Machine Learning I

Vlad Tkachuk

December 12, 2024

Table of Contents

Notation Reference

Set notation

$\mathcal{X}$ A generic set of values. For example, $\mathcal{X} = \{0, 1\}$ is the set containing only 0 and 1, $\mathcal{X} = [0, 1]$ is the interval from 0 to 1 and $\mathcal{X} = \mathbb{R}$ is the set of real numbers. Most of the time caligraphic letters such as $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ will be used for sets.

$\mathcal{P}(\mathcal{X})$ The power set of $\mathcal{X}$, a set containing all possible subsets of $\mathcal{X}$.

$[a, b]$ Closed interval with $a < b$, including both a and b.

(a, b) Open interval with $a < b$, with neither a nor b in the set.

$(a, b]$ Open-closed interval with $a < b$, including b but not a.

$[a, b)$ Closed-open interval with $a < b$, including a but not b.

Tuples, Vectors and Matrices

x Unbold lowercase variables are generally scalars.

$(x_1, x_2, \ldots, x_d)$ A tuple; i.e., an ordered collection of d elements that can have duplicates.

$\mathbf{x}$ Bold lowercase variables are vectors. By default, vectors are column vectors.

$\mathbf{X}$ Bold uppercase variables are matrices. This bold variable looks like a random variable that is a vector, $\boldsymbol{X}$, but the random variable is italicized. It will often be clear from context when this is a random variable representing a vector and when it is a matrix.

$\mathbf{X}^\top$ The transpose of the matrix, where we swap the elements around the diagonal of the matrix. An $n \times d$ matrix consisting of n vectors each of dimension d can be expressed as

$$\mathbf{X} = [\mathbf{x}_1 \, \mathbf{x}_2 \, \ldots \, \mathbf{x}_n]^\top .$$

A vector $\mathbf{x}$ is a matrix (a $d \times 1$ matrix), and the transpose similarly flips the orientation: a row vector becomes a column vector, and a column vector becomes a row vector.

$\mathbf{a}^\top \mathbf{b}$ We primarily use the transpose to obtain the **dot product** between two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$:

$$\mathbf{a}^\top \mathbf{b} = \sum_{j=1}^{d} a_j b_j .$$

Function notation

$f : \mathcal{X} \to \mathcal{Y}$ The function is defined on domain $\mathcal{X}$ to co-domain $\mathcal{Y}$, taking values $x \in \mathcal{X}$ and sending them to $f(x) \in \mathcal{Y}$.

$\frac{df}{dx}(x)$ The derivative of a function at $x \in \mathcal{X}$, where $f : \mathcal{X} \to \mathbb{R}$ for $\mathcal{X} \subset \mathbb{R}$.

$\nabla f(\mathbf{x})$ The gradient of a function at $\mathbf{x} \in \mathcal{X}$, where $f : \mathcal{X} \to \mathbb{R}$ for $\mathcal{X} \subset \mathbb{R}^d$. It holds that

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_d}\right)^\top .$$

$\min_{a \in \mathcal{B}} g(a)$ The minimum value of a function c across values a in a set $\mathcal{B}$. Note that this is equivalent to $-\left(\max_{a \in \mathcal{B}} -g(a)\right)$.

$\operatorname{argmin}_{a \in \mathcal{B}} g(a)$ The item a in set $\mathcal{B}$ that produces the minimum value $g(a)$. Note that this is equivalent to $\operatorname{argmax}_{a \in \mathcal{B}} -g(a)$.

Random variables and probabilities

X A random variable is written in uppercase.

$\mathcal{X}$ The space of values for the random variable.

x Lowercase variable is an instance or outcome, $x \in \mathcal{X}$.

$\boldsymbol{X}$ A random variable representing a vector is written bold uppercase.

$\sim$ indicates that a random variable is distributed as or equivalently that the random variable is sampled according to e.g., $X \sim \mathcal{N}(\mu, \sigma^2)$.

Supervised Learning and ERM

$\mathcal{D}$ A dataset is a tuple $\mathcal{D} = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n))$.

D A random variable representing a dataset

$\mathcal{F}$ The *function class*

Useful formulas and rules

$$\begin{aligned}
&\log(x \cdot y) = \log(x) + \log(y)\\
&\log\left(\frac{x}{y}\right) = \log(x) - \log(y)\\
&\log(x^y) = y\log(x)
\end{aligned}$$

Chapter 1

Introduction

This book focuses on the fundamentals underlying machine learning. Machine learning can be well-characterized as a field of applied mathematics. To understand machine learning concepts, it is important to be comfortable with mathematical terminology and concepts. This requirement can seem like a barrier, since you have to learn the language of mathematics simultaneously to the machine learning concepts. This added difficulty is not so uncommon in learning: to learn about Spanish poetry, for example, you may first have to learn the underlying language (Spanish). If you understand that mathematics is just a language in which you are not yet proficient—rather than that you have some inherent inability—then you can embrace this part of machine learning and buckle down and learn the needed language. And, what better way to learn this language than immersion in a useful and fun topic: machine learning. Much of the required mathematical background will involve basic understanding of probability and optimization; this book will attempt to provide most of that required background throughout. However, we attempt to provide only the necessary math background at the beginning of the book, so as to progress quickly to the machine learning concepts. As such, occasionally we will introduce a new mathematical concept just-in-time when we need it, rather than all at once at the beginning of the book.

In the next section we give a high-level explanation of what is meant by learning and introduce supervised learning which will be the focus of this book. After that, we give an example of a supervised learning problem, predicting house prices based on a dataset of previously sold houses. We conclude the chapter by explaining the structure of this book.

1.1 What is Machine Learning?

In this book we will think of machine learning as learning that is done by a computer. Then the crucial question becomes: what is learning? Learning can be defined as the process of gaining *knowledge* from *experience.* This is a broad definition, and can encompass many different types of learning. In this course, we will focus primarily on *supervised learning*, but it is essential to understand the broader landscape of learning paradigms to appreciate where supervised learning fits within the field of machine learning.

To get a sense of the different types of learning, we will use the next few subsections to go over some examples. In each case, we will identify the experience, and the resulting knowledge that is gained through learning. The different types of learning will differ in how the experience is presented to the learner, and what type of experience it is. The examples are to be treated as useful tools for building ones intuitions about the different types of learning, they are not to be used as the formal definitions of the learning types. For the specific case of supervised learning, we will provide a more formal definition in Chapter 4, as it is the focus of this book; however, for all the other types of learning we only leave you with the following examples.

1.1.1 Unsupervised vs. Supervised vs. Reinforcement Learning

Unsupervised learning, *supervised learning*, and *reinforcement learning* differ in the type of experience. We use the example of a person learning to play tennis to illustrate these concepts.

Unsupervised Learning

- **Experience:** A person has a tennis racket and ball but lacks knowledge of the game's rules. They practice by hitting the ball against a wall.
- **Knowledge:** Improved motor skills and coordination.

