = 0 is equivalent to the constraint

hw1; wi = 0:

1. The Relation between SVD and PCA: Use the SVD theorem (Corol-lary [C.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page433)) for providing an alternative proof of Theorem [23.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page325).
2. Random Projections Preserve Inner Products: The Johnson-Lindenstrauss lemma tells us that a random projection preserves distances between a nite set of vectors. In this exercise you need to prove that if the set of vectors are within the unit ball, then not only are the distances between any two vectors preserved, but the inner product is also preserved.

Let Q be a nite set of vectors in Rd and assume that for every x 2 Q we have kxk 1.

1. Let 2 (0; 1) and n be an integer such that

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| = | r |  |  |  |  |  | 3: |
| 6 log(jn j | | 2 | = ) | |
|  |  |  | Q |  |
|  |  |  |  |  |  |  |  |

Prove that with probability of at least 1 over a choice of a random

|  |  |
| --- | --- |
| 23.7 Exercises | 341 |
|  |  |

matrix W 2 Rn;d, where each element of W is independently distributed according to N (0; 1=n), we have

jhW u; W vi hu; vij

for every u; v 2 Q.

Hint: Use JL to bound both kW (u+v)k and kW (u v)k .

ku+vk ku vk

1. (\*) Let x1; : : : ; xm be a set of vectors in Rd of norm at most 1, and assume that these vectors are linearly separable with margin of . Assume that

d 1= 2. Show that there exists a constant c > 0 such that if we randomly project these vectors into Rn, for n = c= 2, then with probability of at least 99% it holds that the projected vectors are linearly separable with margin =2.

1. Generative Models

We started this book with a distribution free learning framework; namely, we did not impose any assumptions on the underlying distribution over the data. Furthermore, we followed a discriminative approach in which our goal is not to learn the underlying distribution but rather to learn an accurate predictor. In this chapter we describe a generative approach, in which it is assumed that the underlying distribution over the data has a speci c parametric form and our goal is to estimate the parameters of the model. This task is called parametric density estimation.

The discriminative approach has the advantage of directly optimizing the quantity of interest (the prediction accuracy) instead of learning the underly-ing distribution. This was phrased as follows by Vladimir Vapnik in his principle for solving problems using a restricted amount of information:

When solving a given problem, try to avoid a more general problem as an intermediate step.

Of course, if we succeed in learning the underlying distribution accurately, we are considered to be \experts" in the sense that we can predict by using the Bayes optimal classi er. The problem is that it is usually more di cult to learn the underlying distribution than to learn an accurate predictor. However, in some situations, it is reasonable to adopt the generative learning approach. For example, sometimes it is easier (computationally) to estimate the parameters of the model than to learn a discriminative predictor. Additionally, in some cases we do not have a speci c task at hand but rather would like to model the data either for making predictions at a later time without having to retrain a predictor or for the sake of interpretability of the data.

We start with a popular statistical method for estimating the parameters of the data, which is called the maximum likelihood principle. Next, we describe two generative assumptions which greatly simplify the learning process. We also de-scribe the EM algorithm for calculating the maximum likelihood in the presence of latent variables. We conclude with a brief description of Bayesian reasoning.

|  |  |
| --- | --- |
| 24.1 Maximum Likelihood Estimator | 343 |
|  |  |

24.1 Maximum Likelihood Estimator

Let us start with a simple example. A drug company developed a new drug to treat some deadly disease. We would like to estimate the probability of survival when using the drug. To do so, the drug company sampled a training set of m people and gave them the drug. Let S = (x1; : : : ; xm) denote the training set, where for each i, xi = 1 if the ith person survived and xi = 0 otherwise. We can model the underlying distribution using a single parameter, 2 [0; 1], indicating the probability of survival.

We now would like to estimate the parameter on the basis of the training set S. A natural idea is to use the average number of 1's in S as an estimator. That is,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ^ | 1 |  | m |  |
|  | = m | | | Xi | (24.1) |
|  | xi: |
|  |  |  |  | =1 |  |
| ^ | ^ |  |  |  | ^ |
| Clearly, ES[ ] = . That is, is an unbiased estimator of . Furthermore, since is | | | | | |

the average of m i.i.d. binary random variables we can use Hoe ding's inequality to get that with probability of at least 1 over the choice of S we have that

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j^ j | r | log(2= ) | : | (24.2) |
| 2 m |

^

Another interpretation of is as the Maximum Likelihood Estimator, as we formally explain now. We rst write the probability of generating the sample S:

|  |  |
| --- | --- |
| m |  |
| iY | Pi xi (1)Pi(1 xi): |
| P[S = (x1; : : : ; xm)] = xi (1 )1 xi = |
| =1 |  |

We de ne the log likelihood of S, given the parameter , as the log of the preceding expression:

X X

L(S; ) = log (P[S = (x1; : : : ; xm)]) = log( ) xi + log(1 ) (1 xi):

i i

The maximum likelihood estimator is the parameter that maximizes the likeli-hood

|  |  |  |
| --- | --- | --- |
| 2 |  | (24.3) |
| ^ | argmax L(S; ): |

Next, we show that in our case, Equation ([24.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page343)) is a maximum likelihood esti-mator. To see this, we take the derivative of L(S; ) with respect to and equate it to zero:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pi xi | Pi1 |  | i | = 0: |
|  | (1 |  | x ) |  |

Solving the equation for we obtain the estimator given in Equation ([24.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page343)).

1. Generative Models

24.1.1 Maximum Likelihood Estimation for Continuous Random Variables

Let X be a continuous random variable. Then, for most x 2 R we have P[X = x] = 0 and therefore the de nition of likelihood as given before is trivialized. To overcome this technical problem we de ne the likelihood as log of the density of the probability of X at x. That is, given an i.i.d. training set S = (x1; : : : ; xm) sampled according to a density distribution P we de ne the likelihood of S given

as

!

m m

Y

L(S; ) = log P (xi)

X

1. log(P (xi)):

i=1 i=1

As before, the maximum likelihood estimator is a maximizer of L(S; ) with respect to .

As an example, consider a Gaussian random variable, for which the density function of X is parameterized by = ( ; ) and is de ned as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P | (x) = | 1 | | exp |  | (x )2 | : |
|  |  | 2 2 |
|  | p2 | | |  |

We can rewrite the likelihood as

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | m |  |  |  |  |  |
|  | 1 | Xi | 2 | m log( | p |  |  |
|  |
| L(S; ) = 2 2 | |  |  | 2 ): | |
| (xi ) |  |  |
|  |  | =1 |  |  |  |  |  |

To nd a parameter = ( ; ) that optimizes this we take the derivative of the likelihood w.r.t. and w.r.t. and compare it to 0. We obtain the following two equations:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| d | | 1 |  | m |  |  |
|  |  |  |  | Xi |  |  |
| d L(S; ) = 2 | | |  |  |
|  | (xi ) = 0 | |  |
|  |  |  | =1 | |  |  |
| d | | 1 |  | m | m |  |
|  | L(S; ) = |  |  | (xi )2 |  | = 0 |
| d | 3 |  |  |
|  |  |  | =1 | |  |  |
|  |  |  |  | Xi |  |  |

Solving the preceding equations we obtain the maximum likelihood estimates:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | |  |  | v |  |  |  |  |  |
| ^ = |  | 1 m | | xi | and^ = | 1 | | m (xi |  | ^)2 |
|  |  |  |  |  |
| m i=1 | | | m | |
|  |  |  | u | i=1 |  |
|  |  |  | X |  |  | u |  |  | X |  |  |
|  |  |  |  |  |  | t |  |  |  |  |  |

Note that the maximum likelihood estimate is not always an unbiased estimator. For example, while ^ is unbiased, it is possible to show that the estimate ^ of the variance is biased (Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page356)).

Simplifying Notation

To simplify our notation, we use P[X = x] in this chapter to describe both the probability that X = x (for discrete random variables) and the density of the distribution at x (for continuous variables).

|  |  |
| --- | --- |
| 24.1 Maximum Likelihood Estimator | 345 |
|  |  |

24.1.2 Maximum Likelihood and Empirical Risk Minimization

The maximum likelihood estimator shares some similarity with the Empirical Risk Minimization (ERM) principle, which we studied extensively in previous chapters. Recall that in the ERM principle we have a hypothesis class H and we use the training set for choosing a hypothesis h 2 H that minimizes the empirical risk. We now show that the maximum likelihood estimator is an ERM for a particular loss function.

Given a parameter and an observation x, we de ne the loss of on x as

|  |  |
| --- | --- |
| `( ; x) = log(P [x]): | (24.4) |

That is, `( ; x) is the negation of the log-likelihood of the observation x, assuming the data is distributed according to P . This loss function is often referred to as the log-loss. On the basis of this de nition it is immediate that the maximum likelihood principle is equivalent to minimizing the empirical risk with respect to the loss function given in Equation ([24.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page345)). That is,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | m |  |  | m |  |  |  |
| argmin | Xi | ( log(P [xi])) = | argmax | X | P | [x | ]): |
|  | log( |
|  | =1 |  | i=1 | i |  |
|  |  |  |  |  |  |

Assuming that the data is distributed according to a distribution P (not neces-sarily of the parametric form we employ), the true risk of a parameter becomes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x |  | X | |  |  |  | P | | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | P |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| E[`( ; x)] = |  |  |  | [x] log( | | | | [x]) | | |  |  |  |  |  |  |  |  |  |  |  |  |
| = |  |  | x |  |  | P[[x] | | | | | + |  |  | P[x] log | | |  | [x] | | | ; | (24.5) |
|  | x | P[x] log | | |  | x |  |
|  | X | |  |  |  |  |  | x] | | |  | X | |  |  |  |  | 1 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | P |  |  |  |  |  |  |  | P |  |  |  |  |  |
|  | | |  |  |  | {z |  |  |  | | } |  | | |  |  | {z |  |  |  | | } |  |  |
|  |  |  | DRE[PjjP ] | | | | |  |  |  |  |  |  | H(P) | | |  |  |  |  |  |  |

where DRE is called the relative entropy, and H is called the entropy func-tion. The relative entropy is a divergence measure between two probabilities. For discrete variables, it is always nonnegative and is equal to 0 only if the two distributions are the same. It follows that the true risk is minimal when P = P.

The expression given in Equation ([24.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page345)) underscores how our generative as-sumption a ects our density estimation, even in the limit of in nite data. It shows that if the underlying distribution is indeed of a parametric form, then by choosing the correct parameter we can make the risk be the entropy of the distri-bution. However, if the distribution is not of the assumed parametric form, even the best parameter leads to an inferior model and the suboptimality is measured by the relative entropy divergence.

24.1.3 Generalization Analysis

How good is the maximum likelihood estimator when we learn from a nite training set?

1. Generative Models

To answer this question we need to de ne how we assess the quality of an approxi-mated solution of the density estimation problem. Unlike discriminative learning, where there is a clear notion of \loss," in generative learning there are various ways to de ne the loss of a model. On the basis of the previous subsection, one natural candidate is the expected log-loss as given in Equation ([24.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page345)).

In some situations, it is easy to prove that the maximum likelihood principle guarantees low true risk as well. For example, consider the problem of estimating

the mean of a Gaussian variable of unit variance. We saw previously that the maximum likelihood estimator is the average: ^ = m1 Pi xi. Let ? be the optimal parameter. Then,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x NE( ?;1) |  |  | x NE( ?;1) | | | | | | |  |  | | | P^[x] |  |  |  |  |  |  |  |
| [`(^ ; x) |  | `( ?; x)] = |  |  |  |  |  |  |  | log | |  |  | P ? [x] |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | | 1 |  |  |  |  |  | 1 | | |
|  |  | = |  |  |  |  | NE( ?;1) | | |  | |  | (x ?)2 | | + | | (x ^)2 |
|  |  | x | |  |  |  |  |  |  |  |
|  |  |  | | 2 |  | 2 | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | ^2 | | | | | ( ?)2 | | |  |  |  |  |  |  |  |  |  |  |
|  |  | = |  |  | |  | |  |  |  | + ( ? ^) x | | | | |  | NE( ?;1)[x] | | | | |
|  |  |  | 2 | | | | 2 |  |  |
|  |  |  |  | ^2 | | | | | ( ?)2 | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | = |  |  | |  | |  |  |  | + ( ? ^) ? | | | | | |  |  |  |  |  |
|  |  |  | 2 | | | | 2 |  |  |  |  |  |  |
|  |  | = | 1 | | | (^ ?)2: | | | | | |  |  |  |  |  |  |  |  |  | (24.6) |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |
|  |  |  | 2 | |  |  |  |  |  |  |  |  |  |

Next, we note that ^ is the average of m Gaussian variables and therefore it is also distributed normally with mean ? and variance ?=m. From this fact we can derive bounds of the form: with probability of at least 1 we have that j^ ?j where depends on ?=m and on .

In some situations, the maximum likelihood estimator clearly over ts. For example, consider a Bernoulli random variable X and let P[X = 1] = ?. As we saw previously, using Hoe ding's inequality we can easily derive a guarantee

j ? ^j

on that holds with high probability (see Equation ([24.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page343))). However, if

our goal is to obtain a small value of the expected log-loss function as de ned in Equation ([24.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page345)) we might fail. For example, assume that ? is nonzero but very

small. Then, the probability that no element of a sample of size m will be 1 is (1 ?)m, which is greater than e 2 ? m. It follows that whenever m log(2)2? ,

the probability that the sample is all zeros is at least 50%, and in that case, the

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | ^ |  |  |  |  |  |  | ^ |
| maximum likelihood rule will set = 0. But the true risk of the estimate = 0 | | | | | | | | | | |
| is |  |  |  |  |  |  |  |  |  |  |
| x | ^ | ? | ^ |  |  | ? |  |  | ^ |  |
| E ?[`( ; x)] = |  | `( ; 1) + (1 |  | )`( ; 0) | | |  |
|  |  | ? | ^ |  |  |  |  | ? |  | ^ |
|  | = |  |  | |  | ) log(1=(1 |
|  |  | log(1= ) + (1 | |  | )) |

= ? log(1=0) = 1:

This simple example shows that we should be careful in applying the maximum likelihood principle.

To overcome over tting, we can use the variety of tools we encountered pre-

|  |  |
| --- | --- |
| 24.2 Naive Bayes | 347 |
|  |  |

viously in the book. A simple regularization technique is outlined in Exercise [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page356).

24.2 Naive Bayes

The Naive Bayes classi er is a classical demonstration of how generative as-sumptions and parameter estimations simplify the learning process. Consider the problem of predicting a label y 2 f0; 1g on the basis of a vector of features

1. = (x1; : : : ; xd), where we assume that each xi is in f0; 1g. Recall that the Bayes optimal classi er is

hBayes(x) = argmax P[Y = yjX = x]:

y2f0;1g

To describe the probability function P[Y = y jX = x] we need 2d parameters, each of which corresponds to P[Y = 1jX = x] for a certain value of x 2 f0; 1gd. This implies that the number of examples we need grows exponentially with the number of features.

In the Naive Bayes approach we make the (rather naive) generative assumption that given the label, the features are independent of each other. That is,

d

Y

P[X = xjY = y] = P[Xi = xijY = y]:

i=1

With this assumption and using Bayes' rule, the Bayes optimal classi er can be further simpli ed:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| hBayes(x) = | argmax P[Y = yjX = x] | | |  |
|  | y2f0;1g | |  |  |
| = | argmax P[Y = y]P[X = xjY = y] | | |  |
|  | y2f0;1g | |  |  |
|  | 2f | g | d |  |
|  | iY |  |
| = | argmax P[Y = y] | | P[Xi = xijY = y]: | (24.7) |
|  | y | 0;1 | =1 |  |
|  |  |  |  |

That is, now the number of parameters we need to estimate is only 2d + 1. Here, the generative assumption we made reduced signi cantly the number of parameters we need to learn.

When we also estimate the parameters using the maximum likelihood princi-ple, the resulting classi er is called the Naive Bayes classi er.

24.3 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is another demonstration of how generative assumptions simplify the learning process. As in the Naive Bayes classi er we consider again the problem of predicting a label y 2 f0; 1g on the basis of a

1. Generative Models

vector of features x = (x1; : : : ; xd). But now the generative assumption is as

follows. First, we assume that P[Y = 1] = P[Y = 0] = 1=2. Second, we assume that the conditional probability of X given Y is a Gaussian distribution. Finally,

the covariance matrix of the Gaussian distribution is the same for both values of the label. Formally, let 0; 1 2 Rd and let be a covariance matrix. Then, the density distribution is given by

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P[X = xjY = y] = | 1 | exp | 1 | | (x y)T 1(x y) : |
|  |  |  |
| (2 )d=2j j1=2 | 2 | |

As we have shown in the previous section, using Bayes' rule we can write

hBayes(x) = argmax P[Y = y]P[X = xjY = y]:

|  |  |  |
| --- | --- | --- |
|  | y2f0;1g |  |
| This means that we will predict hBayes(x) = 1 i | |  |
|  | P[Y = 0]P[X = xjY = 0] |
| log | P[Y = 1]P[X = xjY = 1] | > 0: |
|  |

This ratio is often called the log-likelihood ratio.

In our case, the log-likelihood ratio becomes

12 (x 0)T 1(x 0) 12 (x 1)T 1(x 1)

We can rewrite this as hw; xi + b where

w = ( 1 0)T 1 and b = 12 T0 1 0 T1 1 1 : (24.8)

As a result of the preceding derivation we obtain that under the aforemen-tioned generative assumptions, the Bayes optimal classi er is a linear classi er. Additionally, one may train the classi er by estimating the parameter 0; 1 and from the data, using, for example, the maximum likelihood estimator. With those estimators at hand, the values of w and b can be calculated as in Equation ([24.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page348)).

24.4 Latent Variables and the EM Algorithm

In generative models we assume that the data is generated by sampling from a speci c parametric distribution over our instance space X . Sometimes, it is convenient to express this distribution using latent random variables. A natural example is a mixture of k Gaussian distributions. That is, X = Rd and we assume that each x is generated as follows. First, we choose a random number in f1; : : : ; kg. Let Y be a random variable corresponding to this choice, and denote P[Y = y] = cy. Second, we choose x on the basis of the value of Y according to a Gaussian distribution

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P[X = xjY = y] = (2 )d=2j yj1=2 | | exp | 2 | | (x y)T y 1(x y) | : (24.9) |
| 1 | |  | 1 | |  |  |
|  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| 24.4 Latent Variables and the EM Algorithm | 349 |
|  |  |

Therefore, the density of X can be written as:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | k |  |  |  |  |  |  |  |  |  |  |
| P[X = x] = | X | P[Y = y]P[X = xjY = y] | | | | | | | |  |  |
| y=1 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| = | k | cy (2 )d=21 | | y |  | 1=2 | exp | 2 | | (x y)T y 1(x y) | : |
| y=1 |  |
|  | X |  |  |  |  |  |  | 1 | |  |  |
|  |  | j |  | j |  |  |  |  |  |  |

Note that Y is a hidden variable that we do not observe in our data. Neverthe-less, we introduce Y since it helps us describe a simple parametric form of the probability of X.

More generally, let be the parameters of the joint distribution of X and Y (e.g., in the preceding example, consists of cy, y, and y, for all y = 1; : : : ; k). Then, the log-likelihood of an observation x can be written as

k

!

X

log (P [X = x]) = log

P [X = x; Y = y]

:

y=1

Given an i.i.d. sample, S = (x1; : : : ; xm), we would like to nd that maxi-mizes the log-likelihood of S,

m

Y

L( ) = log P [X = xi]

i=1

m

X

= log P [X = xi]

i=1

m

k

!

X X

= log

P [X = xi; Y = y]

:

i=1

y=1

The maximum-likelihood estimator is therefore the solution of the maximization problem

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | m |  | P [X = xi; Y = y]! | : |
|  |  |  | k |  |
| argmax L( ) = | argmax | Xi | log | X |  |
|  |  |  |
|  |  | =1 |  | y=1 |  |

In many situations, the summation inside the log makes the preceding opti-mization problem computationally hard. The Expectation-Maximization (EM) algorithm, due to Dempster, Laird, and Rubin, is an iterative procedure for searching a (local) maximum of L( ). While EM is not guaranteed to nd the global maximum, it often works reasonably well in practice.

EM is designed for those cases in which, had we known the values of the latent variables Y , then the maximum likelihood optimization problem would have been tractable. More precisely, de ne the following function over m k matrices and the set of parameters :

m k

X X

F (Q; ) =

Qi;y

log (P [X = xi; Y = y]) :

=1 y=1

1. Generative Models

If each row of Q de nes a probability over the ith latent variable given X = xi,

then we can interpret F (Q; ) as the expected log-likelihood of a training set (x1; y1); : : : ; (xm; ym), where the expectation is with respect to the choice of each yi on the basis of the ith row of Q. In the de nition of F , the summation is outside the log, and we assume that this makes the optimization problem with respect to tractable:

assumption 24.1 For any matrix Q 2 [0; 1]m;k, such that each row of Q sums to 1, the optimization problem

argmax F (Q; )

is tractable.

The intuitive idea of EM is that we have a \chicken and egg" problem. On one hand, had we known Q, then by our assumption, the optimization problem of nding the best is tractable. On the other hand, had we known the parameters

we could have set Qi;y to be the probability of Y = y given that X = xi.

The EM algorithm therefore alternates between nding given Q and nding Q given . Formally, EM nds a sequence of solutions (Q(1); (1)); (Q(2); (2)); : : :

where at iteration t, we construct (Q(t+1); (t+1)) by performing two steps. Expectation Step: Set

|  |  |
| --- | --- |
| Qi;y(t+1) = P (t) [Y = yjX = xi]: | (24.10) |

This step is called the Expectation step, because it yields a new probabil-ity over the latent variables, which de nes a new expected log-likelihood function over .

Maximization Step: Set (t+1) to be the maximizer of the expected log-likelihood, where the expectation is according to Q(t+1):

|  |  |
| --- | --- |
| (t+1) = argmax F (Q(t+1); ): | (24.11) |
|  |  |

By our assumption, it is possible to solve this optimization problem e - ciently.

The initial values of (1) and Q(1) are usually chosen at random and the procedure terminates after the improvement in the likelihood value stops being signi cant.

24.4.1 EM as an Alternate Maximization Algorithm

To analyze the EM algorithm, we rst view it as an alternate maximization algorithm. De ne the following objective function

m k

X X

G(Q; ) = F (Q;

)

Qi;y log(Qi;y):

=1 y=1

|  |  |
| --- | --- |
| 24.4 Latent Variables and the EM Algorithm | 351 |
|  |  |

The second term is the sum of the entropies of the rows of Q. Let

|  |  |  |
| --- | --- | --- |
| Q = (Q 2 [0; 1]m;k : 8i; | y=1 Qi;y = 1 | ) |
|  | k |  |
|  | X |  |

be the set of matrices whose rows de ne probabilities over [k]. The following lemma shows that EM performs alternate maximization iterations for maximiz-ing G.

lemma 24.2 The EM procedure can be rewritten as:

Q(t+1) = argmax G(Q; (t))

Q2Q

(t+1) = argmax G(Q(t+1); ) :

Furthermore, G(Q(t+1); (t)) = L( (t)).

Proof Given Q(t+1) we clearly have that

argmax G(Q(t+1); ) = argmax F (Q(t+1); ):

Therefore, we only need to show that for any , the solution of argmaxQ2Q G(Q; ) is to set Qi;y = P [Y = yjX = xi]. Indeed, by Jensen's inequality, for any Q 2 Q we have that

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G(Q; ) = | k | Qi;y log P | | | | | Qi;y | | ! |
| m |  |  | | |  | [X = xi; Y = y] | |  |
| i=1 | y=1 |  |  |
| X X | | k |  |  |  |  |  |  | !! |
|  | log | Qi;y P | | |  | Qi;y | |
| m |  |  |  |  | [X = xi; Y = y] | | |  |
| Xi |  | X |  |  |  |  |
|  |  |  |  |  |  |  |  |
| =1 |  | y=1 |  |  |  |  | ! |  |  |
| m |  | k |  |  |  |  |  |  |
| X |  | X |  |  |  |  |  |  |  |

1. logP [X = xi; Y = y]

i=1 y=1

m

X

1. log (P [X = xi]) = L( );

i=1

y P (t) [Y = yjX =

1. Generative Models

while for Qi;y = P [Y = yjX = xi] we have

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| G(Q; ) = | k | P [Y = yjX = xi] log | P [Y = y X = xi] | | | ! |
| m |  | [X = xi; Y = y] | | |  |
| i=1 | y=1 |
| X X | |  |  |
|  | P | j |  |  |

1. k

X X

1. P [Y = yjX = xi] log (P [X = xi])

i=1 y=1

1. k

X X

= log (P [X = xi]) P [Y = yjX = xi]

i=1 y=1

m

X

1. log (P [X = xi]) = L( ):

i=1

This shows that setting Qi;y = P [Y = yjX = xi] maximizes G(Q; ) over Q 2 Q and shows that G(Q(t+1); (t)) = L( (t)). 

The preceding lemma immediately implies:

theorem 24.3 The EM procedure never decreases the log-likelihood; namely, for all t,

L( (t+1)) L( (t)):

Proof By the lemma we have

L( (t+1)) = G(Q(t+2); (t+1)) G(Q(t+1); (t)) = L( (t)):

24.4.2 EM for Mixture of Gaussians (Soft k-Means)

Consider the case of a mixture of k Gaussians in which is a triplet (c; f 1; : : : ; kg; f 1; : : : ; kg) where P [Y = y] = cy and P [X = xjY = y] is as given in Equation ([24.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page348)). For

simplicity, we assume that 1 = 2 = = k = I, where I is the identity matrix. Specifying the EM algorithm for this case we obtain the following:

Expectation step: For each i 2 [m] and y 2 [k] we have that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P (t) [Y = yjX = xi] = | 1 | P (t) [Y = y] P (t) [X = xijY = y] | | | | | |  |
|  |  |
| Zi | (24.12) |
| = | Zi | cy(t) exp | 2 kxi y(t)k2 | | |  | ; |
|  | 1 |  | 1 | | |  |  |  |
|  |  |  |  |  |  |  |  |  |

P

where Zi is a normalization factor which ensures that xi] sums to 1.

Maximization step: We need to set t+1 to be a maximizer of Equation ([24.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page350)),

|  |  |
| --- | --- |
| 24.5 Bayesian Reasoning | 353 |
|  |  |

which in our case amounts to maximizing the following expression w.r.t. c and :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| m k | P (t) [Y = yjX = xi] | log(cy) 2 kxi yk2 | | |  | : | (24.13) |
| i=1 y=1 |
| X X |  | 1 | | |  |  |  |
|  |  |  |  |  |  |  |

Comparing the derivative of Equation ([24.13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page353)) w.r.t. y to zero and rear-ranging terms we obtain:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| y | m | |  |  |  | (t) [Y = y X = xi] | |  |
| P | i=1 | |  |  |  |
|  | i | =1 |  |  | (t) [Y = y | | X = xi] xi |  |
| = | m | P | |  | j |  | : |
|  |  | j |
|  | P | |  | P | |  |  |

That is, y is a weighted average of the xi where the weights are according to the probabilities calculated in the E step. To nd the optimal c we need to be more careful since we must ensure that c is a probability vector. In Exercise [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page356) we show that the solution is:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cy = |  | im=1 P (t) [Y = yjX = xi] | | | | | : | (24.14) |
|  |
|  | y0P | |  |  |  |  |  |  |
|  | k | m |  | (t) [Y = y0 |  | X = xi] | |  |
|  | P | =1 i=1 | P | j |  |
|  | P |  |  |  |  |

It is interesting to compare the preceding algorithm to the k-means algorithm described in Chapter [22](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page307). In the k-means algorithm, we rst assign each example to a cluster according to the distance kxi yk. Then, we update each center y according to the average of the examples assigned to this cluster. In the EM approach, however, we determine the probability that each example belongs to each cluster. Then, we update the centers on the basis of a weighted sum over the entire sample. For this reason, the EM approach for k-means is sometimes called \soft k-means."

24.5 Bayesian Reasoning

The maximum likelihood estimator follows a frequentist approach. This means that we refer to the parameter as a xed parameter and the only problem is that we do not know its value. A di erent approach to parameter estimation is called Bayesian reasoning. In the Bayesian approach, our uncertainty about

is also modeled using probability theory. That is, we think of as a random

variable as well and refer to the distribution P[ ] as a prior distribution. As its name indicates, the prior distribution should be de ned by the learner prior to observing the data.

As an example, let us consider again the drug company which developed a new drug. On the basis of past experience, the statisticians at the drug company believe that whenever a drug has reached the level of clinic experiments on people, it is likely to be e ective. They model this prior belief by de ning a density distribution on such that

|  |  |  |  |
| --- | --- | --- | --- |
| P[ ] = | (0:2 | if0:5 | (24.15) |
|  | 0:8 | if > 0:5 |  |

1. Generative Models

As before, given a speci c value of , it is assumed that the conditional proba-

bility, P[X = xj ], is known. In the drug company example, X takes values in f0; 1g and P[X = xj ] = x(1 )1 x.

Once the prior distribution over and the conditional distribution over X given are de ned, we again have complete knowledge of the distribution over

X. This is because we can write the probability over X as a marginal probability

X X

P[X = x] = P[X = x; ] = P[ ]P[X = xj ];

where the last equality follows from the de nition of conditional probability. If

is continuous we replace P[ ] with the density function and the sum becomes an integral:

Z

P[X = x] = P[ ]P[X = xj ] d :

Seemingly, once we know P[X = x], a training set S = (x1; : : : ; xm) tells us nothing as we are already experts who know the distribution over a new point X. However, the Bayesian view introduces dependency between S and X. This is because we now refer to as a random variable. A new point X and the previous points in S are independent only conditioned on . This is di erent from the frequentist philosophy in which is a parameter that we might not know, but since it is just a parameter of the distribution, a new point X and previous points S are always independent.

In the Bayesian framework, since X and S are not independent anymore, what we would like to calculate is the probability of X given S, which by the chain rule can be written as follows:

X

X

P[X = xjS] =

P[X = xj ; S] P[ jS] =

P[X = xj ] P[ jS]:

The second inequality follows from the assumption that X and S are independent when we condition on . Using Bayes' rule we have

P[ jS] = P[Sj ] P[ ] ;

P[S]

and together with the assumption that points are independent conditioned on , we can write

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | m |  |
| [ S] = | P[Sj ] P[ ] | | = |  | 1 |  | iY | [X = x ] [ ]: |
|  | | P |  |  |
| P j | P | [S] |  | [S] | | ij P |
|  |  |  | P |
|  |  |  |  |  |  |  | =1 |  |

We therefore obtain the following expression for Bayesian prediction:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | m |  |
| P[X = xjS] = |  | 1 | X P[X = xj ] Y P[X = xij ] P[ ]: | | (24.16) |
| P | [S] |
|  |  |  |  | i=1 |  |

Getting back to our drug company example, we can rewrite P[X = xjS] as

P[X = xjS] = 1 Z x+Pi xi (1 )1 x+Pi(1 xi) P[ ] d :

P [S]

|  |  |
| --- | --- |
| 24.6 Summary | 355 |
|  |  |

It is interesting to note that when P[ ] is uniform we obtain that

Z

P[X = xjS] / x+Pi xi (1 )1 x+Pi(1 xi) d :

Solving the preceding integral (using integration by parts) we obtain

P

P[X = 1jS] = ( i xi) + 1 :

m + 2

Recall that the prediction according to the maximum likelihood principle in this

P

P j^ xi

case is [X = 1 ] = mi . The Bayesian prediction with uniform prior is rather similar to the maximum likelihood prediction, except it adds \pseudoexamples" to the training set, thus biasing the prediction toward the uniform prior.

Maximum A Posteriori

In many situations, it is di cult to nd a closed form solution to the integral given in Equation ([24.16](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page354)). Several numerical methods can be used to approxi-mate this integral. Another popular solution is to nd a single which maximizes P[ jS]. The value of which maximizes P[ jS] is called the Maximum A Poste-riori estimator. Once this value is found, we can calculate the probability that X = x given the maximum a posteriori estimator and independently on S.

24.6 Summary

In the generative approach to machine learning we aim at modeling the distri-bution over the data. In particular, in parametric density estimation we further assume that the underlying distribution over the data has a speci c paramet-ric form and our goal is to estimate the parameters of the model. We have described several principles for parameter estimation, including maximum like-lihood, Bayesian estimation, and maximum a posteriori. We have also described several speci c algorithms for implementing the maximum likelihood under dif-ferent assumptions on the underlying data distribution, in particular, Naive Bayes, LDA, and EM.

24.7 Bibliographic Remarks

The maximum likelihood principle was studied by Ronald Fisher in the beginning of the 20th century. Bayesian statistics follow Bayes' rule, which is named after the 18th century English mathematician Thomas Bayes.

There are many excellent books on the generative and Bayesian approaches to machine learning. See, for example, (Bishop 2006, Koller & Friedman 2009, MacKay 2003, Murphy 2012, Barber 2012).

1. Generative Models

24.8 Exercises

1. Prove that the maximum likelihood estimator of the variance of a Gaussian variable is biased.
2. Regularization for Maximum Likelihood: Consider the following regularized

loss minimization:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 |  | m | 1 | |  |
|  |  | Xi | log(1=P [xi]) + |  | (log(1= ) + log(1=(1 ))) : |
| m | | =1 | m |
|  |  |  |  |  |

Show that the preceding objective is equivalent to the usual empirical error had we added two pseudoexamples to the training set. Conclude that the regularized maximum likelihood estimator would be

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | ^ = m + 2 1 + xi! | | | | | | : |  |  |  |  |
|  |  |  |  |  |  | 1 | |  |  |  | m |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | Xi |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | =1 |  |  |  |  |  |
|  |  |  |  |  |  | ^ | | |  | ? |  |  | ^ | ^ | | |
| Derive a high probability bound on j | | | | | | | | |  | j. Hint: Rewrite this as j | | | E[ ]+ | | |
| ^ |  | ? | j and then use the triangle inequality and Hoe ding inequality. | | | | | | | | | | | | | |
| E[ ] | |  |
|  |  |  |  |  |  |  |  |  |  |  |  | ^ |  |  | 1 |  |
| Use this to bound the true risk. Hint: Use the fact that now | | | | | | | | | | | | |  | | m+2 | to |
|  | ^ | |  | ? | j to the relative entropy. | | | | |  |  |  |  |  |  |  |
| relate j | | |  |  |  |  |  |  |  |  |
| 3. Consider a general optimization problem of the form: | | | | | | | | | | | | |  |  |  |  |
|  |  |  |  |  |  | k | | |  |  | X | |  |  |  |  |
|  |  |  |  |  |  | X | | |  |  |  |  |  |  |
|  |  |  |  |  | max | y log(cy) s.t. | | | |  | cy > 0; | cy = 1 ; |  |  |  |  |
|  |  |  |  |  | c | y=1 | | |  |  |  | y |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

where 2 Rk+ is a vector of nonnegative weights. Verify that the M step of soft k-means involves solving such an optimization problem.

Let c? = P1 . Show that c? is a probability vector.

y y

Show that the optimization problem is equivalent to the problem:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | (c? | jj |  |  |  |  | X |  |  |
| min D | RE | c) s.t. | c | y | > 0; | c | y | = 1 : |
| c |  |  |  |  | y |  |
|  |  |  |  |  |  |  |  |  |  |

Using properties of the relative entropy, conclude that c? is the solution to the optimization problem.

1. Feature Selection and Generation

In the beginning of the book, we discussed the abstract model of learning, in which the prior knowledge utilized by the learner is fully encoded by the choice of the hypothesis class. However, there is another modeling choice, which we have so far ignored: How do we represent the instance space X ? For example, in the papayas learning problem, we proposed the hypothesis class of rectangles in the softness-color two dimensional plane. That is, our rst modeling choice was to represent a papaya as a two dimensional point corresponding to its softness and color. Only after that did we choose the hypothesis class of rectangles as a class of mappings from the plane into the label set. The transformation from the real world object \papaya" into the scalar representing its softness or its color is called a feature function or a feature for short; namely, any measurement of the real world object can be regarded as a feature. If X is a subset of a vector space, each x 2 X is sometimes referred to as a feature vector. It is important to understand that the way we encode real world objects as an instance space X is by itself prior knowledge about the problem.

Furthermore, even when we already have an instance space X which is rep-resented as a subset of a vector space, we might still want to change it into a di erent representation and apply a hypothesis class on top of it. That is, we may de ne a hypothesis class on X by composing some class H on top of a feature function which maps X into some other vector space X 0. We have al-ready encountered examples of such compositions { in Chapter [15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page202) we saw that kernel-based SVM learns a composition of the class of halfspaces over a feature mapping that maps each original instance in X into some Hilbert space. And, indeed, the choice of is another form of prior knowledge we impose on the problem.

In this chapter we study several methods for constructing a good feature set. We start with the problem of feature selection, in which we have a large pool of features and our goal is to select a small number of features that will be used by our predictor. Next, we discuss feature manipulations and normalization. These include simple transformations that we apply on our original features. Such transformations may decrease the sample complexity of our learning algorithm, its bias, or its computational complexity. Last, we discuss several approaches for feature learning. In these methods, we try to automate the process of feature construction.

1. Feature Selection and Generation

We emphasize that while there are some common techniques for feature learn-ing one may want to try, the No-Free-Lunch theorem implies that there is no ulti-mate feature learner. Any feature learning algorithm might fail on some problem. In other words, the success of each feature learner relies (sometimes implicitly) on some form of prior assumption on the data distribution. Furthermore, the relative quality of features highly depends on the learning algorithm we are later going to apply using these features. This is illustrated in the following example.

Example 25.1 Consider a regression problem in which X = R2, Y = R, and the loss function is the squared loss. Suppose that the underlying distribution is such that an example (x; y) is generated as follows: First, we sample x1 from the uniform distribution over [ 1; 1]. Then, we deterministically set y = x12. Finally, the second feature is set to be x2 = y + z, where z is sampled from the uniform distribution over [ 0:01; 0:01]. Suppose we would like to choose a single feature. Intuitively, the rst feature should be preferred over the second feature as the target can be perfectly predicted based on the rst feature alone, while it cannot be perfectly predicted based on the second feature. Indeed, choosing the rst feature would be the right choice if we are later going to apply polynomial regression of degree at least 2. However, if the learner is going to be a linear regressor, then we should prefer the second feature over the rst one, since the optimal linear predictor based on the rst feature will have a larger risk than the optimal linear predictor based on the second feature.

25.1 Feature Selection

Throughout this section we assume that X = Rd. That is, each instance is repre-sented as a vector of d features. Our goal is to learn a predictor that only relies on k d features. Predictors that use only a small subset of features require a smaller memory footprint and can be applied faster. Furthermore, in applications such as medical diagnostics, obtaining each possible \feature" (e.g., test result) can be costly; therefore, a predictor that uses only a small number of features is desirable even at the cost of a small degradation in performance, relative to a predictor that uses more features. Finally, constraining the hypothesis class to use a small subset of features can reduce its estimation error and thus prevent over tting.

Ideally, we could have tried all subsets of k out of d features and choose the subset which leads to the best performing predictor. However, such an exhaustive search is usually computationally intractable. In the following we describe three computationally feasible approaches for feature selection. While these methods cannot guarantee nding the optimal subset, they often work reasonably well in practice. Some of the methods come with formal guarantees on the quality of the selected subsets under certain assumptions. We do not discuss these guarantees here.

|  |  |
| --- | --- |
| 25.1 Feature Selection | 359 |
|  |  |

25.1.1 Filters

Maybe the simplest approach for feature selection is the lter method, in which we assess individual features, independently of other features, according to some quality measure. We can then select the k features that achieve the highest score (alternatively, decide also on the number of features to select according to the value of their scores).

Many quality measures for features have been proposed in the literature. Maybe the most straightforward approach is to set the score of a feature ac-cording to the error rate of a predictor that is trained solely by that feature.

To illustrate this, consider a linear regression problem with the squared loss. Let v = (x1;j; : : : ; xm;j) 2 Rm be a vector designating the values of the jth feature on a training set of m examples and let y = (y1; : : : ; ym) 2 Rm be the values of the target on the same m examples. The empirical squared loss of an ERM linear predictor that uses only the jth feature would be

min 1 kav + b yk2;

a;b2R m

where the meaning of adding a scalar b to a vector v is adding b to all coordinates

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | |  |  | m |  |  |  |  |  |  |  |  |  |
| of v. To solve this problem, let v = | | | | | | | | | | | | | | |  |  |  |  | i=1 vi be the averaged value of the | | | | | | | | | |
|  |  | m | |  |  |
|  |  |  |  | 1 | |  |  |  | m | |  |  |  |  |  | averaged value of the target. Clearly (see | | | | | | | | | | | | | | |
| feature and let y = | | | | | m |  | P | | i=1 y | | | | i be the | |  |  |  |  | P | |  |  |  |  |  |  |  |  |  |  |
| Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page371)), | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| min | | 1 |  | av + b | | | |  |  | y |  | 2 | = min | |  | 1 | |  |  | a(v | |  | v) + b |  | (y |  | y) |  | 2: | (25.1) |
| m k | |  | | k |  | m k | | | | |  |  |  | k |
| a;b | 2R |  |  |  |  |  |  | a;b | 2R |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Taking the derivative of the right-hand side objective with respect to b and comparing it to zero we obtain that b = 0. Similarly, solving for a (once we know that b = 0) yields a = hv v; y yi=kv vk2. Plugging this value back into the objective we obtain the value

ky yk2 (hv v; y yi)2 :

kv vk2

Ranking the features according to the minimal loss they achieve is equivalent to ranking them according to the absolute value of the following score (where now a higher score yields a better feature):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| hv v; y yi |  |  |  |  |  |  |  | 1 | hv v; y yi | | | | |  |  |  |
|  | = |  |  |  |  |  | m |  | : | (25.2) |
|  |  |  |  |  |  |  |  |  |  |  |
| kv vk ky yk | | q | | | 1 | kv vk2 q | | | | | 1 | ky yk2 |  |
|  |  |  |  |
|  | m | m |  |  |  |

The preceding expression is known as Pearson's correlation coe cient. The nu-merator is the empirical estimate of the covariance of the jth feature and the target value, E[(v E v)(y E y)], while the denominator is the squared root of the empirical estimate for the variance of the jth feature, E[(v E v)2], times the variance of the target. Pearson's coe cient ranges from 1 to 1, where if the Pearson's coe cient is either 1 or 1, there is a linear mapping from v to y with zero empirical risk.

1. Feature Selection and Generation

If Pearson's coe cient equals zero it means that the optimal linear function from v to y is the all-zeros function, which means that v alone is useless for predicting y. However, this does not mean that v is a bad feature, as it might be the case that together with other features v can perfectly predict y. Indeed, consider a simple example in which the target is generated by the function y = x1 + 2x2. Assume also that x1 is generated from the uniform distribution over f 1g, and x2 = 12 x1 + 12 z, where z is also generated i.i.d. from the uniform distribution over f 1g. Then, E[x1] = E[x2] = E[y] = 0, and we also have

E[yx1] = E[x21] + 2 E[x2x1] = E[x21] E[x21] + E[zx1] = 0:

Therefore, for a large enough training set, the rst feature is likely to have a Pearson's correlation coe cient that is close to zero, and hence it will most probably not be selected. However, no function can predict the target value well without knowing the rst feature.

There are many other score functions that can be used by a lter method. Notable examples are estimators of the mutual information or the area under the receiver operating characteristic (ROC) curve. All of these score functions su er from similar problems to the one illustrated previously. We refer the reader to Guyon & Elissee (2003).

25.1.2 Greedy Selection Approaches

Greedy selection is another popular approach for feature selection. Unlike lter methods, greedy selection approaches are coupled with the underlying learning algorithm. The simplest instance of greedy selection is forward greedy selection. We start with an empty set of features, and then we gradually add one feature at a time to the set of selected features. Given that our current set of selected features is I, we go over all i 2= I, and apply the learning algorithm on the set of features I [ fig. Each such application yields a di erent predictor, and we choose to add the feature that yields the predictor with the smallest risk (on the training set or on a validation set). This process continues until we either select k features, where k is a prede ned budget of allowed features, or achieve an accurate enough predictor.

Example 25.2 (Orthogonal Matching Pursuit) To illustrate the forward greedy selection approach, we specify it to the problem of linear regression with the squared loss. Let X 2 Rm;d be a matrix whose rows are the m training instances. Let y 2 Rm be the vector of the m labels. For every i 2 [d], let Xi be the ith column of X. Given a set I [d] we denote by XI the matrix whose columns are fXi : i 2 Ig.

The forward greedy selection method starts with I0 = ;. At iteration t, we look for the feature index jt, which is in

argmin min kXIt 1[fjgw yk2:

j w2Rt

kujk2

2Rt

|  |  |
| --- | --- |
| 25.1 Feature Selection | 361 |
|  |  |

Then, we update It = It 1 [ fjtg.

We now describe a more e cient implementation of the forward greedy selec-tion approach for linear regression which is called Orthogonal Matching Pursuit (OMP). The idea is to keep an orthogonal basis of the features aggregated so far. Let Vt be a matrix whose columns form an orthonormal basis of the columns of XIt .

Clearly,

min kXIt w yk2 = min kVt yk2:

w

We will maintain a vector t which minimizes the right-hand side of the equation. Initially, we set I0 = ;, V0 = ;, and 1 to be the empty vector. At round t, for every j, we decompose Xj = vj + uj where vj = Vt 1Vt>1Xj is the projection of Xj onto the subspace spanned by Vt 1 and uj is the part of Xj orthogonal to

Vt 1 (see Appendix [C](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page430)). Then,

min kVt 1 + uj yk2

;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| = ; | kVt 1 yk | | | | | | 2 |  | 2 |  |  |  | 2 | + 2 huj; Vt 1 yi | | | | | | |
| 2 | + 2 | | kujk2 | | | |
| min |  |  |  |  |  |  |  |  |  |  |  | jk 2 | |  |  | 2h |  |  |  | i |
| = ; | k | t 1 |  |  | y | k | 2 | + | | k | u |  |  | j |  | y |
| min | V |  |  |  |  |  |  |  | + 2 | | | u ; |  |  |
|  |  |  |  |  |  | 2 |  |  | min | | | 2 | k | 2 |  |  |  |  |
| = min | kVt 1 yk | | | | | |  |  | min | | |  | ujk 2 huj; yi | | | | | | | |
|  |  | + | | |  |  |
| = kVt 1 t 1 yk + | | | | | | | | |  |  | kujk 2 huj; yi | | | | | | | | | |

1. kVt 1 t 1 yk2 (huj; yi)2 : kujk2

It follows that we should select the feature

jt = argmax (huj; yi)2 :

j

The rest of the update is to set

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vt = | Vt 1 | ; kujt k2 | | | ; | t = t 1 ; | hkujt k2i | | : |
|  |  |  | ujt |  |  |  | ujt ; y |  |  |

The OMP procedure maintains an orthonormal basis of the selected features, where in the preceding description, the orthonormalization property is obtained by a procedure similar to Gram-Schmidt orthonormalization. In practice, the Gram-Schmidt procedure is often numerically unstable. In the pseudocode that follows we use SVD (see Section [C.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page431)) at the end of each round to obtain an orthonormal basis in a numerically stable manner.

1. Feature Selection and Generation

Orthogonal Matching Pursuit (OMP)

input:

data matrix X 2 Rm;d, labels vector y 2 Rm,

budget of features T

initialize: I1 = ;

for t = 1; : : : ; T

use SVD to nd an orthonormal basis V 2 Rm;t 1 of XIt (for t = 1 set V to be the all zeros matrix)

|  |  |  |  |
| --- | --- | --- | --- |
| foreach j 2 [d] n It let uj | = Xj 2 | | V V >Xj |
| let jt = argmaxj 2=It:kujk>0 |  | (huj;yi) |  |
|  | kujk2 |  |
| update It+1 = It [ fjtg |  |  |  |
| output IT +1 |  |  |  |

More E cient Greedy Selection Criteria

Let R(w) be the empirical risk of a vector w. At each round of the forward greedy selection method, and for every possible j, we should minimize R(w) over the vectors w whose support is It 1 [ fjg. This might be time consuming.

A simpler approach is to choose jt that minimizes

argmin min R(wt 1 + ej);

1. 2R

where ej is the all zeros vector except 1 in the jth element. That is, we keep the weights of the previously chosen coordinates intact and only optimize over the new variable. Therefore, for each j we need to solve an optimization problem over a single variable, which is a much easier task than optimizing over t.

An even simpler approach is to upper bound R(w) using a \simple" function and then choose the feature which leads to the largest decrease in this upper bound. For example, if R is a -smooth function (see Equation ([12.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page162)) in Chap-ter [12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page156)), then

R(w + ej) R(w) + @R(w) + 2=2:

@wj

Minimizing the right-hand side over yields = @R(w)

@wj

value into the above yields

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| R(w + ej) R(w) | 1 |  | @R(w) | |  | 2 |
|  |  |  |  |
| 2 | @wj | |  |

1 and plugging this

:

This value is minimized if the partial derivative of R(w) with respect to wj is maximal. We can therefore choose jt to be the index of the largest coordinate of the gradient of R(w) at w.

Remark 25.3 (AdaBoost as a Forward Greedy Selection Procedure) It is pos-sible to interpret the AdaBoost algorithm from Chapter [10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page130) as a forward greedy

|  |  |
| --- | --- |
| 25.1 Feature Selection | 363 |
|  |  |

selection procedure with respect to the function

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 |  | 0 |  |  | 11 |  |  |
|  | m |  | @ | d |  | AA |  |  |
| R(w) = log | @Xi | exp | X | wjhj(xi) | : | (25.3) |
|  |  | yi |  |
|  | =1 |  |  | j=1 |  |  |  |  |

See Exercise [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page371).

Backward Elimination

Another popular greedy selection approach is backward elimination. Here, we start with the full set of features, and then we gradually remove one feature at a time from the set of features. Given that our current set of selected features is I, we go over all i 2 I, and apply the learning algorithm on the set of features I nfig. Each such application yields a di erent predictor, and we choose to remove the feature i for which the predictor obtained from I n fig has the smallest risk (on the training set or on a validation set).

Naturally, there are many possible variants of the backward elimination idea.

It is also possible to combine forward and backward greedy steps.

25.1.3 Sparsity-Inducing Norms

The problem of minimizing the empirical risk subject to a budget of k features can be written as

min LS(w) s.t. kwk0 k;

w

where[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page363)

kwk0 = jfi : wi 6= 0gj:

In other words, we want w to be sparse, which implies that we only need to measure the features corresponding to nonzero elements of w.