Since the person lacks knowledge of the game's rules the experience they acquire by hitting the ball against the wall is considered unlabeled and the process of learning from unlabeled experience is called *unsupervised learning*.

Supervised Learning

- **Experience:** A person watches instructional videos or tutorials that demonstrate the correct techniques and strategies for playing tennis.
- **Knowledge:** How to play tennis well

Here, the experience is *labeled*, as the correct techniques and strategies are provided to the person, making this an instance of *supervised learning*.

Reinforcement Learning

- **Experience:** The person knows the rules of tennis and plays games of tennis. They receive feedback based on the success of their actions (ex: they hit the the ball and it goes over the net). The person is not told what the correct techniques and strategies are, instead they only get feedback on the success of their actions.
- **Knowledge:** How to play tennis well

Notice how the Knowledge gained in the supervised learning example is the same as the knowledge gained in this case. However, the experience is different. In supervised learning the person is told what the correct techniques and strategies are, which implies that all other techniques and strategies are considered wrong or less correct. In reinforcement learning the person is not told what the correct techniques and strategies are, they only receive feedback on the success of their actions[1]. This scenario illustrates *reinforcement learning*, where learning is driven by feedback from actions taken.

[1]Reinforcement learning usually also emphasizes the fact that experience is recieved with some structure across time. For instance in the tennis example the strategies you take in a later part of a game depend on how you did at the start of the game. Although this is another important distinction, for the purposes of focusing only how labeled data differs from the data recieved in reinforcement learning we exclude this temporal aspect from our discussion.

1.1.2 Offline (Batch) vs. Online Learning

How data is processed during learning also creates an important distinction. The two distinctions are called *offline (batch) learning* and *online learning.* We use the example of learning language to illustrate the difference.

Offline (Batch) Learning

- **Experience:** All available text data from the internet is collected as a single, large dataset.
- **Knowledge:** A language model like ChatGPT that can generate and understand sentences.

Offline learning involves processing all the experience as one large batch.

Online Learning

- **Experience:** A person engages in language lessons and real-time conversations with others.
- **Knowledge:** The individual progressively develops the ability to comprehend and produce sentences in the language.

The key distinction from offline learning lies in how the experience is processed. In online learning, the learning is done incrementally, integrating new experience as it is encountered, rather than all at once.

1.1.3 Stochastic vs. Adversarial Learning

How the experience is generated also differentiates learning types. Two such categories are *stochastic learning* and *adversarial learning.* In this case we use the example of a person learning to play two different games: slot machines and chess.

Stochastic Learning

- **Experience:** A person is at a casino playing 3 different slot machines, each producing different random payouts.
- **Knowledge:** Knows which slot machine offers the highest expected payout based.

The experience is generated by a random process, the slot machines, and the person is learning from this random process. Stochastic learning involves learning from experience that is generated by a random processes.

Adversarial Learning

- **Experience:** A person plays games of chess where their opponent actively tries to defeat them.
- **Knowledge:** How to play chess well.

The important distinction here is that the experience is not generated by a random process, but rather by an opponent who is actively trying to defeat the person. The opponent is often called an adversary and learning from experience that is generated by an adversary it is called Adversarial learning.

1.1.4 Focus of This Book

Roughly speaking, in this book our focus will be on designing algorithms that will take as input experience and output knowledge. We will focus on supervised, offline, stochastic learning, which we will simply refer to as supervised learning, following common practice in the field. In essence, supervised learning involves learning from a randomly sampled batch of labeled data. While we will provide a more rigorous definition of supervised learning in Chapter 4, for now, we present an example to illustrate how an algorithm might do supervised learning.

1.2 An Example of Supervised Learning

In this example we will formulate the problem of predicting house prices as a supervised learning problem. We will do this in fairly mathematical terms. If you are not comfortable with the math, don't worry, we will cover all the necessary math in the next few chapters. The goal of this section is to give you a sense of what you will be learning in this book.

Consider the setting where you would like to predict the price of a house based on some features of the house. For instance you might have house features such as the number of rooms it has and its age. Without any previous examples of houses and their prices, it would be hard to predict the price of an unseen house. Instead assume we are randomly given some examples. We will represent each example as $(\mathbf{x}, y)$ where $\mathbf{x} \in \mathbb{R}^2$ is a vector representing the features (number of rooms and age) of the house and $y \in \mathbb{R}$ is the corresponding selling price. These examples are our exactly the experience we mentioned in the previous section The experience is often called a dataset and denoted as $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$ if we have n examples. Notice how this is the supervised learning setting since the dataset is randomly given to us as a batch of labeled examples (where the label here is the price).

Recall that our goal is to predict the price of a house based on its features. The ability to do this would be considered knowledge learned from experience. We will represent this knowledge as a predictor function $\hat{f} : \mathbb{R}^2 \to \mathbb{R}$ that takes as input the features of a house and outputs a prediction of the price. The question remains of how to design an algorithm that learns a predictor $\hat{f}$ from the dataset $\mathcal{D}$. We will call this algorithm the *learner* and represent it as a function $\mathcal{A}$ that takes as input the dataset $\mathcal{D}$ and outputs a predictor $\hat{f}$. First we have to decide what a predictor that is good means. One common definition is that a good predictor is one that predicts the price of unseen houses well. Since we do not have access to unseen houses (by the definition of unseen), we cannot measure the quality

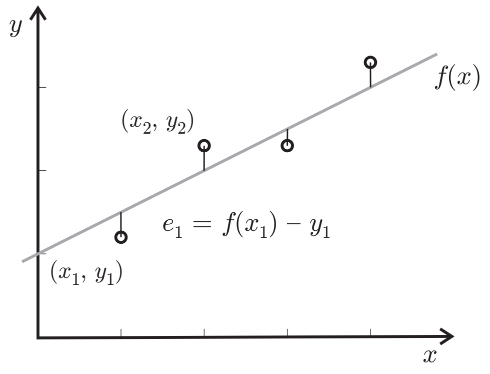


Figure 1.1: An example of a linear function fit to the dataset $\mathcal{D} = \{(1, 1.2), (2, 2.3), (3, 2.3), (4, 3.3)\}$. *The x-axis represents the number of rooms, and the y-axis represents the price in hundreds of thousands.*

of a predictor directly. Instead, we assume that our dataset is a good representation of all the houses we might see (since it is randomly samples) and instead measure the quality of a predictor on the dataset.

Mathematically the objective of the learner $\mathcal{A}$ is to output a predictor $\hat{f}$ that minimizes average *loss* on the dataset $\mathcal{D}$. The loss measures how well the predictor $\hat{f}$ predicts the price of the house given its features. A common choice for the loss is the squared loss $(\hat{f}(\mathbf{x}) - y)^2$. Notice that this loss function is a reasonable choice since if the predictor $\hat{f}$ outputs y when given $\mathbf{x}$ then the loss is zero. The average loss of a predictor f on the dataset $\mathcal{D}$ is then given by:

$$\sum_{i=1}^{n} (\hat{f}(\mathbf{x}_i) - y_i)^2$$

As we will see later, this optimization problem is simple to solve if we restrict $\hat{f}$ to only linear functions. The solution is a straight line that tries to best fit the prices y_i. A simple illustration of such a predictor function, for only one feature (the number of rooms) and 4 examples, is depicted in Figure 1.1.