Solving this optimization problem is computationally hard (Natarajan 1995, Davis, Mallat & Avellaneda 1997). A possible relaxation is to replace the non-

Pd

convex function kwk0 with the `1 norm, kwk1 = i=1 jwij, and to solve the problem

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| w | S | (w) s.t. | k | w | k1 | 1 | (25.4) |
| min L | |  |  | k ; |

where k1 is a parameter. Since the `1 norm is a convex function, this problem can be solved e ciently as long as the loss function is convex. A related problem is minimizing the sum of LS(w) plus an `1 norm regularization term,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| w | S | k | w | k1 | ) ; | (25.5) |
| min (L | | (w) + |  |

where is a regularization parameter. Since for any k1 there exists a such that

1. The function k k0 is often referred to as the `0 norm. Despite the use of the \norm" notation, k k0 is not really a norm; for example, it does not satisfy the positive homogeneity property of norms, kawk0 6= jaj kwk0.

1. Feature Selection and Generation

Equation ([25.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page363)) and Equation ([25.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page363)) lead to the same solution, the two problems

are in some sense equivalent.

The `1 regularization often induces sparse solutions. To illustrate this, let us

start with the simple optimization problem

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| w2R | 2 w | | |  | xw + jwj | : |  |
| min | 1 | | | 2 |  |  | (25.6) |
|  |  |  |  |  |  |

It is easy to verify (see Exercise [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page371)) that the solution to this problem is the \soft thresholding" operator

|  |  |
| --- | --- |
| w = sign(x) [jxj ]+ ; | (25.7) |

def

where [a]+ = maxfa; 0g. That is, as long as the absolute value of x is smaller than , the optimal solution will be zero.

Next, consider a one dimensional regression problem with respect to the squared loss:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | w Rm2m | | | | | | |  |  | (xiw yi) + jwj! : | | | | | | | |  |  |  |
|  |  |  | 2 |  |  | 1 | |  | m | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | Xi | | |  |  |  |  |  | 2 |  |  |  |  |
|  | argmin | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | =1 | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| We can rewrite the problem as | | | | | | | |  |  |  |  |  |  |  |  | xiyi! | w + jwj! |  |  |  |
| w Rm2 m | | | | | |  | xi ! wm | | | | | | | | | : |  |  |
|  | 1 | |  |  |  |  |  |  |  |  |  |  |  |  |  | m |  |  |  |  |
| 2 |  | 1 |  | Xi | |  |  |  | 2 |  | 1 | |  | Xi |  |  |  |  |
| argmin |  |  |  |  | 2 | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | =1 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| For simplicity let us assume that | | | | | | | |  | 1 |  |  | 2 |  |  |  |  |  | m | x y | ; |
|  |  |  | = 1, and denote hx; yi = Pi=1 | | | | | |
| then the optimal solution is | | | | | | | | m | | Pi xi | | | i i |  |

1. = sign(hx; yi) [jhx; yij=m ]+ :

That is, the solution will be zero unless the correlation between the feature x and the labels vector y is larger than .

Remark 25.4 Unlike the `1 norm, the `2 norm does not induce sparse solutions.

Indeed, consider the problem above with an `2 regularization, namely,

argmin

w2Rm

Then, the optimal solution is

|  |  |  |
| --- | --- | --- |
| 2m | (xiw yi)2 | + w2! : |
| 1 | m |  |
|  | Xi |  |
|  |  |
|  | =1 |  |

hx; yi=m

w = kxk2=m + 2 :

This solution will be nonzero even if the correlation between x and y is very small. In contrast, as we have shown before, when using `1 regularization, w will be nonzero only if the correlation between x and y is larger than the regularization parameter .

108 1

0:25

|  |  |
| --- | --- |
| 25.2 Feature Manipulation and Normalization | 365 |
|  |  |

Adding `1 regularization to a linear regression problem with the squared loss yields the LASSO algorithm, de ned as

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| argmin |  | 1 | 2 | : |  |
| 2m kXw yk + kwk1 | | (25.8) |
| w |

Under some assumptions on the distribution and the regularization parameter , the LASSO will nd sparse solutions (see, for example, (Zhao & Yu 2006) and the references therein). Another advantage of the `1 norm is that a vector with low `1 norm can be \sparsi ed" (see, for example, (Shalev-Shwartz, Zhang & Srebro 2010) and the references therein).

25.2 Feature Manipulation and Normalization

Feature manipulations or normalization include simple transformations that we apply on each of our original features. Such transformations may decrease the approximation or estimation errors of our hypothesis class or can yield a faster algorithm. Similarly to the problem of feature selection, here again there are no absolute \good" and \bad" transformations, but rather each transformation that we apply should be related to the learning algorithm we are going to apply on the resulting feature vector as well as to our prior assumptions on the problem.

To motivate normalization, consider a linear regression problem with the squared loss. Let X 2 Rm;d be a matrix whose rows are the instance vectors and let y 2 Rm be a vector of target values. Recall that ridge regression returns the vector

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| w | m kXw yk + kwk | | | | = (2 mI + X>X) X>y: |
| argmin | 1 | | 2 | 2 | 1 |
|  |  |
|  |  |  |  |  |

Suppose that d = 2 and the underlying data distribution is as follows. First we sample y uniformly at random from f 1g. Then, we set x1 to be y + 0:5 , where

is sampled uniformly at random from f 1g, and we set x2 to be 0:0001y. Note that the optimal weight vector is w? = [0; 10000], and LD(w?) = 0. However, the objective of ridge regression at w? is 108. In contrast, the objective of ridge regression at w = [1; 0] is likely to be close to 0:25 + . It follows that whenever

> 0:25 10 8, the objective of ridge regression is smaller at the

suboptimal solution w = [1; 0]. Since typically should be at least 1=m (see the analysis in Chapter [13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page171)), it follows that in the aforementioned example, if the number of examples is smaller than 108 then we are likely to output a suboptimal solution.

The crux of the preceding example is that the two features have completely di erent scales. Feature normalization can overcome this problem. There are many ways to perform feature normalization, and one of the simplest approaches is simply to make sure that each feature receives values between 1 and 1. In the preceding example, if we divide each feature by the maximal value it attains

366 Feature Selection and Generation

we will obtain that x1 = y+0:5 and x2 = y. Then, for 10 3 the solution of

1:5

ridge regression is quite close to w?.

Moreover, the generalization bounds we have derived in Chapter [13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page171) for reg-ularized loss minimization depend on the norm of the optimal vector w? and on the maximal norm of the instance vectors.[2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page366) Therefore, in the aforementioned example, before we normalize the features we have that kw?k2 = 108, while af-ter we normalize the features we have that kw?k2 = 1. The maximal norm of the instance vector remains roughly the same; hence the normalization greatly improves the estimation error.

Feature normalization can also improve the runtime of the learning algorithm. For example, in Section [14.5.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page199) we have shown how to use the Stochastic Gradient Descent (SGD) optimization algorithm for solving the regularized loss minimiza-tion problem. The number of iterations required by SGD to converge also depends on the norm of w? and on the maximal norm of kxk. Therefore, as before, using normalization can greatly decrease the runtime of SGD.

Next, we demonstrate in the following how a simple transformation on features, such as clipping, can sometime decrease the approximation error of our hypoth-esis class. Consider again linear regression with the squared loss. Let a > 1 be a large number, suppose that the target y is chosen uniformly at random from f 1g, and then the single feature x is set to be y with probability (1 1=a) and set to be ay with probability 1=a. That is, most of the time our feature is bounded but with a very small probability it gets a very high value. Then, for any w, the expected squared loss of w is

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LD(w) = | E | 1 | | (wx y)2 | | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | 2 |  |  |  |  | (awy y)2: |
| = |  | 1 a | | | | | 2 (wy y)2 + a 2 | | | | | | |
|  |  | 1 | | | |  | 1 | |  | 1 1 | | | |  |
|  | | | | |  |  |  |  | 22a 1 |  |  |  |  |  |
| Solving for w we obtain that w? = | | | | | | |  |  | , which goes to zero as a goes to in n- | | | | |
|  |  |  |  |  |  |  |  | a +a 1 | |  |  |  |  |  |

ity. Therefore, the objective at w? goes to 0:5 as a goes to in nity. For example,

for a = 100 we will obtain LD(w?) 0:48. Next, suppose we apply a \clipping" transformation; that is, we use the transformation x 7!sign(x) minf1; jxjg. Then, following this transformation, w? becomes 1 and LD(w?) = 0. This simple ex-ample shows that a simple transformation can have a signi cant in uence on the approximation error.

Of course, it is not hard to think of examples in which the same feature trans-formation actually hurts performance and increases the approximation error. This is not surprising, as we have already argued that feature transformations

1. More precisely, the bounds we derived in Chapter [13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page171) for regularized loss minimization depend on kw?k2 and on either the Lipschitzness or the smoothness of the loss function. For linear predictors and loss functions of the form `(w; (x; y)) = (hw; xi; y), where is convex and either 1-Lipschitz or 1-smooth with respect to its rst argument, we have that ` is either kxk-Lipschitz or kxk2-smooth. For example, for the squared loss,

(a; y) = 12 (a y)2, and `(w; (x; y)) = 12 (hw; xi y)2 is kxk2-smooth with respect to its rst argument.

fi fmin

Pm

i=1

|  |  |
| --- | --- |
| 25.2 Feature Manipulation and Normalization | 367 |
|  |  |

should rely on our prior assumptions on the problem. In the aforementioned ex-ample, a prior assumption that may lead us to use the \clipping" transformation is that features that get values larger than a prede ned threshold value give us no additional useful information, and therefore we can clip them to the prede ned threshold.

25.2.1 Examples of Feature Transformations

We now list several common techniques for feature transformations. Usually, it is helpful to combine some of these transformations (e.g., centering + scaling). In the following, we denote by f = (f1; : : : ; fm) 2 Rm the value of the feature f

over the m training examples. Also, we denote by f = m1 fi the empirical

mean of the feature over all examples.

|  |  |
| --- | --- |
| Centering: | fi f. |
| This transformation makes the feature have zero mean, by setting fi |
| Unit Range: |  |

|  |  |  |
| --- | --- | --- |
| This transformation makes the range of each feature | be [0; 1]. Formally, let | |
| fmax = maxi fi and fmin = mini fi. Then, we set fi | fi fmin | . Similarly, |
|  | fmax fmin | |

we can make the range of each feature be [ 1; 1] by the transformation fi

2 fmax fmin 1. Of course, it is easy to make the range [0; b] or [ b; b], where b is

a user-speci ed parameter.

Standardization:

This transformation makes all features have a zero mean and unit variance.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 |  | m |  | 2 |  |
| Formally, let = |  | m | | i=1(fi f) | |  | be the empirical variance of the feature. |
|  | fi f | | | |  |  |  |
| Then, we set fi |  | p | | P. |  |  |  |

Clipping:

This transformation clips high or low values of the feature. For example, fi sign(fi) maxfb; jfijg, where b is a user-speci ed parameter.

Sigmoidal Transformation:

As its name indicates, this transformation applies a sigmoid function on the

feature. For example, fi 1 , where b is a user-speci ed parameter.

1+exp(b fi)

This transformation can be thought of as a \soft" version of clipping: It has a small e ect on values close to zero and behaves similarly to clipping on values far away from zero.

1. Feature Selection and Generation

Logarithmic Transformation:

|  |  |
| --- | --- |
| The transformation is fi | log(b+fi), where b is a user-speci ed parameter. This |
| is widely used when the feature is a \counting" feature. For example, suppose | |
| that the feature represents the number of appearances of a certain word in a | |
| text document. Then, the di erence between zero occurrences of the word and | |
| a single occurrence is much more important than the di erence between 1000 | |
| occurrences and 1001 occurrences. | |
| Remark 25.5 In the aforementioned transformations, each feature is trans- | |
| formed on the basis of the values it obtains on the training set, independently | |
| of other features' values. In some situations we would like to set the parameter | |
| of the transformation on the basis of other features as well. A notable example | |
| is a transformation in which one applies a scaling to the features so that the | |
| empirical average of some norm of the instances becomes 1. | |

|  |  |  |
| --- | --- | --- |
| 25.3 | Feature Learning | |
|  | So far we have discussed feature selection and manipulations. In these cases, we | |
|  | start with a prede ned vector space Rd, representing our features. Then, we select | |
|  | a subset of features (feature selection) or transform individual features (feature | |
|  | transformation). In this section we describe feature learning, in which we start | |
|  | with some instance space, X , and would like to learn a function, : X ! Rd, | |
|  | which maps instances in X into a representation as d-dimensional feature vectors. | |
|  | The idea of feature learning is to automate the process of nding a good rep- | |
|  | resentation of the input space. As mentioned before, the No-Free-Lunch theorem | |
|  | tells us that we must incorporate some prior knowledge on the data distribution | |
|  | in order to build a good feature representation. In this section we present a few | |
|  | feature learning approaches and demonstrate conditions on the underlying data | |
|  | distribution in which these methods can be useful. | |
|  | Throughout the book we have already seen several useful feature construc- | |
|  | tions. For example, in the context of polynomial regression, we have mapped the | |
|  | original instances into the vector space of all their monomials (see Section [9.2.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page125) | |
|  | in Chapter [9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page117)). After performing this mapping, we trained a linear predictor on | |
|  | top of the constructed features. Automation of this process would be to learn | |
|  | a transformation | : X ! Rd, such that the composition of the class of linear |
|  | predictors on top of | yields a good hypothesis class for the task at hand. |
|  | In the following we describe a technique of feature construction called dictio- | |
|  | nary learning. |  |
| 25.3.1 | Dictionary Learning Using Auto-Encoders | |

The motivation of dictionary learning stems from a commonly used represen-tation of documents as a \bag-of-words": Given a dictionary of words D = fw1; : : : ; wkg, where each wi is a string representing a word in the dictionary,

|  |  |
| --- | --- |
| 25.3 Feature Learning | 369 |
|  |  |

and given a document, (p1; : : : ; pd), where each pi is a word in the document, we represent the document as a vector x 2 f0; 1gk, where xi is 1 if wi = pj for some j 2 [d], and xi = 0 otherwise. It was empirically observed in many text processing tasks that linear predictors are quite powerful when applied on this representation. Intuitively, we can think of each word as a feature that measures some aspect of the document. Given labeled examples (e.g., topics of the doc-uments), a learning algorithm searches for a linear predictor that weights these features so that a right combination of appearances of words is indicative of the label.

While in text processing there is a natural meaning to words and to the dic-tionary, in other applications we do not have such an intuitive representation of an instance. For example, consider the computer vision application of object recognition. Here, the instance is an image and the goal is to recognize which object appears in the image. Applying a linear predictor on the pixel-based rep-resentation of the image does not yield a good classi er. What we would like to have is a mapping that would take the pixel-based representation of the image and would output a bag of \visual words," representing the content of the image. For example, a \visual word" can be \there is an eye in the image." If we had such representation, we could have applied a linear predictor on top of this representation to train a classi er for, say, face recognition. Our question is, therefore, how can we learn a dictionary of \visual words" such that a bag-of-words representation of an image would be helpful for predicting which object appears in the image?

A rst naive approach for dictionary learning relies on a clustering algorithm (see Chapter [22](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page307)). Suppose that we learn a function c : X ! f1; : : : ; kg, where c(x) is the cluster to which x belongs. Then, we can think of the clusters as \words," and of instances as \documents," where a document x is mapped to the vector (x) 2 f0; 1gk, where (x)i is 1 if and only if x belongs to the ith cluster. Now, it is straightforward to see that applying a linear predictor on (x) is equivalent to assigning the same target value to all instances that belong to the same cluster. Furthermore, if the clustering is based on distances from a class center (e.g., k-means), then a linear predictor on (x) yields a piece-wise constant predictor on x.

Both the k-means and PCA approaches can be regarded as special cases of a more general approach for dictionary learning which is called auto-encoders. In an auto-encoder we learn a pair of functions: an \encoder" function, : Rd ! Rk, and a \decoder" function, : Rk ! Rd. The goal of the learning process is to

|  |  |
| --- | --- |
| nd a pair of functions such that the reconstruction error, | i kxi ( (xi))k2, |
| is small. Of course, we can trivially set k = d and both | to be the identity |
| ;P |

mapping, which yields a perfect reconstruction. We therefore must restrict and in some way. In PCA, we constrain k < d and further restrict and to be linear functions. In k-means, k is not restricted to be smaller than d, but now and rely on k centroids, 1; : : : ; k, and (x) returns an indicator vector

1. Feature Selection and Generation

in f0; 1gk that indicates the closest centroid to x, while takes as input an indicator vector and returns the centroid representing this vector.

An important property of the k-means construction, which is key in allowing k to be larger than d, is that maps instances into sparse vectors. In fact, in k-means only a single coordinate of (x) is nonzero. An immediate extension of the k-means construction is therefore to restrict the range of to be vectors with at most s nonzero elements, where s is a small integer. In particular, let and be functions that depend on 1; : : : ; k. The function maps an instance vector

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k |  |  |  |  |  |  | should have at most s nonzero elements. | | | | |
| x to a vector (x) 2 R , where | | | (x) | | |  |
| k | |  |  |  |  |
| The function (v) is de ned to be | | | | |  | i=1 vi i | | | | . As before, our goal is to have a | |
| small reconstruction error, and | | therefore we can de ne | | | | | | | | | |
|  |  | P | |  |  |  |  |  |  |
| (x) = | v |  | k | x |  | | (v) | k | 2 | s.t. | kvk0 s; |
|  | argmin | |  |  |  |  |  |

where kvk0 = jfj : vj 6= 0gj. Note that when s = 1 and we further restrict kvk1 = 1 then we obtain the k-means encoding function; that is, (x) is the indicator vector of the centroid closest to x. For larger values of s, the optimization problem in the preceding de nition of becomes computationally di cult. Therefore, in practice, we sometime use `1 regularization instead of the sparsity constraint and de ne to be

(x) = argmin kx (v)k2 + kvk1 ;

v

where > 0 is a regularization parameter. Anyway, the dictionary learning problem is now to nd the vectors 1; : : : ; k such that the reconstruction er-

ror, Pm kxi ( (x))k2, is as small as possible. Even if is de ned using

i=1

the `1 regularization, this is still a computationally hard problem (similar to the k-means problem). However, several heuristic search algorithms may give reasonably good solutions. These algorithms are beyond the scope of this book.

25.4 Summary

Many machine learning algorithms take the feature representation of instances for granted. Yet the choice of representation requires careful attention. We dis-cussed approaches for feature selection, introducing lters, greedy selection al-gorithms, and sparsity-inducing norms. Next we presented several examples for feature transformations and demonstrated their usefulness. Last, we discussed feature learning, and in particular dictionary learning. We have shown that fea-ture selection, manipulation, and learning all depend on some prior knowledge on the data.

|  |  |
| --- | --- |
| 25.5 Bibliographic Remarks | 371 |
|  |

25.5 Bibliographic Remarks

Guyon & Elissee (2003) surveyed several feature selection procedures, including many types of lters.

Forward greedy selection procedures for minimizing a convex objective sub-ject to a polyhedron constraint date back to the Frank-Wolfe algorithm (Frank

1. Wolfe 1956). The relation to boosting has been studied by several authors, including, (Warmuth, Liao & Ratsch 2006, Warmuth, Glocer & Vishwanathan 2008, Shalev-Shwartz & Singer 2008). Matching pursuit has been studied in the signal processing community (Mallat & Zhang 1993). Several papers analyzed greedy selection methods under various conditions. See, for example, Shalev-

Shwartz, Zhang & Srebro (2010) and the references therein.

The use of the `1-norm as a surrogate for sparsity has a long history (e.g. Tib-

shirani (1996) and the references therein), and much work has been done on un-derstanding the relationship between the `1-norm and sparsity. It is also closely related to compressed sensing (see Chapter [23](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page323)). The ability to sparsify low `1

norm predictors dates back to Maurey (Pisier 1980-1981). In Section [26.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page386) we also show that low `1 norm can be used to bound the estimation error of our predictor.

Feature learning and dictionary learning have been extensively studied recently in the context of deep neural networks. See, for example, (Lecun & Bengio 1995, Hinton et al. 2006, Ranzato et al. 2007, Collobert & Weston 2008, Lee et al. 2009, Le et al. 2012, Bengio 2009) and the references therein.

25.6 Exercises

1. Prove the equality given in Equation ([25.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page359)). Hint: Let a ; b be minimizers of the left-hand side. Find a; b such that the objective value of the right-hand side is smaller than that of the left-hand side. Do the same for the other direction.
2. Show that Equation ([25.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page364)) is the solution of Equation ([25.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page364)).
3. AdaBoost as a Forward Greedy Selection Algorithm: Recall the Ad-aBoost algorithm from Chapter [10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page130). In this section we give another interpre-

tation of AdaBoost as a forward greedy selection algorithm.

Given a set of m instances x1; : : : ; xm, and a hypothesis class H of nite VC dimension, show that there exist d and h1; : : : ; hd such that for every h 2 H there exists i 2 [d] with hi(xj) = h(xj) for every j 2 [m].

Let R(w) be as de ned in Equation ([25.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page363)). Given some w, de ne fw to be the function

d

X

fw( ) = wihi( ):

i=1

1. Feature Selection and Generation

Let D be the distribution over [m] de ned by

Di = exp( yifw(xi)) ;

Z

where Z is a normalization factor that ensures that D is a probability vector. Show that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | @R(w) | | | | |  |  | m | |  |  |  |  |  |  |  |  |
|  |  |  |  |  | Xi | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | wj | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | =Diyihj(xi): | | | | | | | |  |  |  |  |  |
|  |  |  |  |  |  | P | |  |  | =1 | |  |  |  |  |  |  |  |  |
|  | Furthermore, denoting j = | | | | | m | | Di1 | | (xi)6=yi] | | , show that | | | |  |  |
|  |  |  |  |  |  |  |  | i=1 | |  | [hj |  |  |  |  |  |  |
|  |  |  |  |  | @R(w) | | | | | = 2 j | | 1: | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | wj | | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  | @R(w) | | |  |  |  |  |  |  |
|  | Conclude that if j 1=2then | | | | | | | | | |  |  | =2. | |  |  |  |  |  |
|  | wj |  | (t+1) |  |  | (t) |  |
|  | 2 |  |  |  |  |  |  |  |  | |  |  | R(w |  | ) | R(w ) |
|  | Show that the update of AdaBoost guarantees | | | | | | | | | | | | |  |  |  |
|  | log(p |  |  | | | | | | | | | | | | |  |  |  |  |
|  | 1 4 ). Hint: Use the proof of Theorem [10.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page135). | | | | | | | | | | | | | |  |  |  |  |

Part IV

Advanced Theory

1. Rademacher Complexities

In Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54) we have shown that uniform convergence is a su cient condition for learnability. In this chapter we study the Rademacher complexity, which measures the rate of uniform convergence. We will provide generalization bounds based on this measure.

26.1 The Rademacher Complexity

Recall the de nition of an -representative sample from Chapter [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page54), repeated here for convenience.

definition 26.1 ( -Representative Sample) A training set S is called -representative (w.r.t. domain Z, hypothesis class H, loss function `, and distribution D) if

sup jLD(h) LS(h)j :

h2H

We have shown that if S is an =2 representative sample then the ERM rule is -consistent, namely, LD(ERMH(S)) minh2H LD(h) + .

To simplify our notation, let us denote

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| def | def | fz 7!`(h; z) : h 2 Hg; | | | |
| F = ` H = | |
| and given f 2 F, we de ne |  |  |  |  | m f(zi): |
| LD(f) = E [f(z)] ; | | | LS(f) = | 1 |
|  |
| z D |  |  |  |  | Xi |
|  |  |  | m =1 | |

We de ne the representativeness of S with respect to F as the largest gap be-tween the true error of a function f and its empirical error, namely,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| def | sup |  | LD(f) LS(f) |  | (26.1) |
| RepD(F; S) = | f2F | : |  |

Now, suppose we would like to estimate the representativeness of S using the sample S only. One simple idea is to split S into two disjoint sets, S = S1 [ S2; refer to S1 as a validation set and to S2 as a training set. We can then estimate the representativeness of S by

|  |  |  |
| --- | --- | --- |
| sup | LS1 (f) LS2 (f) : | (26.2) |
| f2F |  |  |

1. Rademacher Complexities

This can be written more compactly by de ning = ( 1; : : : ; m) 2 f 1gm to

be a vector such that S1 = fzi : i = 1g and S2 = fzi : i = 1g. Then, if we further assume that jS1j = jS2j then Equation ([26.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page375)) can be rewritten as

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 |  |  | m |  |
|  |  | Xi |  |
|  |  | 2F |  |
| m | | fsup | if(zi): | (26.3) |
|  |  |  | =1 |  |

The Rademacher complexity measure captures this idea by considering the ex-pectation of the above with respect to a random choice of . Formally, let F S be the set of all possible evaluations a function f 2 F can achieve on a sample S, namely,

1. S = f(f(z1); : : : ; f(zm)) : f 2 Fg:

Let the variables in be distributed i.i.d. according to P[ i = 1] = P[ i = 1] =

12 . Then, the Rademacher complexity of F with respect to S is de ned as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| F | = mE1 m | | | " f |  | i i | # |
|  | S) def | 1 |  |  |  | m |  |
| R( |  | sup | | f(z ) | : |
|  | f g |
|  |  |  |  | 2F | Xi |  |
|  |  |  |  |  |  | =1 |  |

More generally, given a set of vectors, A Rm, we de ne

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | m | | | "a | A =1 | # |
|  | 1 | |  |  | m |  |
| def | E |  | Xi |  |
| R(A) = |  |  | 2 | iai : |
|  |  | sup | |

(26.4)

(26.5)

The following lemma bounds the expected value of the representativeness of S by twice the expected Rademacher complexity.

lemma 26.2

E m [ RepD(F; S)] 2 E m R(F S):

S D S D

Proof Let S0 = fz10; : : : ; zm0g be another i.i.d. sample. Clearly, for all f 2 F, LD(f) = ES0[LS0(f)]. Therefore, for every f 2 F we have

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D |  | LS(f) = | S0 |  |  | LS(f) = | S0 |  |  | LS(f)]: |
| L (f) |  | E[LS0 | (f)] |  | E[LS0 | (f) |  |

Taking supremum over f 2 F of both sides, and using the fact that the supremum of expectation is smaller than expectation of the supremum we obtain

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| f2F | D |  |  |  | f2F | S0 | 0 |  |  |  | LS(f)] | |  |
| sup | L | (f) |  | LS(f) | = sup | E[LS | | (f) | |  | # |
|  |  |  |  |  | S0 " | f2F |  | 0 |  |  |  | LS(f) |
|  |  |  |  |  | E | sup | LS | | (f) | |  | : |

Taking expectation over S on both sides we obtain

" #

1. sup LD(f) LS(f)
2. f2F

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | S;S0 | | " | f2F | |  | 0 |  |  |  | # |  | (26.6) |
|  |  | E | |  | sup | | LS | | (f) |  | LS(f) | | # |  |
|  | m S;S0 | | | | | " f |  |  | m |  | i |  |  |
|  | 1 | |  |  |  |  |  |  |  |  |  |  |  |
| = |  | E |  | sup | |  | (f(z0) f(zi)) | | | |  | : |
|  |  |  |  |  |  |
|  |  |  |  |  |  |  | 2F | Xi | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | =1 |  |  |  |  |  |

|  |  |
| --- | --- |
| 26.1 The Rademacher Complexity | 377 |
|  |  |

Next, we note that for each j, zj and zj0 are i.i.d. variables. Therefore, we can replace them without a ecting the expectation:

2

E 4 sup

S;S0 f2F

2

E 4 sup

S;S0 f2F

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | (f(zj0) f(zj)) + | (f(zi0) f(zi)) | 13 | = |
| @ |  | X | A5 | (26.7) |
| 0 |  | i6=j | 13 |  |
| (f(zj) f(zj0)) + | (f(zi0) f(zi)) | : |
| @ |  | X | A5 |  |

i6=j

Let j be a random variable such that P[ j = 1] = P[ j = 1] = 1=2. From Equation ([26.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page377)) we obtain that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | S;SE0; j | | | | | 2 sup | | 0 j(f(zj0) f(zj)) + | | | | | | |  |  | (f(zi0) f(zi))1 3 | | | | | | | |  |
|  |  |  | 1 | | | 4 |  | @ | |  |  | 1 | |  | X | | |  |  |  | A 5 | | | |  |
|  |  |  |  |  |  | f2F | |  |  |  |  |  |  |  | i=j | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 | |  |  |  |  |  |  |  |  |  |
|  | = |  |  |  | (l.h.s. of Equation ([26.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page377))) + | | | | | | | |  | (r.h.s. of Equation ([26.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page377))) | | | | | | | | | | | (26.8) |
|  |  |  | 2 | 2 |
|  |  |  |  | S;S0 2 | | | f2F |  | 0 | j | |  |  |  | i=j | | | i | |  | 1 3 | | |  |  |
|  | = |  |  | E | | 4 | sup |  | @ | (f(z0 ) |  |  |  | X | | | |  |  | f(zi)) | A 5 | | | : |  |
|  |  |  |  |  |  |  | f(zj)) + | | |  |  | (f(z0) | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 | |  |  |  |  |  |  |  |  |  |
| Repeating this for all j we obtain that | | | | | | | | | | | | | | |  |  |  |  |  |  |  | |  |  | # |
| S;S0 | " f | | 2F | | | m |  | i | |  | # | S;S0; " f | | | | | 2F | m |  | i |  |  |
|  |  |  | Xi |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |
| E |  | sup | | | | (f(z0) | | |  | f(zi)) |  | = E | | |  | sup | |  | i(f(z0) | |  |  | f(zi)) | | : (26.9) |
|  |  |  |  |  |  | =1 |  |  |  |  |  |  |  |  |  |  |  | i=1 |  |  |  |  |  |  |  |
| Finally, | | f2F Xi | | | | | i(f(zi0) f(zi)) f2F Xi | | | | | | | | |  |  |  |  | f2F Xi | | if(zi) | | | |
|  |  | if(zi0) + | | | |
|  |  | sup | | | |  |  |  |  |  |  | sup | | |  |  |  |  |  | sup |  |  |  |  |  |

and since the probability of is the same as the probability of , the right-hand side of Equation ([26.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page377)) can be bounded by

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | " | Xi |  | f2F | # |
| S;S0 | ; | f2F | i | Xi |
| E |  | sup |  | if(z0) + sup | | if(zi) |

= m E[R(F S0)] + m E[R(F S)] = 2m E[R(F S)]:

S0 S S

The lemma immediately yields that, in expectation, the ERM rule nds a hypothesis which is close to the optimal hypothesis in H.

theorem 26.3 We have

E [LD(ERMH(S)) LS(ERMH(S))] 2 E R(` H S):

S Dm S Dm

Furthermore, for any h? 2 H

E [LD(ERMH(S)) LD(h?)] 2 E R(` H S):

S Dm S Dm

1. Rademacher Complexities

Furthermore, if h? = argminh LD(h) then for each 2 (0; 1) with probability of at least 1 over the choice of S we have

LD(ERMH(S)) LD(h?) 2 ES0 Dm R(` H S0) :

Proof The rst inequality follows directly from Lemma [26.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page376). The second in-equality follows because for any xed h?,

LD(h?) = E[LS(h?)] E[LS(ERMH(S))]:

S S

The third inequality follows from the previous inequality by relying on Markov's inequality (note that the random variable LD(ERMH(S)) LD(h?) is nonnega-tive). 

Next, we derive bounds similar to the bounds in Theorem [26.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page377) with a better dependence on the con dence parameter . To do so, we rst introduce the following bounded di erences concentration inequality.

lemma 26.4 (McDiarmid's Inequality) Let V be some set and let f : V m ! R be a function of m variables such that for some c > 0, for all i 2 [m] and for all x1; : : : ; xm; x0i 2 V we have

jf(x1; : : : ; xm) f(x1; : : : ; xi 1; x0i; xi+1; : : : ; xm)j c:

Let X1; : : : ; Xm be m independent random variables taking values in V . Then, with probability of at least 1 we have

q

jf(X1; : : : ; Xm) E[f(X1; : : : ; Xm)]j c ln 2 m=2:

On the basis of the McDiarmid inequality we can derive generalization bounds with a better dependence on the con dence parameter.

theorem 26.5 Assume that for all z and h 2 H we have that j`(h; z)j c. Then,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | With probability of at least 1 , for all h 2 H, | | | | | | | | |  |  | r | |  |  |  |  |  |  |  |  |  |  |  |
|  | D |  |  |  |  | S0 | Dm |  | H |  |  |  |  |  |  | m | | | | |  |  |
|  | L | (h) |  | LS(h) |  | 2 | E | R(` |  | S0) + c | | |  |  | 2 ln(2= ) | | | | | | : | |  |  |
|  |  |  |  |  |  |  | |  | |  |  |  |  |
|  | In particular, this holds for h = ERMH(S). | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2. | With probability of at least 1 , for all h 2 H, | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | LD(h) LS | | | (h)2 R(` H S) + 4 cr | | | | | | 2 | |  |  | m | | | : | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | ln(4= ) | | | | | | | | | |  |  |
|  | In particular, this holds for h = ERMH(S). | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3. | For any h?, with probability of at least 1 , | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | LD(ERMH(S)) LD(h?) | | | | | | 2 R(` H S) + 5 c r | | | | | | | | |  |  | | | | | |  |  |
|  |  | 2 lnm | | | | | | : | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | (8= ) | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| 26.1 The Rademacher Complexity | 379 |
|  |  |

Proof First note that the random variable RepD(F; S) = suph2H (LD(h) LS(h)) satis es the bounded di erences condition of Lemma [26.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page378) with a constant 2c=m. Combining the bounds in Lemma [26.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page378) with Lemma [26.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page376) we obtain that with probability of at least 1 ,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rep | ( | F | ; S) | E | Rep | ( | F | ; S) + c | r | 2 ln(2= ) | |  | 2 E R(` | H | S0) + c | 2 ln(2= ) | : |
| m |  | m |
|  | D |  |  | D |  | S0 | r |  |

The rst inequality of the theorem follows from the de nition of RepD(F; S). For the second inequality we note that the random variable R(` H S) also satis es the bounded di erences condition of Lemma [26.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page378) with a constant 2c=m. Therefore, the second inequality follows from the rst inequality, Lemma [26.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page378), and the union bound. Finally, for the last inequality, denote hS = ERMH(S) and note that

|  |  |
| --- | --- |
| LD(hS) LD(h?) |  |
| = LD(hS) LS(hS) + LS(hS) LS(h?) + LS(h?) LD(h?) |  |
| (LD(hS) LS(hS)) + (LS(h?) LD(h?)) : | (26.10) |

The rst summand on the right-hand side is bounded by the second inequality of the theorem. For the second summand, we use the fact that h? does not depend on S; hence by using Hoe ding's inequality we obtain that with probaility of at least 1 =2,

|  |  |  |  |
| --- | --- | --- | --- |
| LS(h?) LD(h?) c r |  |  | (26.11) |
| 2m : | |
|  | ln(4= ) | |  |
|  |  |  |  |

Combining this with the union bound we conclude our proof.

The preceding theorem tells us that if the quantity R(` H S) is small then it is possible to learn the class H using the ERM rule. It is important to emphasize that the last two bounds given in the theorem depend on the speci c training set S. That is, we use S both for learning a hypothesis from H as well as for estimating the quality of it. This type of bound is called a data-dependent bound.

26.1.1 Rademacher Calculus

Let us now discuss some properties of the Rademacher complexity measure. These properties will help us in deriving some simple bounds on R(` H S) for speci c cases of interest.

The following lemma is immediate from the de nition.

lemma 26.6 For any A Rm, scalar c 2 R, and vector a0 2 Rm, we have

R(fc a + a0 : a 2 Ag) jcj R(A):

The following lemma tells us that the convex hull of A has the same complexity as A.

380 Rademacher Complexities

lemma 26.7 Let A be a subset of Rm and let A0 = fPN ja(j) : N 2

j=1

N; 8j; a(j) 2 A; j 0; k k1 = 1g. Then, R(A0) = R(A).

Proof The main idea follows from the fact that for any vector v we have

|  |  |  |
| --- | --- | --- |
| k k | N |  |
| Xj | = max vj: |
| sup | jvj |
| 0: 1=1 | =1 | j |
|  |

Therefore,

m R(A0) = E

= E

sup sup

0:k k1=1 a(1);:::;a(N)

|  |  |  |
| --- | --- | --- |
| k k | N |  |
| Xj | j sup |
| sup |  |
| 0: 1=1 | =1 | a(j) |
|  |  |
| m |  |  |

1. N
2. i X ja(ij) i=1 j=1

m

1. ia(ij)

i=1

|  |  |
| --- | --- |
| 2 | Xi |
| = E sup | iai |
| a A | =1 |
|  |
| = m R(A); | |
| and we conclude our proof. |  |

The next lemma, due to Massart, states that the Rademacher complexity of a nite set grows logarithmically with the size of the set.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | lemma) | Let A = | | | f | a | | ; : : : ; a | | N g | be a nite set of vectors | |
| lemma 26.8 (Massart N | | | |  |  |  |  | 1 |  |  |  |  |
| in Rm. De ne a = |  | 1 | Pi=1 ai. Then, | |  |  |  |  |  |  |  |  |  |  |
| N | |  |  |  |  |  |  |  | | |  |
|  |  |  |  |  |  |  | k | p | 2 log(N) | | | |
|  |  |  | R(A) | a2A k |  | a | | m | |  |  |
|  |  |  |  | max | a |  |  |  |  |  |  | : |
|  |  |  |  |  |  |  |  |  |  |

Proof Based on Lemma [26.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page379), we can assume without loss of generality that a = 0. Let > 0 and let A0 = f a1; : : : ; aN g. We upper bound the Rademacher complexity as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | a2A0h |  | i |  |  |  | a2A0 |  |
| mR(A0) = E | max | ; a |  | = | E | log | max eh ;ai |  |

1. !#

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| E | | log |  |  | X eh ;ai |  |
|  |  |  |  |  | a2A0 |  |
|  |  |  |  |  | #! |
|  |  |  |  | " | |
|  |  |  | |  | X |  |
| log |  | eh ;ai | // Jensen's inequality |
|  | E | |  |
|  |  |  |  |  | a2A0 | ! |
|  |  | a |  |  | A0 i=1 i |
|  |  |  |  |  | m |  |
|  |  | X Y | | | |  |
| = log | |  |  |  | E [e iai ] ; | |
|  |  |  | 2 | |  |  |

where the last equality occurs because the Rademacher variables are indepen-dent. Next, using Lemma [A.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page421) we have that for all ai 2 R,

E e iai = exp(ai) + exp( ai) exp(a2i=2);

i 2

|  |  |
| --- | --- |
| 26.1 The Rademacher Complexity | 381 |
|  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| and therefore |  | a2A0 i=1 exp a2i ! | | | | | | |  | a2A0 |  | kak2=2 |  | ! | |
| mR(A0) log | | = log | exp |  |
|  |  |  |  | m | 2 | |  |  |  |  |  |  |  |  |  |
|  |  | X Y | | |  |  |  | X |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | log | j | A | 0j a2A0 | k k | | | 2=2 | = log(jA0j) + a2A0 k | | | | k | | 2=2): |
|  |  | max exp |  | a | |  |  |  | max( a | |  |

Since R(A) = 1 R(A0) we obtain from the equation that

log(jAj) + 2 maxa2A(kak2=2)

R(A) :

p

Setting = 2 log(jAj)= maxa2A kak2 and rearranging terms we conclude our proof. 

The following lemma shows that composing A with a Lipschitz function does not blow up the Rademacher complexity. The proof is due to Kakade and Tewari.

lemma 26.9 (Contraction lemma) For each i 2 [m], let i : R ! R be a - Lipschitz function, namely for all ; 2 R we have j i( ) i( )j j j. For a 2 Rm let (a) denote the vector ( 1(a1); : : : ; m(ym)). Let A = f (a) :

1. 2 Ag. Then,

R( A) R(A):

Proof For simplicity, we prove the lemma for the case = 1. The case 6= 1 will follow by de ning 0 = 1 and then using Lemma [26.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page379). Let Ai = f(a1; : : : ; ai 1; i(ai); ai+1; : : : ; am) : a 2 Ag. Clearly, it su ces to prove that for any set A and all i we have R(Ai) R(A). Without loss of generality we will prove the latter claim for i = 1 and to simplify notation we omit the subscript from 1. We have

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | "a | A1 | =1 |  | # |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 2 |  | m |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| mR(A1) = E | | |  | Xi |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sup | | iai | | |  |  | # |  |  |  |  |  |  |  |  |  |  |
| "a | | | | A |  |  |  | m |  |  |  |  |  |  |  |  |  |  |  |
| = E | | | 2 |  |  |  |  | Xi |  |  |  |  |  |  |  |  |  |  |  |  |
| sup 1 (a1) + | | | | | iai | | |  |  |  |  |  |  |  |  |  |  |
| 2 | | | 2;:::; m "a A | | | |  | =2 |  |  |  |  | ! | a A | |  | |  |  | !# |
|  |  |  | m | |  |  | m |
|  | 1 |  | E | | 2 |  |  |  |  | X | | |  | 2 |  |  |  |  | Xi |  |
| = |  | sup | | (a1) + | | |  |  |  | iai | + sup | |  |  | (a1) + | | iai |
|  |  |  |  |  |  |
| 2 | | | 2;:::; m "a;a0 | | |  | A |  |  | i=2 | | | 1 |  |  |  |  |  | =2 |  |
|  |  |  |  |  |  | m |  |  |  | m | i!# |  |
|  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| = |  | E | | sup | | | (a1) | |  | (a0 ) + | | | iai | | | + | | ia0 |  |
|  |  |  |
|  |  |  |  |  |  | 2 |  |  |  |  |  |  |  | X |  |  |  | Xi |  |  |
| 2 | | | 2;:::; m "a;a0 | | |  | A | j | 1j | | |  |  | i=2 |  |  |  | =2 |  |  |
|  |  | m |  |  | m |  | i!# | |  |
| 1 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | X |  | Xi | | |  |  |  |
|  |  |  |  |  |  | 2 |  |  |  |  |  |  |  |  |  |  |
|  |  |  | E | | sup | | | a1 |  | a0 |  | +iai + | | | |  | ia0 | | ; | (26.12) |
|  |  |  |  |  |  |  |  |  |  |  |  |  | i=2 |  |  | =2 | |  |  |  |

where in the last inequality we used the assumption that is Lipschitz. Next, we note that the absolute value on ja1 a01j in the preceding expression can

1. Rademacher Complexities

be omitted since both a and a0 are from the same set A and the rest of the

expression in the supremum is not a ected by replacing a and a0. Therefore,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | | | 2;:::; m "a;a0 A | | 1 | i!# |
|  | 1 | | |  |  | m | m |
|  |  |  | Xi | X |
|  |  |  |  |  | 2 |
| mR(A1) |  |  |  | E | sup | a1 a0 +iai + | ia0 : (26.13) |
|  |  |  |  |  |  | =2 | i=2 |

But, using the same equalities as in Equation ([26.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page381)), it is easy to see that the right-hand side of Equation ([26.13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page382)) exactly equals m R(A), which concludes our proof. 

26.2 Rademacher Complexity of Linear Classes

In this section we analyze the Rademacher complexity of linear classes. To sim-plify the derivation we rst de ne the following two classes:

H1 = fx 7!wh; xi : kwk1 1g ; H2 = fx 7!wh; xi : kwk2 1g: (26.14)

The following lemma bounds the Rademacher complexity of H2. We allow the xi to be vectors in any Hilbert space (even in nite dimensional), and the bound does not depend on the dimensionality of the Hilbert space. This property becomes useful when analyzing kernel methods.

lemma 26.10 Let S = (x1; : : : ; xm) be vectors in a Hilbert space. De ne: H2

1. = f(hw; x1i; : : : ; hw; xmi) : kwk2 1g. Then,

max kx k

R(H2 S) pi i 2 :

m

Proof Using Cauchy-Schwartz inequality we know that for any vectors w; v we have hw; vi kwk kvk. Therefore,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| H | "a |  | 2 S | | |  | =1 | |  |  | # |  |  |
|  |  | 2H | | | |  | m | |  |  |  |  |  |
| mR( 2 | S) = E | Xi | | | iai | |  |  |  |
| sup | | | |  |  |  |  |  | (26.15) |
|  | "w: w | | |  |  | 1 |  | =1 | | h |  |  | i# |
|  |  | k |  | k | |  |  | m | |  |  |  |  |
|  | = E |  |  | Xi | | | i w; xi | | |  |
|  | sup | | | |  |  |  |  |  |
|  | "w: w | | |  |  | 1 | h |  |  | m |  | i# | |
|  | = E | k |  | k | |  |  |  |  | Xi | ixi |  |  |
|  | sup | | | |  |  | w; | |  |  |  |
|  | "k | |  |  |  |  |  |  |  | =1 |  |  |  |
|  |  |  |  |  |  | k # | | |  |  |  |
|  |  | m | |  |  |  |  |  |  |  |  |  |  |
|  | E | Xi | | | |  |  |  |  |  |  |  |  |
|  |  |  |  | ixi 2 | | | | | : |  |  |  |
|  |  | =1 | | |  |  |  |  |  |  |  |  |  |

Next, using Jensen's inequality we have that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 2 |  |  |  | 1=2 | 3 |  |  |  |  |  | 1=2 |
| " | |  | # |  | 0 |  |  | 1 |  | 0 2 | |  |  | 31 |
|  | m |  |  |  | m |  |  | 2 |  |  |  | m |  |  | 2 |  |
|  | i=1 |  | 2 |  | i=1 |  | 2 | |  |  |  | i=1 |  | 2 | |  |
| E | X | ixi |  | = E | X | ixi |  |  |  |  | E | X | ixi |  |  | (26:.16) |
|  |  | 6 |  |  |  | 7 |  |  |  |
|  |  |  |  |  | 4@ |  | A | |  | 5 | @ 4 | |  | 5A | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| 26.3 Generalization Bounds for SVM | 383 |
|  |  |

Finally, since the variables 1; : : : ; m are independent we have

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| "k | m | k2 | # |  |  | 2 | X | h |  | i | 3 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| E | X | 2 = E | | | | 4 |  |  |  | 5 |  |  |  |  |  |  |
| ixi | i j | | xi; xj |  | m |  |  |  |  |  |
|  | i=1 |  |  |  |  | i;j |  |  |  |  |  |  |  |  |
|  |  |  |  | X | |  |  |  |  | X |  |  |  |  |  |
|  |  |  |  |  |  | E [ i j] + | | | |  |  |  |
|  |  |  |  | = |  |  | xi; xj | h | xi; xi | i | E 2 | |  |
|  |  |  |  |  |  | h | | i |  |  |  |  |  | i |  |
|  |  |  |  |  | i6=j | |  |  |  |  |  | i=1 |  |  |  |  |  |
|  |  |  |  |  | m | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | = | Xi | | kxik22 | m maxi | | | | kxik22: | |  |  |  |  |
|  |  |  |  | =1 | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Combining this with Equation ([26.15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page382)) and Equation ([26.16](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page382)) we conclude our proof. 

Next we bound the Rademacher complexity of H1 S.

lemma 26.11 Let S = (x1; : : : ; xm) be vectors in Rn. Then,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R(H1 S) | | | | |  | maxi kxik1 | | | | | | | | | |  | r |  |  | |  |  | |  |  |  |  |  |  |  |
|  |  | 2 | |  | m | | |  | : | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | log(2n) | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | |  | | | | |  | | |  | |  |  |
| Proof Using Holder's inequality we know that for any vectors w; v we have | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| hw; vi kwk1 kvk1. Therefore, | | | | | | " |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | # | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | m | | | |  |  |  |  |  |  |  |  |  |  |
| mR( | H | 1 |  | S) = E | | a | sup | | | |  |  |  |  |  | iai | | | | |  |  |  |  |  |  |  |
|  |  |  |  | 2 | H1 | |  | S i=1 | | | | | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | " |  |  |  |  |  | X | | | | |  |  |  |  |  |  |  | i# | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | m | |  |  |  |  |  |  |  |
|  |  |  |  |  | = E | w: | | sup | | | |  |  | 1 i=1 | | | |  | i | | h | w; xi | | |
|  |  |  |  |  |  |  | w | 1 | |  | |  |  |  |  |  |  |
|  |  |  |  |  |  | " |  | k | |  | k |  | X | | | | |  | m | |  |  | i# | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | = E |  |  | sup | | | |  |  | 1h | | w; | |  |  |  |  | ixi | |
|  |  |  |  |  |  | w: w 1 | | | | | |  |  |  |  |  | =1 | | |  |  |
|  |  |  |  |  |  |  |  | k | |  | k | | | |  |  |  |  |  | Xi | | | | |  |  |  |  |  |  |
|  |  |  |  |  |  | "k | | | m | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | E |  |  | ixi | | | | | k1 | | | |  | : | |  |  |  | (26.17) | | | | |
|  |  |  |  |  |  | =1 | | |  |  |  |  | # | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | Xi | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| For each j 2 [n], let vj = (x1;j; : : : ; xm;j) 2 Rm. Note that kvjk2 p | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | maxi kxik1. |
| m |
| Let V = fv1; : : : ; vn; v1; : : : ; vng. The right-hand side of Equation ([26.17](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page383)) is | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| m R(V ). Using Massart lemma (Lemma [26.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page380)) we have that | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| R(V ) maxi kxik1 | | | | | | | | | | | p | | |  | | | | | | | | | |  |  |  |  |  |  |  |
| 2 log(2n)=m; | | | | | | | | | | |  |  |  |  |  |  |
| which concludes our proof. | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

26.3 Generalization Bounds for SVM

In this section we use Rademacher complexity to derive generalization bounds for generalized linear predictors with Euclidean norm constraint. We will show how this leads to generalization bounds for hard-SVM and soft-SVM.

1. Rademacher Complexities

We shall consider the following general constraint-based formulation. Let H = fw : kw k2 Bg be our hypothesis class, and let Z = X Y be the examples domain. Assume that the loss function ` : H Z ! R is of the form

|  |  |
| --- | --- |
| `(w; (x; y)) = (hw; xi; y); | (26.18) |

where : R Y ! R is such that for all y 2 Y, the scalar function a 7! (a; y) is -Lipschitz. For example, the hinge-loss function, `(w; (x; y)) = maxf0; 1 yhw; xig, can be written as in Equation ([26.18](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384)) using (a; y) = maxf0; 1 yag, and note that is 1-Lipschitz for all y 2 f 1g. Another example is the absolute loss function, `(w; (x; y)) = jhw; xi y j, which can be written as in Equation ([26.18](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384)) using (a; y) = ja yj, which is also 1-Lipschitz for all y 2 R.