To recap, the learner $\mathcal{A}$ we have designed takes as input a dataset $\mathcal{D}$ and outputs a predictor $\hat{f}$ that minimizes the average loss on the dataset (the predictor is the best fitting line in this case). There are several questions that we might ask about this predictor. For example, how well does $\hat{f}$ predict the price of unseen houses? How does the quality of the predictor depend on the size of the dataset? How does restricting the predictor to linear functions affect the quality of the predictor? How does the predictor change if the dataset changes? Answering these questions will require tools from probability, which we will cover in these notes.

1.3 Structure of the Book

In Chapters 2, 3 and 5 we will provide most of the needed background in mathematics, probability and statistics, for the rest of the book. Math concepts such as basic set notation, function notation, and calculus basics are covered in Chapter 2. The necessary probability is covered in Chapter 3 and the necessary estimation is covered in Chapter 5. After reading these chapters, look over the notation section at the beginning of the book, which summarizes the formulas, as well as the notation we will use throughout. Some of the definitions in the notation section will not be addressed in Chapters 2, 3 and 5, but rather just-in-time when we need to start using it later in the notes.

In Chapter 4 we will formalize the supervised learning problem, and Chapter 6 we will provide some methods for solving it. We will learn in Chapter 4 that supervised learning problems can split into: regression, where the output is a continuous value, and classification, where the output is a discrete label. In Chapter 6 we learn how to use a closed form solution or gradient descent to solve regression problems.

Next, we will be interested in evaluating the learned functions, and in understanding how to modify the learning process to improve the predictions (Chapter 7 and **??**). Here we will be introduce to key concepts such as overfitting, underfitting, and how it related to the bias-variance tradeoff (Chapter 7). To address the issues of overfitting and underfitting, we will introduce regularization techniques (**??**).

In Chapter 8 we will change our objective from prediction to learning the parameters of a distribution. We will see how this can be solved using maximum likelihood estimation, and maximum a posteriori estimation. It will turn out that these two problems are closely related to the prediction problem, and that the same optimization techniques can be used to solve them.

Having studied regression problems in detail, we will move on to classification problems in Chapter 9. We will introduce a new technique called logistic regression. Fortunately, we will find that many of the same techniques used for regression can be applied to classification.

Chapter 2

Math Review

In Chapter 1, we gave an example of what a supervised learning problem looks like. We informally defined supervised learning as learning from a randomly sampled batch of labeled data.

The goal of this chapter is to provide us with the mathematical tools needed to formally define part of the supervised learning setting. In particular, by the end of this chapter you should be able to understand that:

1. A dataset is a batch of labeled data, which can be formally represented as a tuple of tuples $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)) \in (\mathcal{X} \times \mathcal{Y})^n$ where $\mathbf{x}_i \in \mathcal{X}$ is a feature vector, and $y_i \in \mathcal{Y}$ is a label for all $i \in \{1, \ldots, n\}$.

2. A learner can be formally represented as a function $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \{f | f : \mathcal{X} \to \mathcal{Y}\}$ which is a function that takes as input a dataset and outputs a predictor function.

3. A predictor can be formally represented as a function $f : \mathcal{X} \to \mathcal{Y}$ which is a function that takes as input a feature vector and outputs a label.

To achieve this goal we will cover sets, tuples, vectors, and functions. We will also cover summation, itegration and derivatives in this chapter, which will be useful for understanding probability concepts in Chapter 3. After this chapter we will still be missing the definition of what a randomly sampled dataset means, which we will cover in Chapter 3, and what is the objective of the learner, which we will cover in Chapter 4.

2.1 Sets

A *set* is a collection of distinct and unordered objects. For example, the set $\{0, 1, 2\}$ contains the numbers 0, 1, and 2 and $\{0, 1, 2, 2\}$ is not a valid set since it has duplicates. The set $\{\text{cat}, \text{dog}\} = \{\text{dog}, \text{cat}\}$ since order doesn't matter. Some other common sets are: $\varnothing$ (empty set), $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ (natural numbers), and $\mathbb{R}$ (real numbers). As with numbers, it useful to assign variables to sets. Unlike variables for numbers (where we use lower case letters), for sets we will use caligraphic letters, such as $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$. Assigning a variable to a set allows us to refer to the set by the variable name, rather than listing all the elements in the set. For example, we will define $\mathcal{X} = \{0, 1, 2\}$ and $\mathcal{Y} = \{\text{cat}, \text{dog}\}$ for the remainder of this subsection (unless otherwise stated). We will also use $\mathcal{Z}$ to refer to any arbitrary set throughout.

Exercise 2.1: Write a valid set and an invalid set. $\square$

We can talk about elements of a set by using the symbol $\in$ and subsets using $\subset$. The negations of these symbols are $\notin$ and $\not\subset$. Some examples include: cat $\in \mathcal{Y}$, cat $\notin \mathcal{X}$,

$\mathcal{X} \subset \mathbb{N}, \mathcal{X} \subset \mathbb{R}, \mathbb{N} \subset \mathbb{R}$, $\mathcal{X} \not\subset \mathcal{Y}$, and $\mathcal{X} \subset \mathcal{X}$. Since we will not define the set of real numbers $\mathbb{R}$ formally we can get a sense of what the set contains by listing some of the elements: $\{0, 1, -2, 1/2, 0.23, \pi\} \subset \mathbb{R}$, $\infty \notin \mathbb{R}$.

Exercise 2.2: Is $\pi \in \mathbb{N}$? ☐

Exercise 2.3: Is $\{-1, 0, 1\} \subset \mathbb{N}$? ☐

Some other useful sets are the *intervals* which are sets of real numbers between two values. There are three types of intervals: open, closed, half-open (or half-closed if you prefer). An open interval does not include the endpoints and is written as (a, b), a closed interval includes the endpoints and is written as $[a, b]$, a half-open interval includes one endpoint and is written as $[a, b)$ or $(a, b]$.