The following theorem bounds the generalization error of all predictors in H using their empirical error.

theorem 26.12 Suppose that D is a distribution over X Y such that with probability 1 we have that kxk2 R. Let H = fw : kwk2 Bg and let

1. : H Z ! R be a loss function of the form given in Equation ([26.18](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384)) such that for all y 2 Y, a 7! (a; y) is a -Lipschitz function and such that

maxa2[ BR;BR] j (a; y)j c. Then, for any 2 (0; 1), with probability of at least 1 over the choice of an i.i.d. sample of size m,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8w 2 H; LD(w) LS(w) + | 2pm + cr | | | | |  | m | | : |
|  |  | BR | | | | 2 ln(2= ) | | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Proof Let F = f(x; y) 7! (hw; xi; y) : w 2 Hg. We will show that with p

probability 1, R(F S) BR= m and then the theorem will follow from Theorem [26.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page378). Indeed, the set F S can be written as

F S = f( (hw; x1i; y1); : : : ; (hw; xmi; ym)) : w 2 Hg;

and the bound on R(F S) follows directly by combining Lemma [26.9](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page381), Lemma [26.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page382), and the assumption that kxk2 R with probability 1. 

We next derive a generalization bound for hard-SVM based on the previous theorem. For simplicity, we do not allow a bias term and consider the hard-SVM problem:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w | k | w | k | 2 s.t. | 8 | ih | w; x | ii | 1 | (26.19) |
| argmin |  |  | i; y |  |  |

theorem 26.13 Consider a distribution D over X f 1g such that there exists some vector w? with P(x;y) D[yhw?; xi 1] = 1 and such that kxk2 R with probability 1. Let wS be the output of Equation ([26.19](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384)). Then, with probability of at least 1 over the choice of S Dm, we have that

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (x;yP) D[y 6= sign(hwS; xi)] | 2 Rpkm | | | k | | + (1 + R kw?k)r | m | | : |
|  |  | w? | |  |  |  | 2 ln(2= ) | | |
|  |  |  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| 26.3 Generalization Bounds for SVM | 385 |
|  |  |

Proof Throughout the proof, let the loss function be the ramp loss (see Sec-tion [15.2.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page209)). Note that the range of the ramp loss is [0; 1] and that it is a 1-Lipschitz function. Since the ramp loss upper bounds the zero-one loss, we have that

1. [y 6= sign(hwS; xi)] LD(wS):

(x;y) D

Let B = kw?k2 and consider the set H = fw : kwk2 Bg. By the de nition of hard-SVM and our assumption on the distribution, we have that wS 2 H with probability 1 and that LS(wS) = 0. Therefore, using Theorem [26.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384) we have that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LD(wS) LS(wS) + | 2pm + r | | | |  |  |  | : |
|  | m | |
|  | BR | | | |  | 2 ln(2= ) | | |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Remark 26.1 Theorem [26.13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384) implies that the sample complexity of hard-SVM

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | R | 2 | w? |  | 2 |  |  |  |
| grows like |  | k 2 | k |  | . Using a more delicate analysis and the separability assump- | | |
|  |  |  |
|  |  |  |  |  |  |  | R2 kw?k2 |  |
| tion, it is possible to improve the bound to an order of | | | | | | | . |
|  |  |  |  |  |  |  |  |  |

The bound in the preceding theorem depends on kw?k, which is unknown. In the following we derive a bound that depends on the norm of the output of SVM; hence it can be calculated from the training set itself. The proof is similar to the derivation of bounds for structure risk minimization (SRM).

theorem 26.14 Assume that the conditions of Theorem [26.13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384) hold. Then, with probability of at least 1 over the choice of S Dm, we have that

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (x;yP) | [y = sign( w ; x | | | )] |  | 4RkwSk | | | + |
|  |
|  |  |  |  |
| 6 h | S | i |  |  | pm | | |  |
| D | |  |  |  |  |  |  |  |  |

s

|  |  |  |  |
| --- | --- | --- | --- |
| ln( | 4 log2(kwSk) | ) |  |
|  |  | | : |
|  | m | |
|  |  |

Proof For any integer i, let B = 2i, H = fw : kwk B g, and let = .

i i i i 2i2

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8w 2 Hi; LD(w) LS(w) + 2pmi | | | | | | | | | | | | | | | | | | | | | | | | | | | + r | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | m i | ) |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | B R | | | | | |  |  |  | 2 ln(2= | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | 1 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Applying the union bound and using | | | | | | | | | |  | i=1 iwe obtain that with probability | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| of at least 1 this holds for all i. | | | | | | | | Therefore, for all w, if we let i = | | | | | | | | | | | | | | | | | | | | | | | | | | | | d | log | | | | ( | w ) | e |
|  |  |  | P2 | | | |  |  |  | (4 log2(kwk)) | | | | | | | | | | 2 |  |  |  |  |  |  |  |  |  |  | 2 | k | k |
|  |  |  | 2 | | |  |  |  |  | (2i) | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| then w 2 Hi, Bi 2kwk, and | | | i | | |  | = | |  |  |  | | |  |  | | |  |  |  |  |  |  |  | | |  |  | . Therefore, | | | | | | |  |  |  |  |  |  |  |  |
| LD(w) LS(w) + | 2pmi | | | | | |  | + r | | | |  |  | |  |  |  |  | | |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | |  |  |  | m | | | i | | ) | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | B R | | | | | |  |  |  |  | 2 ln(2= | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | |  |  | | |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | |  |  | |  |  |  |  | r | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| S |  | kpm | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | m | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| L (w) + | 4 | wkR | | | | | | | + | | | | | |  |  | 4(ln(4 log2(kwk)) + ln(1= )) | | | | | | | | | | | | | | | | | | | | | : | |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | | | | | | | | | | | | | | | | | | | | |  |  |  |  |

In particular, it holds for wS, which concludes our proof.

1. Rademacher Complexities

Remark 26.2 Note that all the bounds we have derived do not depend on the dimension of w. This property is utilized when learning SVM with kernels, where the dimension of w can be extremely large.

26.4 Generalization Bounds for Predictors with Low `1 Norm

In the previous section we derived generalization bounds for linear predictors with an `2-norm constraint. In this section we consider the following general `1-norm constraint formulation. Let H = fw : kwk1 Bg be our hypothesis class, and let Z = X Y be the examples domain. Assume that the loss function,

1. : H Z ! R, is of the same form as in Equation ([26.18](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384)), with : R Y ! R being -Lipschitz w.r.t. its rst argument. The following theorem bounds the generalization error of all predictors in H using their empirical error.

theorem 26.15 Suppose that D is a distribution over X Y such that with probability 1 we have that kxk1 R. Let H = fw 2 Rd : kwk1 Bg and let ` : H Z ! R be a loss function of the form given in Equation ([26.18](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384)) such that for all y 2 Y, a 7! (a; y) is an -Lipschitz function and such that maxa2[ BR;BR] j (a; y)j c. Then, for any 2 (0; 1), with probability of at least 1 over the choice of an i.i.d. sample of size m,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8w 2 H; LD(w) LS(w) + 2 BRr | 2 | | m | | + cr |  | m |  | : |
|  |  |  | log(2d) | |  |  | 2 ln(2= ) | | |
|  |  |  |  |  |  |  |  |  |  |

Proof The proof is identical to the proof of Theorem [26.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384), while relying on Lemma [26.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page383) instead of relying on Lemma [26.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page382). 

It is interesting to compare the two bounds given in Theorem [26.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384) and The-orem [26.15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page386). Apart from the extra log(d) factor that appears in Theorem [26.15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page386), both bounds look similar. However, the parameters B; R have di erent meanings in the two bounds. In Theorem [26.12](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page384), the parameter B imposes an `2 constraint on w and the parameter R captures a low `2-norm assumption on the instances. In contrast, in Theorem [26.15](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page386) the parameter B imposes an `1 constraint on w (which is stronger than an `2 constraint) while the parameter R captures a low `1-norm assumption on the instance (which is weaker than a low `2-norm as-sumption). Therefore, the choice of the constraint should depend on our prior knowledge of the set of instances and on prior assumptions on good predictors.

26.5 Bibliographic Remarks

The use of Rademacher complexity for bounding the uniform convergence is due to (Koltchinskii & Panchenko 2000, Bartlett & Mendelson 2001, Bartlett

1. Mendelson 2002). For additional reading see, for example, (Bousquet 2002, Boucheron, Bousquet & Lugosi 2005, Bartlett, Bousquet & Mendelson 2005).

|  |  |
| --- | --- |
| 26.5 Bibliographic Remarks | 387 |
|  |  |

Our proof of the concentration lemma is due to Kakade and Tewari lecture notes. Kakade, Sridharan & Tewari (2008) gave a uni ed framework for deriving bounds on the Rademacher complexity of linear classes with respect to di erent assumptions on the norms.

1. Covering Numbers

In this chapter we describe another way to measure the complexity of sets, which is called covering numbers.

27.1 Covering

definition 27.1 (Covering) Let A Rm be a set of vectors. We say that A is r-covered by a set A0, with respect to the Euclidean metric, if for all a 2 A there exists a0 2 A0 with ka a0k r. We de ne by N(r; A) the cardinality of the smallest A0 that r-covers A.

Example 27.1 (Subspace) Suppose that A Rm, let c = maxa2A kak, and as-p

sume that A lies in a d-dimensional subspace of Rm. Then, N(r; A) (2c d=r)d. To see this, let v1; : : : ; vd be an orthonormal basis of the subspace. Then, any

Pd

a 2 A can be written as a = i=1 ivi with k k1 k k2 = kak2 c. Let

2 R and consider the set

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A0 = | | ( d |  | i0vi : 8i; i0 2 f c; c + ; c + 2 ; : : : ; cg) : | | | | | | | | | | | | | | | | | |
|  |  | Xi | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | =1 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | P | | | d | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Given a 2 A s.t. a = | | |  |  | i=1 ivi with k k1 c, there exists a0 2 A0 such that | | | | | | | | | | | | | | | | |
| ka a0k | | | 2 | = k Xi( i0 i)vik | | | | 2 | |  | 2 | Xi | | kvik | | | | 2 |  | 2 | d: |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| p |  | ; then ka a0k r and therefore A0 | | | | | | | | | | | |  |  |  |  |  |  |  |  |
| Choose = r= | d |  | is an r-cover of A. Hence, | | | | | | |
|  |  | N(r; A) jA0j = | | | |  |  | | d | |  |  | p | |  | ! | d | |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  | 2c |  | = |  |  | 2c d | | |  | : |  |  |  |
|  |  |  |  |  | r | | |  |  |  |  |

27.1.1 Properties

The following lemma is immediate from the de nition.

lemma 27.2 For any A Rm, scalar c > 0, and vector a0 2 Rm, we have

8r > 0; N(r; fc a + a0 : a 2 Ag) N(cr; A):

|  |  |
| --- | --- |
| 27.2 From Covering to Rademacher Complexity via Chaining | 389 |
|  |  |

Next, we derive a contraction principle.

lemma 27.3 For each i 2 [m], let i : R ! R be a -Lipschitz function; namely, for all ; 2 R we have j i( ) i( )j j j. For a 2 Rm let (a) denote the vector ( 1(a1); : : : ; m(am)). Let A = f (a) : a 2 Ag. Then,

N( r; A) N(r; A):

Proof De ne B = A. Let A0 be an r-cover of A and de ne B0 = A0. Then, for all a 2 A there exists a0 2 A0 with ka a0k r. So,

X X

k (a) (a0)k2 = ( i(ai) i(a0i))2 2 (ai a0i)2 ( r)2:

i i

Hence, B0 is an ( r)-cover of B.

27.2 From Covering to Rademacher Complexity via Chaining

The following lemma bounds the Rademacher complexity of A based on the covering numbers N(r; A). This technique is called Chaining and is attributed to Dudley.

lemma 27.4 Let c = mina maxa2A ka ak. Then, for any integer M > 0,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | c 2 M | | | | 6 c M | |  |  |  |
| R(A) | 2 kqlog(N(c 2 k; A)): | | |
| pm + | | | | m k=1 | |
|  |  |  |  |  |  | X |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Proof Let a be a minimizer of the objective function given in the de nition of c. On the basis of Lemma [26.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page379), we can analyze the Rademacher complexity assuming that a = 0.

Consider the set B0 = f0g and note that it is a c-cover of A. Let B1; : : : ; BM be sets such that each Bk corresponds to a minimal (c 2 k)-cover of A. Let a = argmaxa2Ah ; ai (where if there is more than one maximizer, choose one in an arbitrary way, and if a maximizer does not exist, choose a such that h ; a i is close enough to the supremum). Note that a is a function of . For each k, let b(k) be the nearest neighbor of a in Bk (hence b(k) is also a function of ). Using the triangle inequality,

kb(k) b(k 1)k kb(k) a k + ka b(k 1)k c (2 k + 2 (k 1)) = 3 c 2 k:

For each k de ne the set

B^k = f(a a0) : a 2 Bk; a0 2 Bk 1; ka a0k 3 c 2 kg:

1. Covering Numbers

We can now write

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | R(A) = | | | | | | 1 | | Eh ; a i | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | m |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | = m E | | | | | "h ; ab(M)i + h ; b(k) b(k 1)i# | | | | | | | | | | | | | | | | | | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  | 1 | |  |  |  |  |  |  |  |  |  |  |  | M | | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X | | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | m E | | | | |  |  |  |  |  |  |  |  |  |  | k=1 | | | | m E "a2B^kh | | | | | | | i# |  |
|  |  |  |  |  |  |  |  |  | hk k ka b ki | | | | | | | | | | | k=1 | | |  |
|  |  |  |  |  |  |  |  |  |  |  | 1 | |  |  |  |  |  |  |  |  |  |  |  |  | M | | | 1 |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | (M) |  | X | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | + | |  |  |  |  | sup | | | ; a | | : |  |
|  |  |  |  |  |  |  | p | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Since k k = | | | | | | |  | and ka b(M)k c 2 M , the rst summand is at most | | | | | | | | | | | | | | | | | | | | | | | | | | |
| m |
| pcm | | | 2 M . Additionally, by Massart lemma, | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
|  | m E | | | a2B^k | h |  |  | i | | |  |  |  |  | |  | p | | |  |  |  |  |  |  |  |  |  |  |  | p | |  |  |  | : |
|  |  |  |  |  |  |  |  |  |  | m |  |  |  |  |  |  |  |  | m |
| 1 | | |  | sup |  | ; a | |  |  |  |  | 3 c 2 | |  |  | k |  |  |  | 2 log(N(c 2 k; A)2) | | | | | | | = 6 c 2 | | |  | k |  | log(N(c 2 k; A)) | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  | | |  |  |  |  |  |  |  |  |
| Therefore, | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  | c 2 M | | | | | | |  | 6c M | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | R(A) | | | | | |  | 2 k | qlog(N(c2 k; A)): | | | | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  | pm | | | | | + m k=1 | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | X |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

As a corollary we obtain the following:

lemma 27.5 Assume that there are ; > 0 such that for any k 1 we have

q

Then,

Proof

P1

k=1

log(N(c2 k; A)) + k:

6c

R(A) m ( + 2 ) :

The bound follows from Lemma [27.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page389) by taking M ! 1 and noting that

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | k | = 1 and P | 1 | k | = 2. |
|  | k=1 k2 |  |

Example 27.2 Consider a set A which lies in a d dimensional subspace of Rm

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | p |  |  | d |
| fore, for any k, | a2A k k | d |
| r |  |
| and such that c = max | a . We have shown that N(r; A) |  | 2c |  | . There- |
|  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| q |  |  | |  |  |  |  | r | d log | | |  |  |  |  |
| log(N(c2 k; A)) | | | | | 2k+1 | | | |
|  |  |  |  |  |  |  |  | q |  |  |  | |  |  |  |
|  |  |  |  |  |  |  |  | d log(2p | | | |  | ) |
|  |  |  |  |  |  |  | d |  |
|  |  |  |  |  |  |  | q | |  |  |  | | | |  |
|  |  |  |  |  |  | d log(2p | | | |  | ) |
|  |  |  |  |  |  | d |  |
| Hence Lemma [27.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page390) yields | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6c q | |  |  |  |  | |  | p | |  |  |  |  |  |  |
|  | p |  |  | |  |  |  |  |

d

p

1. k d

p

1. d k:

!

p

1. d log(d)

R(A) m d log(2 d) + 2 d = O

m

:

|  |  |
| --- | --- |
| 27.3 Bibliographic Remarks | 391 |
|  |  |

27.3 Bibliographic Remarks

The chaining technique is due to Dudley (1987). For an extensive study of cover-ing numbers as well as other complexity measures that can be used to bound the rate of uniform convergence we refer the reader to (Anthony & Bartlet 1999).

jAj

1. Proof of the Fundamental Theorem of Learning Theory

In this chapter we prove Theorem [6.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72) from Chapter [6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page67). We remind the reader the conditions of the theorem, which will hold throughout this chapter: H is a hypothesis class of functions from a domain X to f0; 1g, the loss function is the 0 1 loss, and VCdim(H) = d < 1.

We shall prove the upper bound for both the realizable and agnostic cases and shall prove the lower bound for the agnostic case. The lower bound for the realizable case is left as an exercise.

28.1 The Upper Bound for the Agnostic Case

For the upper bound we need to prove that there exists C such that H is agnostic PAC learnable with sample complexity

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| mH( ; ) C | | d + ln(1= ) | : |  |  |
|  |  |  |
| 2 |  |
| We will prove the slightly looser bound: | |  |  |  |  |
| mH( ; ) C | d log(d= ) + ln(1= ) | | | |  |
|  |  |  | : | (28.1) |
|  | 2 |  |

The tighter bound in the theorem statement requires a more involved proof, in which a more careful analysis of the Rademacher complexity using a technique called \chaining" should be used. This is beyond the scope of this book.

To prove Equation ([28.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page392)), it su ces to show that applying the ERM with a

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sample size |  |  | log |  |  |  | + |  |  |  |
| m 4 | 32d | | 64d | | | 8 | | (8d log(e=d) + 2 log(4= )) |
|  |  |  |  |  |  |  |
| 2 | |  | 2 | | 2 | |

yields an ; -learner for H. We prove this result on the basis of Theorem [26.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page378). Let (x1; y1); : : : ; (xm; ym) be a classi cation training set. Recall that the Sauer-

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Shelah lemma tells us that if VCdim(H) = d then | |  |  |  |  |
| jf(h(x1); : : : ; h(xm)) : h 2 Hgj | e | m | |  | d |
|  |  | : |
|  | d | |

Denote A = f(1[h(x1)6=y1]; : : : ; 1[h(xm)6=ym]) : h 2 Hg. This clearly implies that

e m d

:

d

|  |  |
| --- | --- |
| 28.2 The Lower Bound for the Agnostic Case | 393 |
|  |  |

Combining this with Lemma [26.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page380) we obtain the following bound on the Rademacher complexity:

r

|  |  |  |  |
| --- | --- | --- | --- |
| R(A) | 2d log(em=d) | | : |
|  |  |
|  | m |

Using Theorem [26.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page378) we obtain that with probability of at least 1 , for every h 2 H we have that

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LD(h) LS(h) r |  |  | + r |  |  |
| 8d | m | m | : |
|  |  | log(em=d) |  | 2 log(2= ) | |
|  |  |  |  |  |  |

Repeating the previous argument for minus the zero-one loss and applying the union bound we obtain that with probability of at least 1 , for every h 2 H

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| it holds that |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| jLD(h) LS(h)j | r | |  |  | |  |  |  | + r |  |  |  |  |  |  |
| 8d | | | | m |  |  |  | m | | | |
|  |  |  |  |  |  |  | log(em=d) |  |  |  | 2 log(4= ) | | | | |
|  |  |  | |  |  |  | |  | |  |  |  |  |  |  |
|  | 2 | r | | |  |  | | | | |  | |  |  |  |
|  | 8d log(em=dm | | | | | : | | | |  |
|  |  |  |  |  |  |  |  | ) + 2 log(4= ) | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 4 | | | | (8d log(m) + 8d log(e=d) + 2 log(4= )) : | | | | | | | | |
|  | m | |  |  |
|  | 2 | |
|  | Using Lemma [A.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419), a su cient condition for the inequality to hold is that | | | | | | | | | | | | |
|  |  | 32d | | | 64d | | | 8 | |  |  |  |  |
|  | m 4 |  | | log | |  |  | + |  |  | (8d log(e=d) + 2 log(4= )) : | | |
|  | 2 | | 2 | 2 |
| 28.2 | The Lower Bound for the Agnostic Case | | | | | | | | | | | |  |
|  | Here, we prove that there exists C such that H is agnostic PAC learnable with | | | | | | | | | | | | |
|  | sample complexity | | | |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | mH( ; ) C | | | | | | d + ln(1= ) | | : |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  | 2 |
|  | We will prove the lower bound in two parts. First, we will show that m( ; ) | | | | | | | | | | | | |
|  | 0:5 log(1=(4 ))= 2, and second we will show that for every1=8 we have that | | | | | | | | | | | | |
|  | m( ; ) 8d= 2. These two bounds will conclude the proof. | | | | | | | | | | | | |
| 28.2.1 | Showing That m( ; ) 0:5 log(1=(4 ))= 2 | | | | | | | | | | | |  |

p

We rst show that for any < 1= 2 and any 2 (0; 1), we have that m( ; ) 0:5 log(1=(4 ))= 2. To do so, we show that for m 0:5 log(1=(4 ))= 2, H is not learnable.

Choose one example that is shattered by H. That is, let c be an example such

1. Proof of the Fundamental Theorem of Learning Theory

that there are h+; h 2 H for which h+(c) = 1 and h (c) = 1. De ne two distributions, D+ and D , such that for b 2 f 1g we have

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Db(f(x; y)g) = ( | 1+yb | | | if x = c |
|  |  | 2 |
| 0 | | otherwise: |

That is, all the distribution mass is concentrated on two examples (c; 1) and

|  |  |  |  |
| --- | --- | --- | --- |
| (c; 1), where the probability of (c; b) is | | 1+b | and the probability of (c; b) is |
| 2 |
| 1 2b | . |  |  |

Let A be an arbitrary algorithm. Any training set sampled from Db has the form S = (c; y1); : : : ; (c; ym). Therefore, it is fully characterized by the vector

1. = (y1; : : : ; ym) 2 f 1gm. Upon receiving a training set S, the algorithm A returns a hypothesis h : X ! f 1g. Since the error of A w.r.t. Db only depends on h(c), we can think of A as a mapping from f 1gm into f 1g. Therefore, we denote by A(y) the value in f 1g corresponding to the prediction of h(c), where h is the hypothesis that A outputs upon receiving the training set S = (c; y1); : : : ; (c; ym).