We can perform operations on sets such as *union* ($\cup$), *intersection* ($\cap$), *difference* ($\backslash$), and *complement* ($\mathcal{Z}^c$). The union of two sets $\mathcal{X}$ and $\mathcal{Y}$ is the set of elements that are in either $\mathcal{X}$ or $\mathcal{Y}$ or both and is written as $\mathcal{X} \cup \mathcal{Y}$. The intersection of two sets $\mathcal{X}$ and $\mathcal{Y}$ is the set of elements that are in both $\mathcal{X}$ and $\mathcal{Y}$ and is written as $\mathcal{X} \cap \mathcal{Y}$. The difference of two sets $\mathcal{X}$ and $\mathcal{Y}$ is the set of elements that are in $\mathcal{X}$ but not in $\mathcal{Y}$ and is written as $\mathcal{X} \backslash \mathcal{Y}$. To define the complement of a set $\mathcal{Z}$ we need to define the *universal set* $\mathcal{U}$ which is the set of all possible elements. The complement of a set $\mathcal{Z}$ is the set of elements that are in $\mathcal{U}$ but not in $\mathcal{Z}$ and is written as $\mathcal{Z}^c = \mathcal{U} \backslash \mathcal{Z}$. Some examples of these operations are: $\{0, 1, 2\} \cup \{2, 3\} = \{0, 1, 2, 3\}$, $\{0, 1, 2\} \cap \{2, 3\} = \{2\}$, $\{0, 1, 2\} \backslash \{2, 3\} = \{0, 1\}$, and $\{0, 1, 2\}^c = \{3, 4, 5, \ldots\}$.

Exercise 2.4: Given the universal set $\mathcal{U} = \{0, 1, 2, 3, 4, 5\}$, what is the complement of the set $\mathcal{Z} = \{0, 1, 2\}$? ☐

Exercise 2.5: Given the sets $\mathcal{X} = \{0, 1\}$ and $\mathcal{Y} = \{1, 2\}$, find $\mathcal{X} \cup (\mathcal{X} \cap \mathcal{Y})$. ☐

To construct more complex sets we can use *set builder notation* which is a way to define a set by specifying a property that the elements must satisfy. The structure of set builder notation is {element|property}, and is read as "the set of elements such that the elements satisfy the property". For example, the set $\mathcal{X} = \{0, 1, 2\}$ can be defined as $\{x | x \in \mathbb{N}, x < 3\}$ which is read as "the set of natural numbers less than 3". The power of set builder notation is that it allows us to define complex sets without listing all the elements. Some examples are the even numbers $\{x | x \in \mathbb{N}, x \text{ is even}\}$, and the prime numbers $\{x | x \in \mathbb{N}, x \text{ is prime}\}$. The set builder notation also allows us to define some previous definitions more formally. For instance, set difference can be defined as $\mathcal{X} \backslash \mathcal{Y} = \{x | x \in \mathcal{X}, x \notin \mathcal{Y}\}$. Finally, we can use the set builder notation to define the *power set* $\mathcal{P}(\mathcal{Z}) = \{\mathcal{S} | \mathcal{S} \subset \mathcal{Z}\}$ which is the set of all subsets of $\mathcal{Z}$.

Exercise 2.6: Express the following in set builder notation: the set of all real numbers greater than -1 and less than 2. ☐

Exercise 2.7: Express the set of all real numbers that are not natural numbers ☐

The number of elements in a set $\mathcal{Z}$ is called the *cardinality* of the set and is denoted as $|\mathcal{Z}|$. For example, $|\mathcal{X}| = 3, |\mathcal{Y}| = 2$ and $|\varnothing| = 0$. The cardinality of a set can be infinite, such as $|\mathbb{N}|$ and $|\mathbb{R}|$. It turns out that there are different sizes of infinity and $|\mathbb{N}|$ is smaller than $|\mathbb{R}|$. The cardinality of the set of natural numbers is called *countably infinite* and the cardinality of the set of real numbers is called *uncountably infinite*. Roughly speaking, a set is countably infinite if you can list all the elements in the set and uncountably infinite if you cannot. Some examples of countably infinite sets are $\mathbb{N}$, $\{x \in \mathbb{N} | x \text{ is even}\}$ (the even numbers), and $\{x \in \mathbb{N} | x \text{ is prime}\}$ (the prime numbers). Examples of uncountably infinite sets are $\mathbb{R}$, $[0, 1]$, and $[0, 1] \cup [2, 3]$. If the cardinality of a set is finite or countably infinite we will say the set if *countable*, while if the cardinality is uncountably infinite we will say the set is *uncountable*.

2.2 Tuples

An *tuple* is a collection of ordered objects with duplicates allowed. For example $(0, 1, 2)$ is a tuple that contains the numbers 0, 1, and 2 in that order, and (cat, dog) is a tuple that contains the animals cat and dog in that order. Notice that $(\text{cat}, \text{dog}) \neq (\text{dog}, \text{cat})$ since the order matters. Also, $(0, 1, 2) \neq (0, 1, 2, 2)$ since duplicates are allowed in tuples. Notice that the brackets are round instead of curly. This is the standard notation; however, you might have noticed that we already used round brackets for intervals. As such, it will be the job of the reader to determine if the round brackets are for intervals or tuples based on the context.

Tuples are often created through the *Cartesian product* which is a way to create a set of tuples from two sets. The Cartesian product of two sets $\mathcal{X}$ and $\mathcal{Y}$ is written as $\mathcal{X} \times \mathcal{Y}$, and is the set of all possible tuples where the first element is from $\mathcal{X}$ and the second element is from $\mathcal{Y}$. We can use set builder notation to formally define the Cartesian product as $\mathcal{X} \times \mathcal{Y} = \{(x, y) | x \in \mathcal{X}, y \in \mathcal{Y}\}$. For example,

$$\mathcal{Y} \times \mathcal{X} = \{(\text{cat}, 0), (\text{cat}, 1), (\text{cat}, 2), (\text{dog}, 0), (\text{dog}, 1), (\text{dog}, 2)\}.$$

If the two sets are the same set then often the Cartesian product is written as $\mathcal{Z}^2 = \mathcal{Z} \times \mathcal{Z}$. A classic example is the real plane $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$ which is the set of all possible points in the plane, such as $(0, 0) \in \mathbb{R}^2$ and $(\pi, 2) \in \mathbb{R}^2$. The Cartesian product can be extended to any finite number of sets, such as $\mathcal{Z}_1 \times \mathcal{Z}_2 \times \cdots \times \mathcal{Z}_n = \{(z_1, z_2, \ldots, z_n) | z_1 \in \mathcal{Z}_1, z_2 \in \mathcal{Z}_2, \ldots, z_n \in \mathcal{Z}_n\}$. Similarly, we use the notation $\mathcal{Z}^n = \mathcal{Z} \times \mathcal{Z} \times \cdots \times \mathcal{Z}$, where the cartesian product of $\mathcal{Z}$ is take n times.