Note that for any hypothesis h we have

1 h(c)b

LDb (h) = 2 :

In particular, the Bayes optimal hypothesis is hb and

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Db |  | Db | b |  |  |  | 2 |  |  | 2 |  | (0 | otherwise: |
| L |  | (A(y)) | L | (h | ) = | 1 |  | A(y)b |  | 1 |  | = |  | if A(y) 6= b |
|  |  |  |  |  |  |  |  |

Fix A. For b 2 f 1g, let Y b = fy 2 f0; 1gm : A(y) 6= bg. The distribution Db induces a probability Pb over f 1gm. Hence,

X

P [LDb (A(y)) LDb (hb) = ] = Db(Y b) = Pb[y]1[A(y)6=b]:

y

Denote N+ = fy : jfi : yi = 1gj m=2g and N = f 1gm n N+. Note that for any y 2 N+ we have P+[y] P [y] and for any y 2 N we have P [y] P+[y].

|  |  |
| --- | --- |
| 28.2 The Lower Bound for the Agnostic Case | 395 |
|  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Therefore, | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | b | max | | | | | P | | | [L | |  | (A(y)) | | | |  | | L | Db | | | (h | | ) = ] | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 1 |  | Db | |  |  |  |  |  |  | b |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 2f g | | | | | | |  |  | X | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 2f g | | | | | | Pb[y]1 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | = | | b | | max | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | 1 |  | y |  |  |  | [A(y)6=b] | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | | | | | | | | | |  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  |  |  | 1 | | | X | | | |  |  |  |  |  |  |  |  |  |  |  | 1 | | | X | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | |  |  | P+[y]1[A(y)6=+] + | | | | | | | | | | | |  |  | P [y]1[A(y)6= ] | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 2 | |  | y | |  | 2 | | y | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | = 1 | | | | |  |  |  |  |  | (P [y]1 | | | | | |  |  |  |  | + P [y]1 | | | | | | | | |  |  |  |  |  |  |  |  |  | ) + | | | | | 1 | | |  |  | (P [y]1 | | |  | + P [y]1 | | | | |  |  | ) |
|  |  |  | 2 | | |  | X | | | | | + | |  |  |  |  |  | 6 |  |  |  |  |  |  | |  |  |  |  |  |  | 6 | | | | |  |  |  |  |  |  | 2 | | |  | X | | + |  | 6 |  |  |  |  |  | 6 |  |  |
|  |  |  |  |  |  | y2N+ | | | | | |  | [A(y)=+] | | | | | |  |  |  |  |  | [A(y)= | | | | | | | | |  | ] |  |  |  |  |  |  |  |  |  | y2N | | [A(y)=+] | |  |  |  | [A(y)= | | ] |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 1 | | |  | X | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | | |  | X | |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | |  | |  |  |  |  |  | (P [y]1[A(y)6=+] + P [y]1[A(y)6= ]) + | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  | (P+[y]1[A(y)6=+] + P+[y]1[A(y)6= ]) | | | | | | | | | | | |
|  | 2 | |  |  |  |  |  | 2 | |  |
|  |  |  |  |  |  | y2N+ | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | y2N | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | 1 | | |  | X | | | | |  |  |  | 1 | | |  | X | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | | | | | | |  | | | | | | |  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  | = | | |  | |  |  |  |  |  | P [y] + | | | | |  | |  |  |  |  |  | P+[y] : | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 2 | |  |  |  |  |  | 2 | | y2N | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | y2N+ | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | Next note that | | | | | | | | | | | | | | y2N+ P [y] = | | | | | | | | | | | | |  |  | y2N P+[y], and both values are the prob- | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |
|  | ability that a | | | | | | | | | | | | | Binomial (m; (1 | | | | | | | | | | | |  | )=2) random variable will have value greater | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |
|  |  | P | |  |  |  |  |  |  |  |  |  |  | | P | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | than m=2. Using Lemma [B.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page428), this probability is lower bounded by | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |
|  | | | | | | | 1 |  | | | | | | |  | | | | | | | | | | | | | | | | | |  | | | | | | 1 | | | | |  | | | | | |  | | | |  | | | | | | |
|  |  |  |  |  |  |  |  |  | |  |  | p | | 1 exp( m 2=(1 2)) | | | | | | | | | | | | | | | | | | | | | |  |  |  |  | | |  | 1 exp( 2m 2) ; | | | | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | 1 |  |  | | | | |  |  |  | 1 |  |  |  |  |  |
|  |  |  |  |  |  |  | 2 |  |  | 2 | |  |  |  |  |  |  |
|  | | | | | | | | | | | | | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | p | | | | | | | | | 2 | | |
|  | where we used the assumption that | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  | 1=2. It follows that if m 0:5 log(1=(4 ))= | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |
|  | then there exists b such that | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | P [LDb (A(y)) LDb (hb) = ] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 1 q | | | | | | | | | | |  |  | | | |  |  | |  | | |  | | | ; |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 p4 | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | |  | | | | | | | | | | | | | | |  | | | |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | | | | | | | | | |  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  |  | | | | | | | | | | | | | | | | | | | | | | | | |  |  | | |  | |  | |  |  |  | | | |  |  | |  | | |  | | | | | | | |  |  | | |  |  |  |
|  | where the last inequality follows by standard algebraic manipulations. This con- | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |
|  | cludes our proof. | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 28.2.2 | Showing That m( ; 1=8) 8d= 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  | p | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 8d | |  |  |  |  |  |
|  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | | | | | | | | | | | | | |  | | | | |
|  | We shall now prove that for every < 1=(8 2) we have that m( ; ) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | . | |  |  |  |  |
|  | 2 |  |  |  |  |
| p | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  | |  | | | | | | | | | | | | | | | | | | | | | | | | |
|  |  | Let = 8 and note that 2 (0; 1= 2). We will construct a family of distri- | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |
|  | butions as follows. First, let C = fc1; : : : ; cdg be a set of d instances which are | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |
|  | shattered by H. Second, for each vector (b1; : : : ; bd) 2 f 1gd, de ne a distribu- | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |  |
|  | tion Db such that | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Db(f(x; y)g) = ( | | | | | | | | | | | | | | | 1 |  | 1+ybi | | | | | | | | |  | | |  | if 9i : x = ci | | | | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | d | 2 | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |

1. otherwise:

That is, to sample an example according to Db, we rst sample an element ci 2 C uniformly at random, and then set the label to be bi with probability (1 + )=2 or bi with probability (1 )=2.

It is easy to verify that the Bayes optimal predictor for Db is the hypothesis

1. Proof of the Fundamental Theorem of Learning Theory

h 2 H such that h(ci) = bi for all i 2 [d], and its error is 1 2 . In addition, for any other function f : X ! f 1g, it is easy to verify that

LDb (f) =

Therefore,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 + |  | jfi 2 [d] : f(ci) 6= bigj | + | 1 |  |  | jfi 2 [d] : f(ci) = bigj | : |
| 2 | d | 2 | d |
|  |  |  |

jfi 2 [d] : f(ci) 6= bigj

LDb (f) min LDb (h) = d : (28.2)

h2H

Next, x some learning algorithm A. As in the proof of the No-Free-Lunch theorem, we have that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Db:b2f 1gd S Dbm | | | | |  |  | D |  |  |  | h2H | |  | D |  |  |  |  |  |  |
| max | | | | E |  | L |  | b (A(S)) | | |  | min L | |  | b | (h) |  |  |  |  |
| Db | | | | :b U(f 1gd) | | S Dbm | | |  | D |  |  | h2H | | | | D |  |  |
|  |  |  |  | E |  |  | E | | L | b (A(S)) | | |  |  | min L | |  | b | (h) |  |
| Db | | | | :b U(f 1gd) | | S Dbm | | |  |  |  |  |  |  |  | d |  |  |  |
| = |  |  |  | E |  |  | E | |  |  |  | jfi 2 [d] : A(S)(ci) 6= bij | | | | | | | |  |
|  |  |  |  |  | | | | | | | |  |
|  |  |  | Xi D | | f g | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| = |  |  |  |  | E | |  |  | E | |  | 1[A(S)(ci)=bi]; | | | | |  |  |  |  |
|  | d | |  | =1 b:b U( 1 d) S Dbm | | | | | | | | |  | 6 | | |  |  |  |  |

(28.3)

(28.4)

(28.5)

(28.6)

where the rst equality follows from Equation ([28.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page396)). In addition, using the de nition of Db, to sample S Db we can rst sample (j1; : : : ; jm) U([d])m, set xr = cji , and nally sample yr such that P[yr = bji ] = (1 + )=2. Let us simplify the notation and use y b to denote sampling according to P[y = b] = (1 + )=2. Therefore, the right-hand side of Equation ([28.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page396)) equals

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | d |  |  |  |  |  |  |  |
|  |  | Xi |  | E |  | f g | E | 1[A(S)(ci)=bi]: | (28.7) |
|  |  | =1 j |  |  | E |
| d | |  | U([d])m b | U( | 1 | d) 8r;yr bjr | 6 |  |

We now proceed in two steps. First, we show that among all learning algorithms, A, the one which minimizes Equation ([28.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page396)) (and hence also Equation ([28.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page396))) is the Maximum-Likelihood learning rule, denoted AML. Formally, for each i, AML(S)(ci) is the majority vote among the set fyr : r 2 [m]; xr = cig. Second, we lower bound Equation ([28.7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page396)) for AML.

lemma 28.1 Among all algorithms, Equation ([28.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page396)) is minimized for A being the Maximum-Likelihood algorithm, AML, de ned as

|  |  |  |
| --- | --- | --- |
| 8i; AML(S)(ci) = sign | x =c | yr! : |
| r: | Xr | i |

Proof Fix some j 2 [d]m. Note that given j and y 2 f 1gm, the training set

1. is fully determined. Therefore, we can write A(j; y) instead of A(S). Let us also x i 2 [d]. Denote b:i the sequence (b1; : : : ; bi 1; bi+1; : : : ; bm). Also, for any

|  |  |
| --- | --- |
| 28.2 The Lower Bound for the Agnostic Case | 397 |
|  |  |

y 2 f 1gm, let yI denote the elements of y corresponding to indices for which jr = i and let y:I be the rest of the elements of y. We have

1. E1[A(S)(ci)6=bi]

b U(f 1gd) 8r;yr bjr

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | | X |  |  |  |  |  |  | X | | |  |  |  |  |  |  |
| = |  |  |  |  |  |  | U(E | |  | P [yjb:i; bi]1[A(j;y)(ci)6=bi] | | | | | | |  |  |
| 2 |  |  | b:i |  | 1 d 1) |  |  |
|  |  | bi2f 1g | | |  |  | f g | | y | | |  | 0 |  |  | 1 |  |
|  |  |  |  |  |  | P [y:I jb:i] 2 | | | |  |  | P [yI jbi]1[A(j;y)(ci)6=bi] |  |
| = b:i | | | U(E | 1 d 1) | | | |  |  |  | : |
|  |  |  |  |  |  |  |  | X |  | 1 | | | X @ | | X |  | A |  |
|  |  |  | f g | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | y:I |  |  |  |  | yI | bi | 1 | g |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2f |  |  |

The sum within the parentheses is minimized when A(j; y)(ci) is the maximizer of P [yI jbi] over bi 2 f 1g, which is exactly the Maximum-Likelihood rule. Re-peating the same argument for all i we conclude our proof. 

Fix i. For every j, let ni(j) = fjt : jt = ijg be the number of instances in which the instance is ci. For the Maximum-Likelihood rule, we have that the quantity

1. E 1[AML(S)(ci)6=bi] b U(f 1gd) 8r;yr bjr

is exactly the probability that a binomial (ni(j); (1 )=2) random variable will be larger than ni(j)=2. Using Lemma [B.11](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page428), and the assumption 2 1=2, we have that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  | 1 | |  |  |  |  |  |  |  |  |  |
|  | P [B ni(j)=2] | | | | | | | | | 1 p1 e 2ni(j) 2 : | | | | | | | | |
|  |  |  |
| 2 | |
| We have thus shown that | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | d |  | |  |  |  |  | f g | | |  |  |  |  |  |  |  |  |
|  |  |  | Xi |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | E |  |  | E | |  |  | E1[A(S)(ci)=bi] | | | | | | |
| d | | | =1 j U([d])m b U( | | | | | | |  |  | 1 d) 8r;yr bjr | | | | | 6 | | | |
|  |  |  |  |  | |  | d |  |  |  |  |  |  |  |  |  |  |  |  | |
|  |  |  |  |  | i=1 | j U([d])m | | | |  | | | | 2 | |
|  |  |  | 2d | |  |  |  |
|  |  |  |  |  |  | X | | | E |  |  | 1 |  | p | 1 e 2 ni(j) | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  | d |  |  |  |  |  |  |  |  |  |  |  | | |
|  |  |  |  |  | i=1 | j U([d])m | | | |  | p |  |  |  |
|  |  |  | 2d | |  |  |  |  |
|  |  |  |  |  |  | X | | | E |  |  | 1 |  | 2 2ni(j) ; | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

where in the last inequality we used the inequality 1 e a a.

Since the square root function is concave, we can apply Jensen's inequality to obtain that the above is lower bounded by

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2d | | | | i=1 | |  |  | r | | | | | j U([d])m | | ! | |
|  |  | | |  | d |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | X | |  | 1 | |  |  | 2 2 | |  | E | ni(j) | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | d |  | 1 | |  | 2 2m=d | | | | |  |  |
| = 2d i=1 | | | | | |  |  |  |
|  |  |  |  | X | |  |  |  | p |  | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| = |  |  | | | 1 | p | | 2 2m=d : | | | | | | |  |  |
| 2 |  |  |

1. Proof of the Fundamental Theorem of Learning Theory

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| As long as m < |  | d |  | , this term would be larger than =4. | | | | | | | | | | |
| 8 | | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| In summary, we have shown that if m < | | | | | | | | |  | d | | then for any algorithm there | | |
|  | 2 | |
|  |  |  |  |  |  |  |  |  | 8 | | | |  |  |
| exists a distribution such that | | | | | | |  | h2H |  |  |  |  |  |  |
|  |  |  |  | S Dm | L | D | (A(S)) |  | D | (h) | | =4: |
|  |  |  |  | E |  | min L | |  |  |

Finally, Let = 1 (LD(A(S)) minh2H LD(h)) and note that 2 [0; 1] (see Equation ([28.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page396))). Therefore, using Lemma [B.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page422), we get that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P[LD(A(S)) h2H | D |  |  |  | P | | |  |  |  |  |  | | E |  |  | | |
| min L |  | (h) > ] = | | |  |  |  |  | > | | |  |  |  | [ ] |  |  | |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 1 | | |  |  |  | | | |  |  |  |  |  |
|  |  |  |  |  |  | |  |  | : | | |  |  |  |  |  |
|  |  |  | 4 | | |  |  |  |  |  |  |
| Choosing = 8 we conclude that if m < | | |  | d |  |  |  | | , then with probability of at least | | | | | | | | | |
| 512 | | | | 2 | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1=8 we will have LD(A(S)) minh2H LD(h) .

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 28.3 | The Upper Bound for the Realizable Case | | | |  |  |
|  | Here we prove that there exists C such that H is PAC learnable with sample | | | | | |
|  | complexity |  |  |  |  |  |
|  | mH( ; ) C | d ln(1= ) + ln(1= ) | | | | |
|  |  |  |  |  | : |
|  |  |  |  |  |
|  | We do so by showing that for m |  | C | d ln(1= )+ln(1= ) | , H is learnable using the | |
|  |  |  |

ERM rule. We prove this claim based on the notion of -nets.

definition 28.2 ( -net) Let X be a domain. S X is an -net for H 2X with respect to a distribution D over X if

8h 2 H : D(h) ) h \ S 6= ;:

theorem 28.3 Let H 2X with VCdim(H) = d. Fix 2 (0; 1), 2 (0; 1=4)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| and let | |  | 2d log |  | | + log |  | | | : |
| m | |
| 8 | | |  | 16e | |  | 2 | | |  |
|  |  |  |  |  |  |  |  |  |  |  |

Then, with probability of at least 1 over a choice of S Dm we have that S is an -net for H.

Proof Let

B = fS X : jSj = m; 9h 2 H; D(h) ; h \ S = ;g

be the set of sets which are not -nets. We need to bound P[S 2 B]. De ne

B0 = f(S; T ) X : jSj = jT j = m; 9h 2 H; D(h) ; h \ S = ;; jT \ hj > 2m g:

|  |  |
| --- | --- |
| 28.3 The Upper Bound for the Realizable Case | 399 |
|  |  |

Claim 1

P[S 2 B] 2 P[(S; T ) 2 B0].

Proof of Claim 1 : Since S and T are chosen independently we can write

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P |  | 2 | B0] = | (S;T ) D2m |  | 2 |  | 0 |  | | S Dm | hT Dm |  | 2 | 0 |  | i |
|  | [(S; T ) |  | E | 1[(S;T ) |  | B | | ] | = | E | E | 1[(S;T ) |  | B | ] | : |
| Note that (S; T ) 2 B0 implies S 2 B and therefore 1[(S;T )2B0] | | | | | | | | | | | | | = 1[(S;T )2B0] 1[S2B], | | | | |
| which gives | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | P[(S; T ) 2 B0] = S E m T | | | | | E | | m 1[(S;T )2B0] 1[S2B] | | | |  |  |  |  |
|  |  |  |  |  | D |  |  | D | |  |  |  |  |  |  |  |  |

1. E 1[S2B] E 1[(S;T )2B0]:

S DmT Dm

Fix some S. Then, either 1[S2B] = 0 or S 2 B and then 9hS such that D(hS) and jhS \ Sj = 0. It follows that a su cient condition for (S; T ) 2 B0 is that jT \ hSj > 2m . Therefore, whenever S 2 B we have

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | E | m 1[(S;T )2B0] |  | P | m[jT \ hSj > | m | ]: |
| T | T | 2 |
|  | D |  |  | D |  |  |  |

But, since we now assume S 2 B we know that D(hS) = . Therefore, jT \ hSj is a binomial random variable with parameters (probability of success for a single try) and m (number of tries). Cherno 's inequality implies

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 2 | |  | 2 |  |  |  |  |  |  |  |
| [ T h |  | m | ] |  | e | m | (m m =2) | = e m =2 |  | e m =2 |  | e d log(1= )=2 | = d=2 |  | 1=2: |
| P j \ | Sj | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |

Thus,

P[jT \ hSj > 2m ] = 1 P[jT \ hSj 2m ] 1 P[jT \ hSj 2m ] 1=2:

Combining all the preceding we conclude the proof of Claim 1.

Claim 2 (Symmetrization):

P[(S; T ) 2 B0] e m=4 H(2m):

Proof of Claim 2 : To simplify notation, let = m =2 and for a sequence A =

(x1; : : : ; x2m) let A0 = (x1; : : : ; xm). Using the de nition of B0 we get that

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | [A | 2 | B0] = | E | max 1 | | | 1 | 1 |
|  | A | 2m h | 2H | [D(h) ] | [jh\A0j=0] | [jh\Aj ] |
|  |  |  |  | D |  |  |  |  |

E max 1[jh\A0j=0] 1[jh\Aj ]: A D2m h2H

Now, let us de ne by HA the e ective number of di erent hypotheses on A, namely, HA = fh \ A : h 2 H g. It follows that

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | [A | 2 | B0] | A | E | max 1 | | | | 1 |
|  |  | 2m h | 2H | A | [jh\A0j=0] | [jh\Aj ] |
|  |  |  |  |  | D |  |  |  |  |
|  |  |  |  | A | E | X | | | 1[jh\A0j=0] | 1[jh\Aj ]: |
|  |  |  |  | 2m |  |  |
|  |  |  |  |  | D | h2HA | | |  |  |
|  |  |  |  |  |  |  |  |

Let J = fj [2m] : jjj = mg. For any j 2 J and A = (x1; : : : ; x2m) de ne

Aj = (xj1 ; : : : ; xjm ). Since the elements of A are chosen i.i.d., we have that for any j 2 J and any function f(A; A0) it holds that EA D2m [f(A; A0)] =

1. Proof of the Fundamental Theorem of Learning Theory

EA D2m [f(A; Aj)]. Since this holds for any j it also holds for the expectation of j chosen at random from J. In particular, it holds for the function f(A; A0) =

P

h2HA 1[jh\A0j=0] 1[jh\Aj ]. We therefore obtain that

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P[A 2 B0] A | E |  | EJ | X |
| 2m j | 1[jh\Aj j=0] 1[jh\Aj ] |
|  | D |  |  | h2HA |
|  |  |  |  |

X

1. E1[jh\Aj ] E 1[jh\Aj j=0]:

A D2mj J

h2HA

Now, x some A s.t. jh \ Aj . Then, Ej 1[jh\Aj j=0] is the probability that when choosing m balls from a bag with at least red balls, we will never choose a red ball. This probability is at most

(1 =(2m))m = (1 =4)m e m=4:

We therefore get that

X

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P[A 2 B0] A | E | 2m | e m=4 e m=4 A | E | 2m jHAj: |
|  | D |  | h2HA | D |  |
|  |  |  |  |  |

Using the de nition of the growth function we conclude the proof of Claim 2.

Completing the Proof: By Sauer's lemma we know that H(2m) (2em=d)d.

Combining this with the two claims we obtain that

P[S 2 B] 2(2em=d)d e m=4:

We would like the right-hand side of the inequality to be at most ; that is,

2(2em=d)d e m=4 :

Rearranging, we obtain the requirement

m 4 (d log(2em=d) + log(2= )) = 4d log(m) + 4 (d log(2e=d) + log(2= ):

Using Lemma [A.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419), a su cient condition for the preceding to hold is that

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | | |  |  | log | | | | | | | |  | + (d log(2e=d) + log(2= ): | | | | | | | | | | | | |
|  |  |  | 16d | | |  | 8d | | | | | |  |  |  | 8 | | | |  |  |  |  |  |  |  |
|  | |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| A su cient condition for this is that | | | | | | | | | | | | | | | |  | (d log(2e=d) + | | | | | | | | 21 log(2= ) | |
| m |  |  |  | log | |  |  | | | | | + | | | |  |
|  | 16d | | |  |  |  | 8d | | | | |  |  |  |  | 16 | | |  |  |  |  |  |  |  |  |
|  |  |  |  | log | |  |  | |  |  |  |  | |  |  |  |  | |  | |  |  |  |  |  |  |
| = |  |  |  |  | | | | | | d | |  |  | + log(2= ) | | | | | | | | |  |  |
|  | 16d | | |  |  |  |  |  | 8d 2e | | | | | | | 8 | | | | | |  |  |  |  |  |
|  | |  |  |  | |  |  | |  |  | |  | |  |  | | | | | |  |  |  |  |  |  |
| = | | |  | 2d log | |  |  | | | | |  | | + log: | | | | | | | | | | |  |  |
|  | 8 | |  |  |  |  |  |  |  | 16e | | | |  |  | 2 | | | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

and this concludes our proof.

|  |  |
| --- | --- |
| 28.3 The Upper Bound for the Realizable Case | 401 |
|  |  |

28.3.1 From -Nets to PAC Learnability

theorem 28.4 Let H be a hypothesis class over X with VCdim(H) = d. Let D be a distribution over X and let c 2 H be a target hypothesis. Fix ; 2 (0; 1) and let m be as de ned in Theorem [28.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page398). Then, with probability of at least 1 over a choice of m i.i.d. instances from X with labels according to c we have that any ERM hypothesis has a true error of at most .

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Proof De ne the class Hc = fc | h : h 2 Hg, where c | | | h = (h nc) [(c nh). It is | | |
| is shattered by | | H | then it is also shattered by | | H | c |
| easy to verify that if some A Xa |  | a |  |  |

and vice versa. Hence, VCdim(H) = VCdim(Hc). Therefore, using Theorem [28.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page398)

we know that with probability of at least 1 , the sample S is an -net for Hc. a

Note thata LD(h) = D(h c). Therefore, for any h 2 H with LD(h) we have that j(h c)\Sj > 0, which implies that h cannot be an ERM hypothesis, which concludes our proof. 

**Python in Machine Learning**



Python has libraries that enables developers to use optimized algorithms. It implements popular machine learning techniques such as recommendation, classification, and clustering. Therefore, it is necessary to have a brief introduction to machine learning before we move further.

**What is Machine Learning?**



Data science, machine learning and artificial intelligence are some of the top trending topics in the tech world today. Data mining and Bayesian analysis are trending and this is adding the demand for machine learning. This tutorial is your entry into the world of machine learning.

Machine learning is a discipline that deals with programming the systems so as to make them automatically learn and improve with experience. Here, learning implies recognizing and understanding the input data and taking informed decisions based on the supplied data. It is very difficult to consider all the decisions based on all possible inputs. To solve this problem, algorithms are developed that build knowledge from a specific data and past experience by applying the principles of statistical science, probability, logic, mathematical optimization, reinforcement learning, and control theory.

**Applications of Machine Learning Algorithms**



The developed machine learning algorithms are used in various applications such as:

1. Vision processing
2. Language processing
3. Forecasting things like stock market trends, weather
4. Pattern recognition
5. Games
6. Data mining
7. Expert systems
8. Robotics

**Steps Involved in Machine Learning**



A machine learning project involves the following steps:

* Defining a Problem
* Preparing Data
* Evaluating Algorithms
* Improving Results
* Presenting Results

The best way to get started using Python for machine learning is to work through a project end-to-end and cover the key steps like loading data, summarizing data, evaluating algorithms and making some predictions. This gives you a replicable method that can be used dataset after dataset. You can also add further data and improve the results.

**3. Python Machine Learning –**

In this chapter, you will learn how to setup the working environment for Python machine learning on your local computer.

**Libraries and Packages**



To understand machine learning, you need to have basic knowledge of Python programming. In addition, there are a number of libraries and packages generally used in performing various machine learning tasks as listed below:

* **numpy** - is used for its N-dimensional array objects
* **pandas** –is a data analysis library that includes dataframes
* **matplotlib** –is 2D plotting library for creating graphs and plots
* **scikit-learn** - the algorithms used for data analysis and data mining tasks
* **seaborn** –a data visualization library based on matplotlib

**Installation**



You can install software for machine learning in any of the two methods as discussed here:

**Method 1**

Download and install Python separately from **python.org** on various operating systems as explained below:

To install Python after downloading, double click the **.exe** (for Windows) or .**pkg** (for Mac) file and follow the instructions on the screen.

For Linux OS, check if Python is already installed by using the following command at the prompt:



$ python --version. ...

If Python 2.7 or later is not installed, install Python with the distribution's package manager. Note that the command and package name varies.

On Debian derivatives such as Ubuntu, you can use **apt**:



$ sudo apt-get install python3

Now, open the command prompt and run the following command to verify that Python is installed correctly:

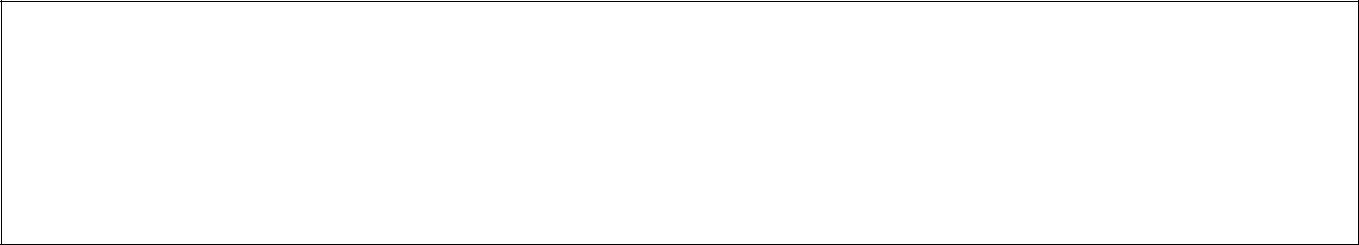


$ python3 --version



Python 3.6.2

Similarly, we can download and install necessary libraries like numpy, matplotlib etc. individually using installers like **pip**. For this purpose, you can use the commands shown here:



$pip install numpy

$pip install matplotlib

$pip install pandas

$pip install seaborn

**Method 2**

Alternatively, to install Python and other scientific computing and machine learning packages simultaneously, we should install **Anaconda** distribution. It is a Python implementation for Linux, Windows and OSX, and comprises various machine learning packages like numpy, scikit-learn, and matplotlib. It also includes **Jupyter Notebook**, an interactive Python environment. We can install Python 2.7 or any 3.x version as per our requirement.

To download the free Anaconda Python distribution from Continuum Analytics, you can do the following:

Visit the official site of Continuum Analytics and its download page. Note that the installation process may take 15-20 minutes as the installer contains Python, associated packages, a code editor, and some other files. Depending on your operating system, choose the installation process as explained here:

**For Windows:** Select the **Anaconda for Windows** section and look in the column withPython 2.7 or 3.x. You can find that there are two versions of the installer, one for 32-bit Windows, and one for 64-bit Windows. Choose the relevant one.

**For Mac OS:** Scroll to the **Anaconda for OS X** section. Look in the column with Python

2.7 or 3.x. Note that here there is only one version of the installer: the 64-bit version.

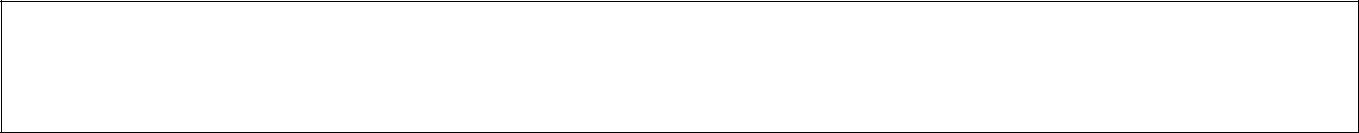
**For Linux OS:** We select the "Anaconda for Linux" section. Look in the column with Python2.7 or 3.x.

Note that you have to ensure that Anaconda’s Python distribution installs into a single directory, and does not affect other Python installations, if any, on your system.

To work with graphs and plots, we will need these Python library packages: **matplotlib** and **seaborn**.

If you are using Anaconda Python, your system already has numpy, matplotlib, pandas, seaborn, etc. installed. We start the Anaconda Navigator to access either Jupyter Note book or Spyder IDE of python.

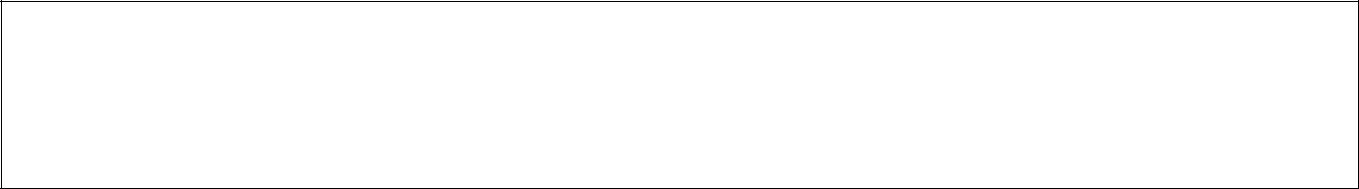
After opening either of them, type the following commands:



import numpy

import matplotlib

Now, we need to check if installation is successful. For this, go to the command line and type in the following command:

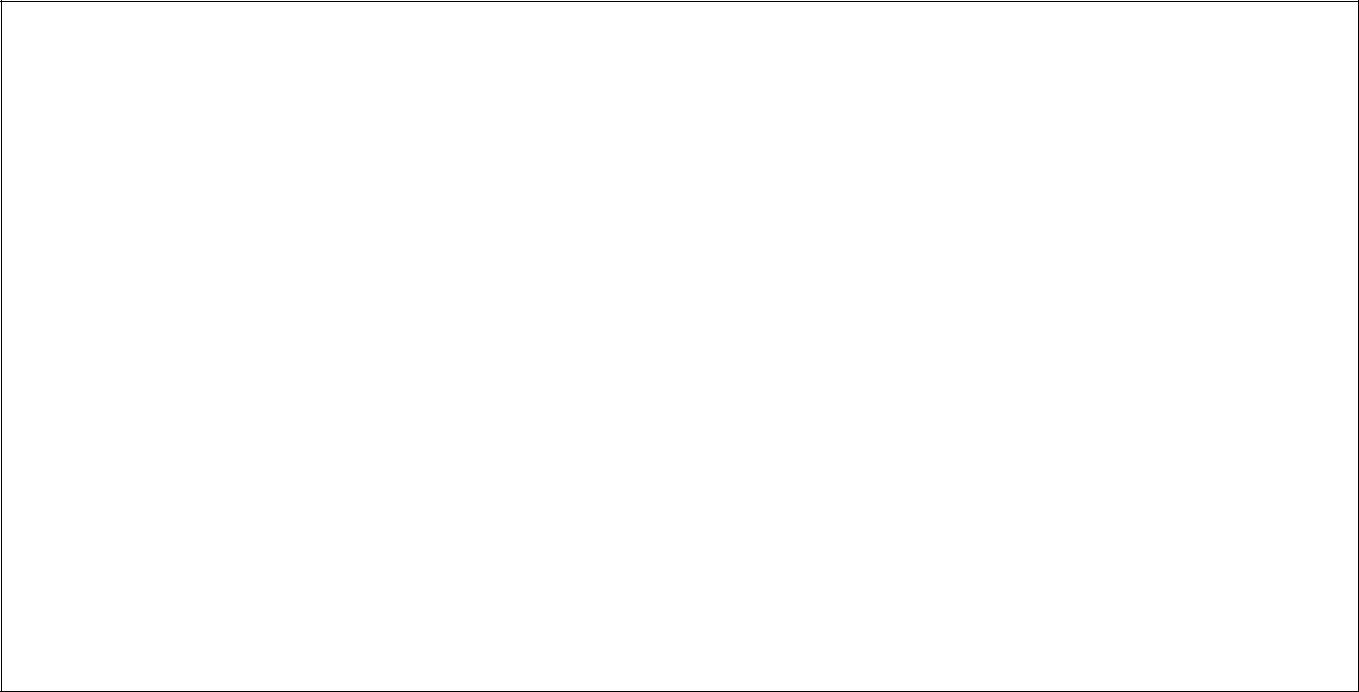


$ python

Python 3.6.3 |Anaconda custom (32-bit)| (default, Oct 13 2017, 14:21:34)

[GCC 7.2.0] on linux

Next, you can import the required libraries and print their versions as shown:



>>>import numpy

>>>print numpy.\_\_version\_\_

1.14.2

* import matplotlib
* print (matplotlib.\_\_version\_\_) 2.1.2

>> import pandas

* print (pandas.\_\_version\_\_)

0.22.0

1. import seaborn
2. print (seaborn.\_\_version\_\_) 0.8.1

**4. Python Machine**

1. Python For Machine Learning

**Machine Learning (ML)** is an automated learning with little or no human intervention.It involves programming computers so that they learn from the available inputs. The main purpose of machine learning is to explore and construct algorithms that can learn from the previous data and make predictions on new input data.

The **input** to a learning algorithm is training data, representing experience, and the **output** is any expertise, which usually takes the form of another algorithm that canperform a task. The input data to a machine learning system can be numerical, textual, audio, visual, or multimedia. The corresponding output data of the system can be a floating-point number, for instance, the velocity of a rocket, an integer representing a category or a class, for example, a pigeon or a sunflower from image recognition.

In this chapter, we will learn about the training data our programs will access and how learning process is automated and how the success and performance of such machine learning algorithms is evaluated.

**Concepts of Learning**



Learning is the process of converting experience into expertise or knowledge.

Learning can be broadly classified into three categories, as mentioned below, based on the nature of the learning data and interaction between the learner and the environment.

1. Supervised Learning
2. Unsupervised Learning
3. Semi-supervised learning

Similarly, there are four categories of machine learning algorithms as shown below:

1. Supervised learning algorithm
2. Unsupervised learning algorithm
3. Semi-supervised learning algorithm
4. Reinforcement learning algorithm

However, the most commonly used ones are **supervised** and **unsupervised learning**.

**Supervised Learning**



Supervised learning is commonly used in real world applications, such as face and speech recognition, products or movie recommendations, and sales forecasting. Supervised learning can be further classified into two types: **Regression** and **Classification**.

**Regression** trains on and predicts a continuous-valued response, for example predictingreal estate prices.

**Classification** attempts to find the appropriate class label, such as analyzingpositive/negative sentiment, male and female persons, benign and malignant tumors, secure and unsecure loans etc.

In supervised learning, learning data comes with description, labels, targets or desired outputs and the objective is to find a general rule that maps inputs to outputs. This kind of learning data is called **labeled data**. The learned rule is then used to label new data with unknown outputs.

Supervised learning involves building a machine learning model that is based on **labeled** **samples**. For example, if we build a system to estimate the price of a plot of land or ahouse based on various features, such as size, location, and so on, we first need to create a database and label it. We need to teach the algorithm what features correspond to what prices. Based on this data, the algorithm will learn how to calculate the price of real estate using the values of the input features.

Supervised learning deals with learning a function from available training data. Here, a learning algorithm analyzes the training data and produces a derived function that can be used for mapping new examples. There are many **supervised learning algorithms** such as Logistic Regression, Neural networks, Support Vector Machines (SVMs), and Naive Bayes classifiers.

Common **examples** of supervised learning include classifying e-mails into spam and not-spam categories, labeling webpages based on their content, and voice recognition.

**Unsupervised Learning**



Unsupervised learning is used to detect anomalies, outliers, such as fraud or defective equipment, or to group customers with similar behaviors for a sales campaign. It is the opposite of supervised learning. There is no labeled data here.

When learning data contains only some indications without any description or labels, it is up to the coder or to the algorithm to find the structure of the underlying data, to discover hidden patterns, or to determine how to describe the data. This kind of learning data is called **unlabeled data**.

Suppose that we have a number of data points, and we want to classify them into several groups. We may not exactly know what the criteria of classification would be. So, an unsupervised learning algorithm tries to classify the given dataset into a certain number of groups in an optimum way.

Unsupervised learning algorithms are extremely powerful tools for analyzing data and for identifying patterns and trends. They are most commonly used for clustering similar input into logical groups. Unsupervised learning algorithms include Kmeans, Random Forests, Hierarchical clustering and so on.

**Semi-supervised Learning**



If some learning samples are labeled, but some other are not labeled, then it is semi-supervised learning. It makes use of a large amount of **unlabeled data for training** and a small amount of **labeled data for testing**. Semi-supervised learning is applied in cases where it is expensive to acquire a fully labeled dataset while more practical to label a small subset. For example, it often requires skilled experts to label certain remote sensing images, and lots of field experiments to locate oil at a particular location, while acquiring unlabeled data is relatively easy.

**Reinforcement Learning**



Here learning data gives feedback so that the system adjusts to dynamic conditions in order to achieve a certain objective. The system evaluates its performance based on the feedback responses and reacts accordingly. The best known instances include self-driving cars and chess master algorithm AlphaGo.

**Purpose of Machine Learning**



Machine learning can be seen as a branch of AI or Artificial Intelligence, since, the ability to change experience into expertise or to detect patterns in complex data is a mark of human or animal intelligence.

As a field of science, machine learning shares common concepts with other disciplines such as statistics, information theory, game theory, and optimization.

As a subfield of information technology, its objective is to program machines so that they will learn.

However, it is to be seen that, the purpose of machine learning is not building an automated duplication of intelligent behavior, but using the power of computers to complement and supplement human intelligence. For example, machine learning programs can scan and process huge databases detecting patterns that are beyond the scope of human perception.

In the real world, we usually come across lots of raw data which is not fit to be readily processed by machine learning algorithms. We need to preprocess the raw data before it is fed into various machine learning algorithms. This chapter discusses various techniques for preprocessing data in Python machine learning.

**Data Preprocessing**



In this section, let us understand how we preprocess data in Python.

Initially, open a file with a **.py** extension, for example **prefoo.py** file, in a text editor like notepad.

Then, add the following piece of code to this file:

import numpy as np

from sklearn import preprocessing

#We imported a couple of packages. Let's create some sample data and add the line to this file:

input\_data = np.array([[3, -1.5, 3, -6.4], [0, 3, -1.3, 4.1], [1, 2.3, -2.9, - 4.3]])

We are now ready to operate on this data.

**Preprocessing Techniques**



Data can be preprocessed using several techniques as discussed here:

**Mean removal**

It involves removing the mean from each feature so that it is centered on zero. Mean removal helps in removing any bias from the features.

You can use the following code for mean removal:

data\_standardized = preprocessing.scale(input\_data)

print "\nMean =", data\_standardized.mean(axis=0)

print "Std deviation =", data\_standardized.std(axis=0)

Now run the following command on the terminal:

$ python prefoo.py

You can observe the following output:

Mean = [ 5.55111512e-17 -3.70074342e-17 0.00000000e+00 -1.85037171e-17]

Std deviation = [1. 1. 1. 1.]

Observe that in the output, mean is almost 0 and the standard deviation is 1.

**Scaling**

The values of every feature in a data point can vary between random values. So, it is important to scale them so that this matches specified rules.

You can use the following code for scaling:

data\_scaler = preprocessing.MinMaxScaler(feature\_range=(0, 1))

data\_scaled = data\_scaler.fit\_transform(input\_data)

print "\nMin max scaled data =", data\_scaled

Now run the code and you can observe the following output:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Min max scaled data = [[ 1. | | | | 0. | 1. | 0. | ] |
| [ | 0. | 1. | 0.27118644 | 1. | ] |  |  |
| [ | 0.33333333 | 0.84444444 | 0. | 0.2 | ]] |  |  |

Note that all the values have been scaled between the given range.

**Normalization**

Normalization involves adjusting the values in the feature vector so as to measure them on a common scale. Here, the values of a feature vector are adjusted so that they sum up to 1. We add the following lines to the prefoo.py file:

You can use the following code for normalization:

data\_normalized = preprocessing.normalize(input\_data, norm='l1')

print "\nL1 normalized data =", data\_normalized

Now run the code and you can observe the following output:

L1 normalized data = [[ 0.21582734 -0.10791367 0.21582734 -0.46043165]

[ 0. 0.35714286 -0.1547619 0.48809524] [ 0.0952381 0.21904762 -0.27619048 -0.40952381]]

Normalization is used to ensure that data points do not get boosted due to the nature of their features.

**Binarization**

Binarization is used to convert a numerical feature vector into a Boolean vector. You can use the following code for binarization:

data\_binarized = preprocessing.Binarizer(threshold=1.4).transform(input\_data)

print "\nBinarized data =", data\_binarized

Now run the code and you can observe the following output:

Binarized data = [[ 1. 0. 1. 0.]

[ 0. 1. 0. 1.]

[ 0. 1. 0. 0.]]

This technique is helpful when we have prior knowledge of the data.

1. Multiclass Learnability

In Chapter [17](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page227) we have introduced the problem of multiclass categorization, in which the goal is to learn a predictor h : X ! [k]. In this chapter we address PAC learnability of multiclass predictors with respect to the 0-1 loss. As in Chapter [6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page67), the main goal of this chapter is to:

Characterize which classes of multiclass hypotheses are learnable in the (mul-ticlass) PAC model.

Quantify the sample complexity of such hypothesis classes.

In view of the fundamental theorem of learning theory (Theorem [6.8](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page72)), it is natu-ral to seek a generalization of the VC dimension to multiclass hypothesis classes. In Section [29.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page402) we show such a generalization, called the Natarajan dimension, and state a generalization of the fundamental theorem based on the Natarajan dimension. Then, we demonstrate how to calculate the Natarajan dimension of several important hypothesis classes.

Recall that the main message of the fundamental theorem of learning theory is that a hypothesis class of binary classi ers is learnable (with respect to the 0-1 loss) if and only if it has the uniform convergence property, and then it is learnable by any ERM learner. In Chapter [13](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page171), Exercise [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page181), we have shown that this equivalence breaks down for a certain convex learning problem. The last section of this chapter is devoted to showing that the equivalence between learnability and uniform convergence breaks down even in multiclass problems with the 0-1 loss, which are very similar to binary classi cation. Indeed, we construct a hypothesis class which is learnable by a speci c ERM learner, but for which other ERM learners might fail and the uniform convergence property does not hold.

29.1 The Natarajan Dimension

In this section we de ne the Natarajan dimension, which is a generalization of the VC dimension to classes of multiclass predictors. Throughout this section, let H be a hypothesis class of multiclass predictors; namely, each h 2 H is a function from X to [k].

|  |  |
| --- | --- |
| 29.2 The Multiclass Fundamental Theorem | 403 |
|  |  |

To de ne the Natarajan dimension, we rst generalize the de nition of shat-tering.

definition 29.1 (Shattering (Multiclass Version)) We say that a set C X is shattered by H if there exist two functions f0; f1 : C ! [k] such that

For every x 2 C, f0(x) 6= f1(x).

For every B C, there exists a function h 2 H such that

8x 2 B; h(x) = f0(x) and 8x 2 C n B; h(x) = f1(x):

definition 29.2 (Natarajan Dimension) The Natarajan dimension of H, de-noted Ndim(H), is the maximal size of a shattered set C X .

It is not hard to see that in the case that there are exactly two classes, Ndim(H) = VCdim(H). Therefore, the Natarajan dimension generalizes the VC dimension. We next show that the Natarajan dimension allows us to general-ize the fundamental theorem of statistical learning from binary classi cation to multiclass classi cation.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 29.2 | The Multiclass Fundamental Theorem | | | | | | | | | | | |  |  |  |  |  |  |  |
|  | theorem 29.3 (The Multiclass Fundamental Theorem) | | | | | | | | | | | | | | | | There exist absolute | | |
|  | constants C1; C2 > 0 such that the following holds. For every hypothesis class H | | | | | | | | | | | | | | | | | | |
|  | of functions from X to [k], such that the Natarajan dimension of H is d, we have | | | | | | | | | | | | | | | | | | |
|  | 1. | H has the uniform convergence property with sample complexity | | | | | | | | | | | | | | | | | |
|  |  |  | d + log(1= ) | | | | | | mHUC( ; ) C2 | | | d log (k) + log(1= ) | | | | | | | |
|  |  | C1 | |  |  |  |  |  |  |  |  |  |  |  |  | : |
|  |  |  |  | 2 | | |  |  |  |  | 2 |  |  |
|  | 2. | H is agnostic PAC learnable with sample complexity | | | | | | | | | | | | | |  |  |  |  |
|  |  |  |  |  | d + log(1= ) | | | | mH( ; ) C2 | | d log (k) + log(1= ) | | | | | | | | |
|  |  | C1 | | |  |  |  | |  | |  |  |  |  |  | : | |
|  |  |  | 2 | | |  | |  |  |  | 2 |  |
|  | 3. | H is PAC learnable (assuming realizability) with sample complexity | | | | | | | | | | | | | | | | | |
|  |  |  | d + | | | log(1= ) | | | d log | | | |  | kd |  | + log(1= ) | | | |
|  |  |  |  |  |
|  |  | C1 |  |  | | mH( ; ) C2 |  | | |  | | |  |  | | : |
|  |  |  | | |  | | |  | | |

29.2.1 On the Proof of Theorem [29.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page403)

The lower bounds in Theorem [29.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page403) can be deduced by a reduction from the binary fundamental theorem (see Exercise [5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page409)).

The upper bounds in Theorem [29.3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page403) can be proved along the same lines of the proof of the fundamental theorem for binary classi cation, given in Chapter [28](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page392) (see Exercise [4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page409)). The sole ingredient of that proof that should be modi ed in a nonstraightforward manner is Sauer's lemma. It applies only to binary classes and therefore must be replaced. An appropriate substitute is Natarajan's lemma:

1. Multiclass Learnability

lemma 29.4 (Natarajan) jHj jX jNdim(H) k2Ndim(H).

The proof of Natarajan's lemma shares the same spirit of the proof of Sauer's lemma and is left as an exercise (see Exercise [3](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page409)).

29.3 Calculating the Natarajan Dimension

In this section we show how to calculate (or estimate) the Natarajan dimen-sion of several popular classes, some of which were studied in Chapter [17](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page227). As these calculations indicate, the Natarajan dimension is often proportional to the number of parameters required to de ne a hypothesis.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 29.3.1 | One-versus-All Based Classes | | | |  |  |  |  |  |  |  |  |  |  |
|  | In Chapter [17](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page227) we have seen two reductions of multiclass categorization to bi- | | | | | | | | | | | | | |
|  | nary classi cation: One-versus-All and All-Pairs. In this section we calculate the | | | | | | | | | | | | | |
|  | Natarajan dimension of the One-versus-All method. | | | | | | |  |  |  |  |  |  |  |
|  |  | Recall that in One-versus-All we train, for each label, a binary classi er that | | | | | | | | | | | | |
|  | distinguishes between that label and the rest of the labels. This naturally sug- | | | | | | | | | | | | | |
|  | gests considering multiclass hypothesis classes of the following form. Let | | | | | | | | | | | | | kHbin |
|  | f | 0; 1 | g | X be a binary hypothesis class. For every h | | | = (h |  | ; : : : ; h ) | 2 | ( | Hbin | ) | de ne |
|  |  |  |  |  |  | 1 | k |  |  |  |
|  | T (h) : X ! [k] by | | | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | |  |  |  |  |  |  |  |  |  |
|  |  |  |  | T (h)(x) = argmax hi(x): | | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  | i2[k] |  |  |  |  |  |  |  |  |
|  | If there are two labels that maximize hi(x), we choose the smaller one. Also, let | | | | | | | | | | | | | |
|  |  |  |  | OvA;k |  |  |  | k | g: |  |  |  |  |  |
|  |  |  |  | Hbin | = fT (h) | : h 2 (Hbin) | |  |  |  |  |  |  |

HOvA;k

What \should" be the Natarajan dimension of bin ? Intuitively, to specify a hypothesis in Hbin we need d = VCdim(Hbin) parameters. To specify a hypothe-sis in HbinOvA;k, we need to specify k hypotheses in Hbin. Therefore, kd parameters should su ce. The following lemma establishes this intuition.

lemma 29.5 If d = VCdim(Hbin) then

Ndim(HbinOvA;k) 3kd log (kd) :

Proof Let C X be a shattered set. By the de nition of shattering (for mul-ticlass hypotheses)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | HbinOvA;k | C |  | 2jCj: |
|  |  |  |  |  |
|  |  |  |  |  |

On the other hand, each hypothesis in HbinOvA;k is determined by using k hypothe-ses from Hbin. Therefore,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | HbinOvA;k | C |  | j (Hbin)C jk: |
|  |  |  |  |  |

|  |  |
| --- | --- |
| 29.3 Calculating the Natarajan Dimension | 405 |
|  |  |

By Sauer's lemma, j (Hbin)C j jCjd. We conclude that

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2jCj |  | HbinOvA;k | C |  | jCjdk: |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

The proof follows by taking the logarithm and applying Lemma [A.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page419).

How tight is Lemma [29.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page404)? It is not hard to see that for some classes, Ndim(HbinOvA;k) can be much smaller than dk (see Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page409)). However there are several natural binary classes, Hbin (e.g., halfspaces), for which Ndim(HbinOvA;k) = (dk) (see Exercise [6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page409)).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 29.3.2 | General Multiclass-to-Binary Reductions | | |  |  |  |  |
|  | The same reasoning used to establish Lemma [29.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page404) can be used to upper bound | | | | | | |
|  | the Natarajan dimension of more general multiclass-to-binary reductions. These | | | | | | |
|  | reductions train several binary classi ers on the data. Then, given a new in- | | | | | | |
|  | stance, they predict its label by using some rule that takes into account the | | | | | | |
|  | labels predicted by the binary classi ers. These reductions include One-versus- | | | | | | |
|  | All and All-Pairs. |  |  |  |  |  |  |
|  | Suppose that such a method trains l binary classi ers from a binary class Hbin, | | | | | | |
|  | and r : f0; 1gl ! [k] is the rule that determines the (multiclass) label according | | | | | | |
|  | to the predictions of the binary classi ers. The hypothesis class corresponding | | | | | | |
|  |  |  |  |  |  | (Hbin) | l |
|  | to this method can be de ned as follows. For every h = (h1; : : : ; hl) 2 | | | | |  |
|  |  |  |  |  |  |  |  |
|  | de ne R(h) : X ! [k] by |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  | R(h)(x) = r(h1(x); : : : ; hl(x)): | | | | |  |  |
|  | Finally, let |  |  |  |  |  |  |
|  | r |  |  | l | g: |  |  |
|  | Hbin | = fR(h) | : h 2 | (Hbin) |  |  |

Similarly to Lemma [29.5](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page404) it can be proven that:

lemma 29.6 If d = VCdim(Hbin) then

Ndim(Hbinr) 3 l d log (l d) :

The proof is left as Exercise [2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page409).

29.3.3 Linear Multiclass Predictors

Next, we consider the class of linear multiclass predictors (see Section [17.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page230)). Let

: X [k] ! Rd be some class-sensitive feature mapping and let

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| H = | (x 7! | i [k] | h | i : w 2 R | d | ) : | (29.1) |
|  | argmax w; (x; i) | | |  |  |
|  |  | 2 |  |  |  |  |  |

Each hypothesis in H is determined by d parameters, namely, a vector w 2 Rd. Therefore, we would expect that the Natarajan dimension would be upper bounded by d. Indeed:

1. Multiclass Learnability

theorem 29.7 Ndim(H ) d .