Example 2.1: This notation will be used to define our dataset. We begin by defining an individual feature-label pair. Suppose the features belong to $\mathcal{X} = \mathbb{R}^2$ and labels belong to $\mathcal{Y} = \mathbb{R}$. Then $\mathcal{X} \times \mathcal{Y}$ would contain tuples of the form $((x_1, x_2), y)$ where $x_1, x_2, y \in \mathbb{R}$. Now, the dataset is a collection of n feature-label pairs, $\mathcal{D} = (((x_{1,1}, x_{1,2}), y_1), \ldots, ((x_{n,1}, x_{n,2}), y_n))$. Notice that the dataset $\mathcal{D}$ is an element of the set $(\mathcal{X} \times \mathcal{Y})^n = (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \times \cdots \times (\mathcal{X} \times \mathcal{Y})$, where the cartesian product of $\mathcal{X} \times \mathcal{Y}$ is taken n times. $\square$

Exercise 2.8: Explicitly write what the set $\{1, 2\} \times \{a, b\}$ looks like. $\square$

Exercise 2.9: Let $\mathcal{X} = \{0, 1\}$. What is the cardinality of the set $\mathcal{X}^3$ (the set of all ordered triples formed by the Cartesian product of $\mathcal{X}$)? $\square$

Exercise 2.10: Suppose you wanted to represent any house by using three features of it. The first feature was the number of rooms (an natural number), the year it was made (a real number), and the third feature was the year the house was built (a natural number).

How would you write the set of all possible houses that are represented by these features, such that an example of an element from this set is $(1, 1000.7, 2005)$? □

2.3 Vectors

The motivation to introduce *vectors* instead of just using *tuples* is that vectors have additional structure that allows us to perform mathematical operations such as addition, scalar multiplication, and computing the dot product. In particular, vectors enable us to model relationships between features and labels. For example, the target value y might be a linear function of the features, i.e.,

$$y = w_1 x_1 + w_2 x_2.$$

For the purposes of this course, we define a *vector space* as a set of vectors that can be added together and scaled by scalars (real numbers). For example, $\mathbb{R}^2$ is a vector space. A *vector* is an element of a vector space and will be denoted using a bold variable. We will write vectors in column format as follows:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2.$$

In this course, we will work exclusively with the vector spaces $\mathbb{R}$, $\mathbb{R}^2$, $\mathbb{R}^3$, and so on. When working with $\mathbb{R}$, we will omit the bold font and simply write $x \in \mathbb{R}$.

To see why $\mathbb{R}^2$ is a vector space, notice that for any two vectors $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$, and any scalar $c \in \mathbb{R}$, we have:

$$\mathbf{x} + \mathbf{y} = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \end{pmatrix} \in \mathbb{R}^2,$$

and

$$c\mathbf{x} = \begin{pmatrix} cx_1 \\ cx_2 \end{pmatrix} \in \mathbb{R}^2.$$

Thus, vector addition and scalar multiplication result in vectors that are still elements of $\mathbb{R}^2$.

In contrast, a set like $\mathcal{Y} = \{\text{cat}, \text{dog}\}$ is not a vector space, since there is no meaningful way to add cat + dog or to multiply cat by a scalar.

Some concrete examples of vectors are:

$$\mathbf{x} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \in \mathbb{R}^2, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \in \mathbb{R}^3, \quad y = 0 \in \mathbb{R}.$$

A useful operation on vectors is the *transpose*, denoted by $\mathbf{x}^\top$, which converts a column vector into a row vector or a row vector into a column vector. For instance,

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \mathbf{x}^\top = (x_1, x_2) \quad \text{and} \quad (x_1, x_2)^\top = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

The transpose is particularly useful when defining the *dot product* of two vectors, which is a method of multiplying two vectors together to obtain a scalar. The dot product is defined as follows. For vectors $\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \cdots \\ w_d \end{pmatrix}$ and $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_d \end{pmatrix}$ in $\mathbb{R}^d$, the dot product is:

$$\mathbf{w}^\top \mathbf{x} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d.$$

The dot product outputs a scalar value in $\mathbb{R}$ and is commutative, i.e., $\mathbf{w}^\top \mathbf{x} = \mathbf{x}^\top \mathbf{w}$.

Additionally, using the transpose allows us to write vectors in row format when appropriate, saving space in our notation.

It is important to note that while the notation for the transpose of a vector $\mathbf{x}^\top$ and a tuple (x_1, x_2) looks similar, they serve different purposes:

- A *tuple* (x_1, x_2) can have elements from any set. This means that x_1, x_2 do not need to be elements of $\mathbb{R}$. Instead we could have that $x_1 \in \mathbb{N}$ while $x_2 \in \{\text{cat}, \text{dog}\}$.
- A *vector* $\mathbf{x}$ is an element of a vector space (like $\mathbb{R}^2$) and carries additional structure that allows for algebraic operations like vector addition and scalar multiplication.
- The *transpose* $\mathbf{x}^\top$ is a row vector, which is useful for certain mathematical operations like computing the dot product.

For practical purposes in this course, we can often treat tuples and vectors interchangeably when dealing with elements of $\mathbb{R}^d$, since they both can represent points in d-dimensional space. In particular, in this course all vectors can be thought of as tuples, but not all tuples can be thought of as vectors (again this is since math operations with tuples might not always make sense). The context will make it clear whether we are considering an object as a tuple (for data representation) or as a vector (for mathematical operations).

Exercise 2.11: Given two vectors $\mathbf{x} = (1, 3)^\top$ and $\mathbf{y} = (2, -1)^\top$, calculate $3\mathbf{x} - 2\mathbf{y}$. $\square$

Exercise 2.12: What is the dot product of the vectors $\mathbf{x} = (2, 3, 1)^\top$ and $\mathbf{y} = (1, 4, 0)^\top$? $\square$

We will also need to define a *matrix* which we will think of as a multiple row vectors vertically stacked. A matrix is denoted by a bold capital letter, such as $\mathbf{M}$, and is written using square brackets as:

$$\mathbf{M} = \begin{bmatrix} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \end{bmatrix} = \begin{bmatrix} v_{1,1} & v_{1,2} \\ v_{2,1} & v_{2,2} \end{bmatrix}$$

where $\mathbf{v}_1 = (v_{1,1}, v_{1,2})^\top \in \mathbb{R}^2$ and $\mathbf{v}_2 = (v_{2,1}, v_{2,2})^\top \in \mathbb{R}^2$. In this case the matrix only has 2 rows (i.e. 2 row vectors verticlaly stacked), but we can have matrices with any number of rows. Matrix multiplication is a way to multiply a matrix by a vector or another matrix and can be defined through the dot product. For example, the matrix $\mathbf{M}$ can be multiplied by the vector $\mathbf{x}$ as follows:

$$\mathbf{M}\mathbf{x} = \begin{bmatrix} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1^\top \mathbf{x} \\ \mathbf{v}_2^\top \mathbf{x} \end{pmatrix} = \begin{pmatrix} v_{1,1} x_1 + v_{1,2} x_2 \\ v_{2,1} x_1 + v_{2,2} x_2 \end{pmatrix}$$

Exercise 2.13: Given the matrix $\mathbf{M} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and the vector $\mathbf{x} = (1, 2)^\top$, calculate $\mathbf{M}\mathbf{x}$. $\square$

2.4 Functions

A *function* is a rule that assigns an output to an input. For example, the function $f(x) = x^2$ assigns the output $f(2) = 4$ to the input $x = 2$. More generally we write a function as $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is the domain of the function and $\mathcal{Y}$ is the codomain. This is read as "f is a function from $\mathcal{X}$ to $\mathcal{Y}$". The domain is the set of all possible inputs and the codomain is the set of all possible outputs. For example, the function $f : \mathbb{R} \to \mathbb{R}$ is the function that takes a real number as input and outputs a real number. Once we specify the function, we can talk about its *range*, which is the set of all possible outputs for this specific function. Next we demonstrate these concepts through various examples, which also introduce some commonly used functions such as the logarithm and exponential functions.