Proof Let C X be a shattered set, and let f0; f1 : C ! [k] be the two functions that witness the shattering. We need to show that jCj d. For every

def

x 2 C let (x) = (x; f0(x)) (x; f1(x)). We claim that the set (C) = f (x) : x 2 Cg consists of jC j elements (i.e., is one to one) and is shattered by the binary hypothesis class of homogeneous linear separators on Rd,

H = fx 7!sign(hw; xi) : w 2 Rdg:

Since VCdim(H) = d, it will follow that jCj = j (C)j d, as required.

To establish our claim it is enough to show that jH (C)j = 2jCj. Indeed, given a subset B C, by the de nition of shattering, there exists hB 2 H for which

8x 2 B; hB(x) = f0(x) and 8x 2 C n B; hB(x) = f1(x):

Let wB 2 Rd be a vector that de nes hB. We have that, for every x 2 B,

hw; (x; f0(x))i > hw; (x; f1(x))i ) hw; (x)i > 0:

Similarly, for every x 2 C n B,

hw; (x)i < 0:

It follows that the hypothesis gB 2 H de ned by the same w 2 Rd label the points in (B) by 1 and the points in (C n B) by 0. Since this holds for every

1. C we obtain that jCj = j (C)j and jH (C)j = 2jCj, which concludes our

proof.

The theorem is tight in the sense that there are mappings for which Ndim(H ) = (d). For example, this is true for the multivector construction (see Section [17.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page230) and the Bibliographic Remarks at the end of this chapter). We therefore con-clude:

corollary 29.8 Let X = Rn and let : X [k] ! Rnk be the class sensitive feature mapping for the multi-vector construction:

(x; y) = [ 0; : : : ; 0 ; x1; : : : ; xn ; 0; : : : ; 0 ]:

| {z } | {z } | {z }

2R(y 1)n 2Rn 2R(k y)n

Let H be as de ned in Equation ([29.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page405)). Then, the Natarajan dimension of H satis es

(k 1)(n 1) Ndim(H ) kn:

29.4 On Good and Bad ERMs

In this section we present an example of a hypothesis class with the property that not all ERMs for the class are equally successful. Furthermore, if we allow an in nite number of labels, we will also obtain an example of a class that is

jXj 1

6

1

|  |  |
| --- | --- |
| 29.4 On Good and Bad ERMs | 407 |
|  |  |

learnable by some ERM, but other ERMs will fail to learn it. Clearly, this also implies that the class is learnable but it does not have the uniform convergence property. For simplicity, we consider only the realizable case.

The class we consider is de ned as follows. The instance space X will be any nite or countable set. Let Pf (X ) be the collection of all nite and co nite subsets of X (that is, for each A 2 Pf (X ), either A or X n A must be nite). Instead of [k], the label set is Y = Pf (X ) [ f g, where is some special label. For every A 2 Pf (X ) de ne hA : X ! Y by

(

A x 2 A

hA(x) =

x 2= A

Finally, the hypothesis class we take is

H = fhA : A 2 Pf (X )g:

Let A be some ERM algorithm for H. Assume that A operates on a sample labeled by hA 2 H. Since hA is the only hypothesis in H that might return the label A, if A observes the label A, it \knows" that the learned hypothesis is hA, and, as an ERM, must return it (note that in this case the error of the returned hypothesis is 0). Therefore, to specify an ERM, we should only specify the hypothesis it returns upon receiving a sample of the form

S = f(x1; ); : : : ; (xm; )g:

We consider two ERMs: The rst, Agood, is de ned by

Agood(S) = h;;

that is, it outputs the hypothesis which predicts `\*' for every x 2 X . The second ERM, Abad, is de ned by

Abad(S) = hfx1;:::xmgc :

The following claim shows that the sample complexity of Abad is about jX j-times larger than the sample complexity of Agood. This establishes a gap between di erent ERMs. If X is in nite, we even obtain a learnable class that is not learnable by every ERM.

claim 29.9

1. Let ; > 0, D a distribution over X and hA 2 H. Let S be an i.i.d. sample consisting of m 1 log examples, sampled according to D and labeled by

hA. Then, with probability of at least 1 , the hypothesis returned by Agood will have an error of at most .

2. There exists a constant a > 0 such that for every 0 < < a there exists a distribution D over X and hA 2 H such that the following holds. The hypoth-

esis returned by Abad upon receiving a sample of size m , sampled

according to D and labeled by hA, will have error with probability e 16 .

1. Multiclass Learnability

Proof Let D be a distribution over X and suppose that the correct labeling is hA. For any sample, Agood returns either h; or hA. If it returns hA then its true error is zero. Thus, it returns a hypothesis with error only if all the m examples in the sample are from X n A while the error of h;, LD(h;) = PD[A], is . Assume m 1 log( 1 ); then the probability of the latter event is no more than (1 )m e m . This establishes item 1.

Next we prove item 2. We restrict the proof to the case that jX j = d < 1. The proof for in nite X is similar. Suppose that X = fx0; : : : ; xd 1g.

Let a > 0 be small enough such that 1 2 e 4 for every < a and x some < a. De ne a distribution on X by setting P[x0] = 1 2 and for all 1 i d 1, P[xi] = d2 1 . Suppose that the correct hypothesis is h; and let the sample size be m. Clearly, the hypothesis returned by Abad will err on all the examples from X which are not in the sample. By Cherno 's bound, if m d6 1 , then with probability e 16 , the sample will include no more than d 2 1 examples from X . Thus the returned hypothesis will have error . 

The conclusion of the example presented is that in multiclass classi cation, the sample complexity of di erent ERMs may di er. Are there \good" ERMs for every hypothesis class? The following conjecture asserts that the answer is yes.

conjecture 29.10 The realizable sample complexity of every hypothesis class H [k]X is

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| mH( ; ) = O~ |  | H | ) |  | : |
|  | Ndim( |  |  |  |
|  |  |  |  |  |  |

~

We emphasize that the O notation may hide only poly-log factors of ; , and Ndim(H), but no factor of k.

29.5 Bibliographic Remarks

The Natarajan dimension is due to Natarajan (1989). That paper also established the Natarajan lemma and the generalization of the fundamental theorem. Gen-eralizations and sharper versions of the Natarajan lemma are studied in Haussler

1. Long (1995). Ben-David, Cesa-Bianchi, Haussler & Long (1995) de ned a large family of notions of dimensions, all of which generalize the VC dimension and may be used to estimate the sample complexity of multiclass classi cation.

The calculation of the Natarajan dimension, presented here, together with calculation of other classes, can be found in Daniely et al. (2012). The example of good and bad ERMs, as well as conjecture [29.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page408), are from Daniely et al. (2011).

|  |  |
| --- | --- |
| 29.6 Exercises | 409 |
|  |  |

29.6 Exercises

1. Let d; k > 0. Show that there exists a binary hypothesis Hbin of VC dimension d such that Ndim(HbinOvA;k) = d.
2. Prove Lemma [29.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page405).
3. Prove Natarajan's lemma.

Hint: Fix some x0 2 X . For i; j 2 [k], denote by Hij all the functions f :

X n fx0g ! [k] that can be extended to a function in H both by de ning

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| and0use induction. | 0 | ) = j. Show that | jHj jHXnfx0gj | Pi6=j jHijj |
| f(x ) = i and by de ning f(x |  |  | + |

1. Adapt the proof of the binary fundamental theorem and Natarajan's lemma to prove that, for some universal constant C > 0 and for every hypothesis class of Natarajan dimension d, the agnostic sample complexity of H is

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | kd | + log(1= ) |  |
|  |  |  |  |
| mH( ; ) C | d log |  | 2 | : |

1. Prove that, for some universal constant C > 0 and for every hypothesis class of Natarajan dimension d, the agnostic sample complexity of H is

|  |  |  |
| --- | --- | --- |
| mH( ; ) C | d + log(1= ) | : |
|  |
| 2 |

1. Let H be the binary hypothesis class of (nonhomogenous) halfspaces in Rd. The goal of this exercise is to prove that Ndim(HOvA;k) (d 1) (k 1).
   1. Let Hdiscrete be the class of all functions f : [k 1] [d 1] ! f0; 1g for which there exists some i0 such that, for every j 2 [d 1]

|  |  |
| --- | --- |
|  | 8i < i0; f(i; j) = 1 while 8i > i0; f(i; j) = 0: |
|  | Show that Ndim(HdiscreteOvA;k) = (d 1) (k 1). |
| 2. | Show that Hdiscrete can be realized by H. That is, show that there exists |
|  | a mapping : [k 1] [d 1] ! Rd such that |
|  | Hdiscrete fh : h 2 Hg : |
|  | Hint: You can take (i; j) to be the vector whose jth coordinate is 1, whose |
|  | last coordinate is i and the rest are zeros. |
| 3. | Conclude that Ndim(HOvA;k) (d 1) (k 1). |

1. Compression Bounds

Throughout the book, we have tried to characterize the notion of learnability using di erent approaches. At rst we have shown that the uniform conver-gence property of a hypothesis class guarantees successful learning. Later on we introduced the notion of stability and have shown that stable algorithms are guaranteed to be good learners. Yet there are other properties which may be su cient for learning, and in this chapter and its sequel we will introduce two approaches to this issue: compression bounds and the PAC-Bayes approach.

In this chapter we study compression bounds. Roughly speaking, we shall see that if a learning algorithm can express the output hypothesis using a small sub-set of the training set, then the error of the hypothesis on the rest of the examples estimates its true error. In other words, an algorithm that can \compress" its output is a good learner.

30.1 Compression Bounds

To motivate the results, let us rst consider the following learning protocol. First, we sample a sequence of k examples denoted T . On the basis of these examples, we construct a hypothesis denoted hT . Now we would like to estimate the performance of hT so we sample a fresh sequence of m k examples, denoted V , and calculate the error of hT on V . Since V and T are independent, we immediately get the following from Bernstein's inequality (see Lemma [B.10](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page427)).

lemma 30.1 Assume that the range of the loss function is [0; 1]. Then,

s

" #

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P LD(hT ) LV (hT ) | 2LV (hT ) log(1= ) | | | | + | 4 log(1= ) | | | | : |
|  | j | V | j | j | V | j |  |
|  |  |  |  |  |  |

To derive this bound, all we needed was independence between T and V . Therefore, we can rede ne the protocol as follows. First, we agree on a sequence of k indices I = (i1; : : : ; ik) 2 [m]k. Then, we sample a sequence of m examples

1. = (z1; : : : ; zm). Now, de ne T = SI = (zi1 ; : : : ; zik ) and de ne V to be the rest of the examples in S. Note that this protocol is equivalent to the protocol we de ned before { hence Lemma [30.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page410) still holds.

Applying a union bound over the choice of the sequence of indices we obtain the following theorem.

"

P LD(hI ) LV (hI )

P 9I 2 [m]k s:t: LD(hI ) LV (hI )

"

r

|  |  |
| --- | --- |
| 30.1 Compression Bounds | 411 |
|  |  |

theorem 30.2 Let k be an integer and let B : Zk ! H be a mapping from sequences of k examples to the hypothesis class. Let m 2k be a training set size and let A : Zm ! H be a learning rule that receives a training sequence S of size m and returns a hypothesis such that A(S) = B(zi1 ; : : : ; zik ) for some (i1; : : : ; ik) 2 [m]k. Let V = fzj : j 2= (i1; : : : ; ik)g be the set of examples which were not selected for de ning A(S). Then, with probability of at least 1 over the choice of S we have

r

LD(A(S)) LV (A(S)) + LV (A(S)) 4k log(m= ) + 8k log(m= ) :

m m

Proof For any I 2 [m]k let hI = B(zi1 ; : : : ; zik ). Let n = m k. Combining Lemma [30.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page410) with the union bound we have

X

1. 2[m]k

#

2LV (hI ) log(1= ) + 4 log(1= )

n n

#

r

2LV (hI ) log(1= ) + 4 log(1= )

n n

mk :

Denote 0 = mk . Using the assumption k m=2, which implies that n = m k m=2, the above implies that with probability of at least 1 0 we have that

r

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LD(A(S)) LV (A(S)) + LV (A(S)) | 4k log(m= 0) | + | 8k log(m= 0) | | ; |
| m | m |  |
| which concludes our proof. |  |  |  |  |  |
| As a direct corollary we obtain: |  |  |  |  |  |

corollary 30.3 Assuming the conditions of Theorem [30.2](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page410), and further as-suming that LV (A(S)) = 0, then, with probability of at least 1 over the choice of S we have

8k log(m= )

LD(A(S)) :

These results motivate the following de nition:

definition 30.4 (Compression Scheme) Let H be a hypothesis class of functions from X to Y and let k be an integer. We say that H has a compression scheme of size k if the following holds:

For all m there exists A : Zm ! [m]k and B : Zk ! H such that for all h 2 H,

if we feed any training set of the form (x1; h(x1)); : : : ; (xm; h(xm)) into A and

then feed (xi1 ; h(xi1 )); : : : ; (xik ; h(xik )) into B, where (i1; : : : ; ik) is the output of A, then the output of B, denoted h0, satis es LS(h0) = 0.

It is possible to generalize the de nition for unrealizable sequences as follows.

1. Compression Bounds

definition 30.5 (Compression Scheme for Unrealizable Sequences)

Let H be a hypothesis class of functions from X to Y and let k be an integer. We say that H has a compression scheme of size k if the following holds:

For all m there exists A : Zm ! [m]k and B : Zk ! H such that for all h 2 H,

if we feed any training set of the form (x1; y1); : : : ; (xm; ym) into A and then

feed (xi1 ; yi1 ); : : : ; (xik ; yik ) into B, where (i1; : : : ; ik) is the output of A, then the output of B, denoted h0, satis es LS(h0) LS(h).

The following lemma shows that the existence of a compression scheme for the realizable case also implies the existence of a compression scheme for the unrealizable case.

lemma 30.6 Let H be a hypothesis class for binary classi cation, and assume it has a compression scheme of size k in the realizable case. Then, it has a compression scheme of size k for the unrealizable case as well.

Proof Consider the following scheme: First, nd an ERM hypothesis and denote it by h. Then, discard all the examples on which h errs. Now, apply the realizable compression scheme on the examples that have not been removed. The output of the realizable compression scheme, denoted h0, must be correct on the examples that have not been removed. Since h errs on the removed examples it follows that the error of h0 cannot be larger than the error of h; hence h0 is also an ERM hypothesis. 

30.2 Examples

In the examples that follows, we present compression schemes for several hy-pothesis classes for binary classi cation. In light of Lemma [30.6](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page412) we focus on the realizable case. Therefore, to show that a certain hypothesis class has a com-pression scheme, it is necessary to show that there exist A; B; and k for which LS(h0) = 0.

30.2.1 Axis Aligned Rectangles

Note that this is an uncountable in nite class. We show that there is a simple compression scheme. Consider the algorithm A that works as follows: For each dimension, choose the two positive examples with extremal values at this dimen-sion. De ne B to be the function that returns the minimal enclosing rectangle. Then, for k = 2d, we have that in the realizable case, LS(B(A(S))) = 0.

30.2.2 Halfspaces

Let X = Rd and consider the class of homogenous halfspaces, fx 7!sign(hw; xi) :

1. 2 Rdg.
3. kxik2+kwk2
4. kwk2

|  |  |
| --- | --- |
| 30.2 Examples | 413 |
|  |  |

A Compression Scheme:

W.l.o.g. assume all labels are positive (otherwise, replace xi by yixi). The com-pression scheme we propose is as follows. First, A nds the vector w which is in the convex hull of fx1; : : : ; xmg and has minimal norm. Then, it represents it as a convex combination of d points in the sample (it will be shown later that this is always possible). The output of A are these d points. The algorithm B receives these d points and set w to be the point in their convex hull of minimal norm.

Next we prove that this indeed is a compression sceme. Since the data is linearly separable, the convex hull of fx1; : : : ; xmg does not contain the origin. Consider the point w in this convex hull closest to the origin. (This is a unique point which is the Euclidean projection of the origin onto this convex hull.) We claim that w separates the data.[1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page413) To see this, assume by contradiction that

hw; xii 0 for some i. Take w0 = (1 )w + xi for = 2 (0; 1).

Then w0 is also in the convex hull and

kw0k2 = (1 )2kwk2 + 2kxik2 + 2 (1 )hw; xii

(1 )2kwk2 + 2kxik2

= kxik4kwk2 + kxik2kwk4

(kwk2 + kxik2)2

= kxik2kwk2

kwk2 + kxik2

|  |  |
| --- | --- |
| = kwk2 | 1 |
|  |
| kwk2=kxik2 + 1 |
| < kwk2; |  |

which leads to a contradiction.

We have thus shown that w is also an ERM. Finally, since w is in the convex hull of the examples, we can apply Caratheodory's theorem to obtain that w is also in the convex hull of a subset of d + 1 points of the polygon. Furthermore, the minimality of w implies that w must be on a face of the polygon and this implies it can be represented as a convex combination of d points.

It remains to show that w is also the projection onto the polygon de ned by the d points. But this must be true: On one hand, the smaller polygon is a subset of the larger one; hence the projection onto the smaller cannot be smaller in norm. On the other hand, w itself is a valid solution. The uniqueness of projection concludes our proof.

30.2.3 Separating Polynomials

Let X = Rd and consider the class x 7!sign(p(x)) where p is a degree r polyno-mial.

1. It can be shown that w is the direction of the max-margin solution.

1. Compression Bounds

Note that p(x) can be rewritten as hw; (x)i where the elements of (x) are all the monomials of x up to degree r. Therefore, the problem of constructing a com-pression scheme for p(x) reduces to the problem of constructing a compression scheme for halfspaces in Rd0 where d0 = O(dr).

30.2.4 Separation with Margin

Suppose that a training set is separated with margin . The Perceptron algorithm guarantees to make at most 1= 2 updates before converging to a solution that makes no mistakes on the entire training set. Hence, we have a compression scheme of size k 1= 2.

30.3 Bibliographic Remarks

Compression schemes and their relation to learning were introduced by Little-stone & Warmuth (1986). As we have shown, if a class has a compression scheme then it is learnable. For binary classi cation problems, it follows from the funda-mental theorem of learning that the class has a nite VC dimension. The other direction, namely, whether every hypothesis class of nite VC dimension has a compression scheme of nite size, is an open problem posed by Manfred War-muth and is still open (see also (Floyd 1989, Floyd & Warmuth 1995, Ben-David & Litman 1998, Livni & Simon 2013).

1. PAC-Bayes

The Minimum Description Length (MDL) and Occam's razor principles allow a potentially very large hypothesis class but de ne a hierarchy over hypotheses and prefer to choose hypotheses that appear higher in the hierarchy. In this chapter we describe the PAC-Bayesian approach that further generalizes this idea. In the PAC-Bayesian approach, one expresses the prior knowledge by de ning prior distribution over the hypothesis class.

31.1 PAC-Bayes Bounds

As in the MDL paradigm, we de ne a hierarchy over hypotheses in our class H. Now, the hierarchy takes the form of a prior distribution over H. That is, we assign a probability (or density if H is continuous) P (h) 0 for each h 2 H and refer to P (h) as the prior score of h. Following the Bayesian reasoning approach, the output of the learning algorithm is not necessarily a single hy-pothesis. Instead, the learning process de nes a posterior probability over H, which we denote by Q. In the context of a supervised learning problem, where H contains functions from X to Y, one can think of Q as de ning a randomized prediction rule as follows. Whenever we get a new instance x, we randomly pick a hypothesis h 2 H according to Q and predict h(x). We de ne the loss of Q on an example z to be

|  |  |
| --- | --- |
| def | [`(h; z)]: |
| `(Q; z) = E |

h Q

By the linearity of expectation, the generalization loss and training loss of Q can be written as

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| def | | [LD(h)] | def | | [LS(h)]: |
| LD(Q) = | E | and LS(Q) = | E |
|  | h Q |  |  | h Q |  |

The following theorem tells us that the di erence between the generalization loss and the empirical loss of a posterior Q is bounded by an expression that depends on the Kullback-Leibler divergence between Q and the prior distribu-tion P . The Kullback-Leibler is a natural measure of the distance between two distributions. The theorem suggests that if we would like to minimize the gen-eralization loss of Q, we should jointly minimize both the empirical loss of Q and the Kullback-Leibler distance between Q and the prior distribution. We will

1. PAC-Bayes

later show how in some cases this idea leads to the regularized risk minimization principle.

theorem 31.1 Let D be an arbitrary distribution over an example domain Z. Let H be a hypothesis class and let ` : H Z ! [0; 1] be a loss function. Let P be a prior distribution over H and let 2 (0; 1). Then, with probability of at least 1 over the choice of an i.i.d. training set S = fz1; : : : ; zm g sampled according to D, for all distributions Q over H (even such that depend on S), we have

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | D |  | S |  | s |  |  |
|  |  |  | 2(m 1) |  |
| L |  | (Q) | L (Q) + | |  | D(QjjP ) + ln m= | ; |
|  |  |  |
|  |  |  |  |  |  |  |
| where |  |  |  |  |  |  |  |
|  |  |  | def | EQ[ln(Q(h)=P (h))] | | |  |
|  |  | D(QjjP ) = h | |  |
|  |  |  |  |  |  |  |  |

is the Kullback-Leibler divergence.

Proof For any function f(S), using Markov's inequality:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | f(S) |  |  |  | ES[ef(S)] | | |  |  |
| S | S |  | e ] |  |  | | : | (31.1) |
|  |  | e | |
| P[f(S) |  | ] = P[e |  |  |  |  |  |  |

Let (h) = LD(h) LS(h). We will apply Equation ([31.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page416)) with the function

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Q |  |  |  |  | h Q | ( (h))2 |  | jj |  |
| f(S) = sup |  | 2(m |  | 1) | E |  | D(Q P ) | : |

We now turn to bound ES[ef(S)]. The main trick is to upper bound f(S) by using an expression that does not depend on Q but rather depends on the prior probability P . To do so, x some S and note that from the de nition of D(QjjP ) we get that for all Q,

2(m 1) E ( (h))2 D(QjjP ) = E [ln(e2(m 1) (h)2 P (h)=Q(h))]

h Q h Q

ln E [e2(m 1) (h)2 P (h)=Q(h)]

h Q

|  |  |  |
| --- | --- | --- |
| = ln E [e2(m 1) (h)2 | ]; | (31.2) |
| h P |  |  |

where the inequality follows from Jensen's inequality and the concavity of the log function. Therefore,

E[ef(S)] E E [e2(m 1) (h)2 ]: (31.3)

1. S h P

The advantage of the expression on the right-hand side stems from the fact that we can switch the order of expectations (because P is a prior that does not depend on S), which yields

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| E[ef(S)] | h | E E[e2(m 1) (h)2 | | ]: | (31.4) |
| S |  | P S |  |  |
|  |  |  |

|  |  |
| --- | --- |
| 31.2 Bibliographic Remarks | 417 |
|  |

Next, we claim that for all h we have ES[e2(m 1) (h)2 ] m. To do so, recall that Hoe ding's inequality tells us that

P[ (h) ] e 2m 2 :

S

This implies that ES[e2(m 1) (h)2 ] m (see Exercise [1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page417)). Combining this with Equation ([31.4](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page416)) and plugging into Equation ([31.1](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page416)) we get

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S |  |  | e | | |  |
| P[f(S) |  | ] |  | m | : | (31.5) |
|  |  |  |

Denote the right-hand side of the above , thus = ln(m= ), and we therefore obtain that with probability of at least 1 we have that for all Q

2(m 1) E ( (h))2 D(QjjP ) = ln(m= ):

h Q

Rearranging the inequality and using Jensen's inequality again (the function x2 is convex) we conclude that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| h Q |  |  | 2 |  |  | 2(m 1) | |  |
|  | h Q |  |  |
| E | (h) |  | E | ( (h))2 |  | ln(m= ) + D(QjjP ) | : | (31.6) |
|  |  |  |

Remark 31.1 (Regularization) The PAC-Bayes bound leads to the following learning rule:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Given a prior P , return a posterior Q that minimizes the function | | | |  |
| s |  |  |  |  |
| 2(m 1) |  |  |  |
| LS(Q) + | D(QjjP ) + ln m= | : | | (31.7) |
|  |  |  |  |  |

This rule is similar to the regularized risk minimization principle. That is, we jointly minimize the empirical loss of Q on the sample and the Kullback-Leibler \distance" between Q and P .

31.2 Bibliographic Remarks

PAC-Bayes bounds were rst introduced by McAllester (1998). See also (McAllester 1999, McAllester 2003, Seeger 2003, Langford & Shawe-Taylor 2003, Langford 2006).

31.3 Exercises

1. Let X be a random variable that satis es P[X] e 2m 2 . Prove that

E[e2(m 1)X2 ] m.

1. PAC-Bayes
   1. Suppose that H is a nite hypothesis class, set the prior to be uniform over H, and set the posterior to be Q(hS) = 1 for some hS and Q(h) = 0 for all other h 2 H. Show that

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D | S |  |  | S |  | s | 2(m 1) |  |
| L | (h | ) |  | L | (h) + |  | ln(jHj) + ln(m= ) | : |
|  |  |  |
|  |  |  |  |  |  |  |  |

Compare to the bounds we derived using uniform convergence.

Derive a bound similar to the Occam bound given in Chapter [7](file:///C:\Users\shashankj\Downloads\understanding-machine-learning-theory-algorithms.doc#page83) using the PAC-Bayes bound