Example 2.2: $f : \{0, 1, 2\} \to \mathbb{N}$ is a function with domain $\{0, 1, 2\}$ and the codomain $\mathbb{N}$. To define the function, we need to specify the output for each input. Let us select $f(0) = 1, f(1) = 2$ and $f(2) = 2$. The range of this specific function is $\{1, 2\}$ since that is the set of all its possible outputs. If instead we selected $f(0) = 1, f(1) = 2$ and $f(2) = 3$, then the range would be $\{1, 2, 3\}$. Importantly, the range is specific to the function we select and is always a subset of the codomain $\square$

Example 2.3: $f : \mathbb{R} \to \mathbb{R}$ is a function with domain $\mathbb{R}$ and the codomain $\mathbb{R}$. If we define $f(x) = x^2$, then the range of this function is $[0, \infty)$ since the function $f(x) = x^2$ is always non-negative. $\square$

Example 2.4: Let $f : \mathbb{R} \to \mathbb{R}$. If we define $f(x) = e^x$, then the range of this function is $(0, \infty)$. Notice how the range does not include zero, since e^x approaches but never reaches zero. Occasionally we will write $\exp(x) = e^x$ to make the notation more readable if the exponent is complicated. $\square$

Example 2.5: The function $\ln(x) = \log_e(x)$ is the natural logarithm function. If we define $f(x) = \ln(x)$, then its domain cannot be $\mathbb{R}$ since the logarithm is not defined for negative numbers. Instead we write $f : (0, \infty) \to \mathbb{R}$. $\square$

There are a few rules for logarithms and exponentials that will be useful later in the notes. These include:

$$\begin{aligned}
&\ln(ab) = \ln(a) + \ln(b) \\
&\ln\left(\frac{a}{b}\right) = \ln(a) - \ln(b) \\
&\ln\left(a^b\right) = b\ln(a) \\
&\exp(a + b) = \exp(a)\exp(b) \\
&\exp(a)^b = \exp(ab)
\end{aligned}$$

Exercise 2.14: Show the second logarithm rule above, using the first and third rule. □

Functions can be combined to create new functions. For example, if we have two functions $f : \mathcal{X} \to \mathcal{Y}$ and $g : \mathcal{Y} \to \mathcal{Z}$, then we can define a new function $h : \mathcal{X} \to \mathcal{Z}$ as $h(x) = g(f(x))$. A notable example is if $g(y) = \ln(y)$ and $f(x) = \exp(x)$ then $h(x) = \ln(\exp(x)) = x$, which shows that the logarithm and exponential functions are inverses of each other.

Functions are not limited to only outputting numbers. They can also output functions. This means the codomain becomes a set of functions.

Example 2.6: $g : \mathbb{R} \to \{f | f : \mathbb{R} \to \mathbb{R}\}$ is a function with domain $\mathbb{R}$ and codomain the set of all functions from $\mathbb{R}$ to $\mathbb{R}$. If we define $g(z) = f_z$ where $f_z(x) = z + x$, then g is a function that takes a number z and outputs a function f_z that adds z to its input. □

Example 2.7: Let $g : \mathbb{R} \to \{f | f : \mathbb{R} \to \mathbb{R}\}$. If we define $g(z) = f_z$ where $f_z(x) = 1$ if $z = x$ and $f_z(x) = 0$ otherwise, then g is a function that takes a number z and outputs a function f_z that is 1 at z and 0 elsewhere. □

A function that outputs a function will be a very important concept in machine learning because the learner will take as input dataset and output a predictor function.

Example 2.8: In the section about tuples (Section 2.2), we defined a dataset

$$\begin{aligned}\mathcal{D} &= (((x_{1,1}, x_{1,2}), y_1), \ldots, ((x_{n,1}, x_{n,2}), y_n)) \\ &= ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)),\end{aligned}$$

where we simplified the notation here by using vectors $\mathbf{x}_i = (x_{i,1}, x_{i,2})^\top$ for all $i \in \{1, \ldots, n\}$. We also showed that the dataset $\mathcal{D}$ is an element of the set $(\mathcal{X} \times \mathcal{Y})^n$ where $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \mathbb{R}$.

A learner $\mathcal{A}$ is a function that takes a dataset as input and outputs a predictor function $\hat{f} : \mathbb{R}^2 \to \mathbb{R}$. Formally,

$$\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \{f \mid f : \mathcal{X} \to \mathcal{Y}\}.$$

A specific example of such a learner is:

$$\mathcal{A}(\mathcal{D}) = \hat{f},$$

where

$$\hat{f}(\mathbf{x}) = \begin{cases} y_i & \text{if } \mathbf{x} = \mathbf{x}_j \text{ for some } j \in \{1, \ldots, n\}, \\ & \quad \text{then } i \in \{1, \ldots, n\} \text{ is the smallest value that satisfies } \mathbf{x} = \mathbf{x}_i \\ 0 & \text{otherwise.} \end{cases}$$

This type of learner $\mathcal{A}$ is called a *memorizer*. We can see this by analyzing the behaviour of the predictor function $\hat{f}$ that $\mathcal{A}$ outputs. The predictor $\hat{f}$ takes as input any vector $\mathbf{x} \in \mathbb{R}^2$, but only outputs a value other than 0 if the input is one of the $\mathbf{x}_i$ vectors in the dataset $\mathcal{D}$. In particular $\hat{f}$ outputs y_i only when $\mathbf{x} = \mathbf{x}_i$ for some $i \in \{1, \ldots, n\}$. As such, it is said to memorize all the $(\mathbf{x}_i, y_i)$ in the dataset $\mathcal{D}$. □

Another useful class of functions is *continuous functions*. Roughly speaking a function is called continuous if it does not have abrupt changes in value, or discontinuities[1]. The functions w^2 and $w + 5$ are both continuous: you can trace your finger continuously along the lines produced by these functions. The function

$$f(w) = \begin{cases} w^2 & w \in [-1, 1] \\ 5 + w & w < -1 \\ 5 - w & w > 1 \end{cases} \tag{2.1}$$

has two discontinuities, one at $w = -1$ and at $w = 1$. The function changes from $f(1) = 1$ and suddenly jumps up to > 5 right after 1. This discontinuous function is depicted in Figure 2.1. Note that the function restricted to particular subintervals is continuous: it is continuous on the interval $w \in [-1, 1]$, on the interval $(-\infty, -1)$ and on the interval $(1, \infty)$.

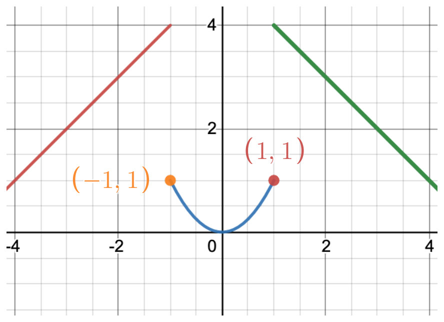


Figure 2.1: An example of a function that is not continuous, with formula in Eq. (2.1).

2.5 Summation and Integration

Summation and integration notations will be used frequently when we get to the next chapter on probability and need to calculate expected values. These operations, though conceptually distinct, share a common foundation: they represent the accumulation of quantities, either discrete or continuous, across a collection of elements.

2.5.1 Summation

When we are interested in computing the sum of all elements within a finite or countably infinite set $\mathcal{X}$, we utilize the summation symbol $\sum$. Specifically, the expression $\sum_{x \in \mathcal{X}} x$

[1]We do not need the formal definition of a continuous function, since the intuitive definition is sufficient. For you interest, the formal definition is that a function $f : \mathcal{X} \to \mathbb{R}$ is continuous at $x_0 \in \mathcal{X}$ if for any $\epsilon > 0$ there exists a $\delta > 0$ such that if $\|x - x_0\| < \delta$ for $x \in \mathcal{X}$ then $|f(x) - f(x_0)| < \epsilon$. In other words, for arbitrarily nearby points x and x_0 (δ can be made very small) the difference in function values is also arbitrarily small. The discontinuous function example does not satisfy this property at $x_0 = 1$, because in the neighborhood around 1, if we pick an $\epsilon < 4$, we can pick an arbitrarily small neighborhood (δ approaching zero), but the distance in the function values remains higher than ϵ.

represents the sum of the elements of $\mathcal{X}$. For instance, if $\mathcal{X} = \{0, 1, 2\}$, the corresponding summation is:

$$\sum_{x \in \mathcal{X}} x = 0 + 1 + 2 = 3.$$

Moreover, if our goal is to sum the values of a function $f(x)$ evaluated at each element of $\mathcal{X}$, we would write $\sum_{x \in \mathcal{X}} f(x)$. For example, if $f(x) = x^2$ and $\mathcal{X} = \{0, 1, 2\}$, the sum of the function values is given by:

$$\sum_{x \in \mathcal{X}} f(x) = 0^2 + 1^2 + 2^2 = 5.$$

The summation notation is equally valid when $\mathcal{X}$ is a tuple, whether it is finite or countably infinite. For example, if $\mathcal{X} = (0, 1, 2, 2)$, then:

$$\sum_{x \in \mathcal{X}} x = 0 + 1 + 2 + 2 = 5 \quad \text{and} \quad \sum_{x \in \mathcal{X}} f(x) = 0^2 + 1^2 + 2^2 + 2^2 = 9.$$

More generally, a tuple with n elements can be expressed as $\mathcal{X} = (x_1, x_2, \ldots, x_n)$. In this case, the sum of the elements of $\mathcal{X}$ is written as:

$$\sum_{x \in \mathcal{X}} x = x_1 + x_2 + \cdots + x_n.$$

This formulation naturally motivates an alternative, yet equivalent, notation often used when dealing with ordered collections like tuples. Specifically, we may write:

$$\sum_{i=1}^{n} x_i = \sum_{x \in \mathcal{X}} x,$$

where the index i ranges from 1 to n, reflecting the ordered nature of the elements in the tuple $\mathcal{X} = (x_1, \ldots, x_n)$.

2.5.2 Integration

If instead of summing discrete elements, we are interested in accumulating values over an uncountably infinite set, we use *integration*. The integral can be viewed as the continuous analog of summation, where we accumulate the values of a function over an interval.

For a continuous function $f(x)$ with a domain $\mathcal{X} \supset [a, b]$, the integral of $f(x)$ with respect to x over the interval $[a, b]$ is denoted by:

$$\int_{[a,b]} f(x) = \int_a^b f(x)\, dx$$

This represents the "total area" under the curve of $f(x)$ between $x = a$ and $x = b$, where the area is counted positively when the curve is above the x-axis and negatively when it is below.

Just as the summation notation allows us to accumulate function values across a discrete set, integration allows us to accumulate function values continuously across an interval. The integral can be interpreted as the limiting case of summation as the number of subdivisions of the interval increases infinitely and the width of each subdivision approaches zero.

For example, if we want to integrate the function $f(x) = x^2$ over the interval $[0, 1]$, we compute:

$$\int_0^1 x^2\, dx$$

We can evaluate this integral using the power rule for integration, which states that for $f(x) = x^n$, we have:

$$\int x^n\, dx = \frac{x^{n+1}}{n+1} + C$$

Applying this to $f(x) = x^2$, we get:

$$\int_0^1 x^2\, dx = \frac{x^3}{3}\Big|_0^1 = \frac{1^3}{3} - \frac{0^3}{3} = \frac{1}{3}$$

Thus, the area under the curve $f(x) = x^2$ from 0 to 1 is $\frac{1}{3}$.

Similar to summation, integration respects linearity, meaning that constants can be factored out, and the integral of a sum is the sum of the integrals:

$$\int_a^b cf(x)\, dx = c \int_a^b f(x)\, dx$$

$$\int_a^b [f(x) + g(x)]\, dx = \int_a^b f(x)\, dx + \int_a^b g(x)\, dx$$

As a specific example of this property, consider the integral of a constant c over an interval $[a, b]$:

$$\int_a^b c\, dx = c \int_a^b 1\, dx = c(b - a)$$

This result shows that the integral of a constant function over an interval is simply the product of the constant and the length of the interval.

2.5.3 Summation and Integration over Multivariate Functions

In many cases, we may be interested in summing or integrating functions that depend on more than one variable. These operations allow us to extend summation and integration from one-dimensional functions to higher-dimensional spaces.

Multivariate Summation

If we are summing a function that depends on multiple variables, such as $f(x, y)$, over a discrete set of points $\mathcal{X} \times \mathcal{Y}$, we can perform a double summation. This is often written as:

$$\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} f(x, y)$$

This notation indicates that for each element x in $\mathcal{X}$, we perform a sum over all elements y in $\mathcal{Y}$. For example, let $\mathcal{X} = \{0, 1\}$ and $\mathcal{Y} = \{1, 2\}$, and let $f(x, y) = x + y$. Then:

$$\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} f(x, y) = (0 + 1) + (0 + 2) + (1 + 1) + (1 + 2) = 1 + 2 + 2 + 3 = 8$$

This demonstrates how we can compute sums over multivariate functions by iterating over all combinations of the variables.

Multivariate Integration

Similarly, we can integrate functions that depend on multiple variables. If $f(x, y)$ is a continuous function over the rectangular domain $\mathcal{X} = [a_1, b_1]$ and $\mathcal{Y} = [a_2, b_2]$, the double integral is written as:

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x, y)\, dy\, dx$$

This represents the total "volume" under the surface defined by $f(x, y)$. The integration is performed first over y, for a fixed x, and then over x.

As an example, consider the function $f(x, y) = x + y$ over the domain $\mathcal{X} = [0, 1]$ and $\mathcal{Y} = [0, 1]$. The double integral is:

$$\int_0^1 \int_0^1 (x + y)\, dy\, dx$$

We can evaluate the inner integral with respect to y first:

$$\int_0^1 (x + y)\, dy = x \int_0^1 1\, dy + \int_0^1 y\, dy = x(1) + \left.\frac{y^2}{2}\right|_0^1 = x + \frac{1}{2}$$

Now, we integrate with respect to x:

$$\int_0^1 \left(x + \frac{1}{2}\right) dx = \int_0^1 x\, dx + \frac{1}{2}\int_0^1 1\, dx = \left.\frac{x^2}{2}\right|_0^1 + \frac{1}{2}(1) = \frac{1}{2} + \frac{1}{2} = 1$$

Thus, the total volume under the surface $f(x, y) = x + y$ over the unit square is 1.

Linearity in Multivariate Summation and Integration

Just as with single-variable cases, summation and integration over multivariate functions respect linearity. This means that constants can be factored out, and we can distribute sums and integrals over sums of functions:

$$\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} [f(x, y) + g(x, y)] = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} f(x, y) + \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} g(x, y)$$

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} [f(x, y) + g(x, y)]\, dy\, dx = \int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x, y)\, dy\, dx + \int_{a_1}^{b_1} \int_{a_2}^{b_2} g(x, y)\, dy\, dx$$

This property is useful when working with complex functions, as it allows us to break down problems into simpler parts that can be summed or integrated individually.

2.6 Derivatives

The *derivative* of a function describes how the function changes locally. For a function $f(x)$, the derivative is commonly denoted either as $f'(x)$ or $\frac{df}{dx}(x)$, both of which express how $f(x)$ changes with respect to x. The derivative itself is also a function of x, representing the slope of the tangent line to the curve of $f(x)$ at any point.

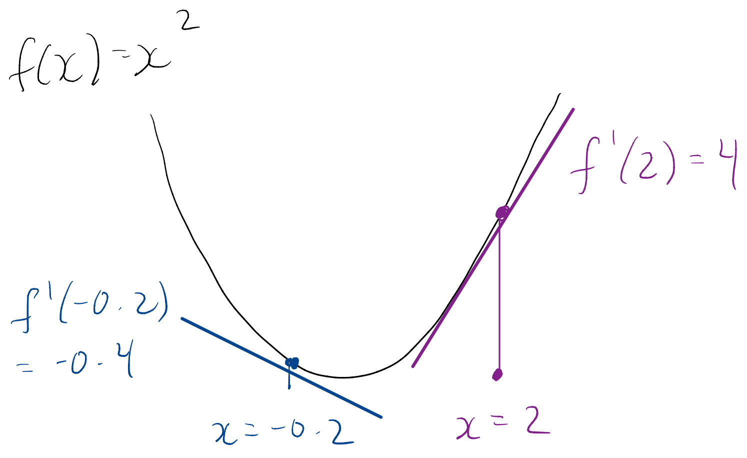


Figure 2.2: A plot of the function $f(x) = x^2$ and its derivative $f'(x) = 2x$ evaluated at -0.2 and 2. The derivative gives the slope of the line tangent to the function, at the given point. The sign and magnitude of this slope reflects how quickly the function is increasing or decreasing at that point.

For example, for the function $f(x) = x^2$, the derivative is $f'(x) = 2x$. Evaluating this derivative at $x = 2$, we find that $f'(2) = 4$, indicating that the function is increasing at this point. Since $f'(2) > 0$, the function is increasing as we increase x. More generally, because $f'(x) = 2x$, we observe that the rate of increase of the function grows as x increases.

If instead we evaluate the derivative at $x = -0.2$, we get $f'(-0.2) = -0.4$. In this case, since $f'(-0.2) < 0$, the function is decreasing at this point, although it decreases at a slower rate due to the smaller magnitude of the derivative. Near $x = 0$, the derivative approaches zero, meaning the function is flatter and changes more slowly in this region.

It is important to note that we write $\frac{df}{dx}(2)$, not $\frac{df(2)}{dx}$, because the derivative function $f' = \frac{df}{dx}$ is computed first, and only then is it evaluated at a specific point, such as $x = 2$. The derivative at different points for this function is illustrated in Figure 2.2.

2.6.1 Useful Derivative Rules

There are several key derivative rules that are commonly used and important to remember:

$$\begin{aligned}
&f(x) = x^a, && f'(x) = \frac{df}{dx}(x) = ax^{a-1} \\
&f(x) = \exp(x), && f'(x) = \frac{df}{dx}(x) = \exp(x) \\
&f(x) = \ln(x), && f'(x) = \frac{df}{dx}(x) = \frac{1}{x} \\
&f(x) = g(h(x)), \quad u = h(x) && f'(x) = \frac{df}{dx}(x) = \frac{dg}{du}\frac{du}{dx}(x) = g'(u)h'(x) \quad \triangleright \text{Chain rule} \quad (2.2) \\
&f(x) = g(x)h(x), && f'(x) = \frac{df}{dx}(x) = g'(x)h(x) + g(x)h'(x) \quad \triangleright \text{Product rule}
\end{aligned} \tag{2.3}$$

Example 2.9: For a more concrete example, consider the function $f(x) = \exp(x^2)$. We can apply the chain rule by setting $u = x^2, g(u) = \exp(u), h(x) = x^2$, and then differentiating as follows:

$$f'(x) = g'(u)h'(x) = \exp(u) \cdot 2x = 2x \exp(x^2)$$

Thus, the derivative of $f(x) = \exp(x^2)$ is $f'(x) = 2x \exp(x^2)$. □

2.6.2 Linearity and Derivative Properties

As with summation and integration, derivatives obey several useful properties, particularly linearity. The derivative of a sum of functions is the sum of the derivatives, and constants can be factored out:

$$\frac{d}{dx}[af(x) + bg(x)] = a\frac{df}{dx} + b\frac{dg}{dx}$$

These properties, along with the chain rule, will be used throughout the notes.

Exercise 2.15: What is the derivative of $f(x) = \exp(-x^3)$? □

2.6.3 Partial Derivatives and Gradients

When dealing with multivariate functions (i.e., functions that depend on more than one variable) we use *partial derivatives* to describe how the function changes with respect to one variable, while keeping the other variables fixed.

For a function $f(x_1, x_2, \ldots, x_n)$, the partial derivative with respect to x_i is denoted as $\frac{\partial f}{\partial x_i}$ or f'_{x_i}. This tells us the rate of change of f as x_i changes, while all other variables remain constant.

Example 2.10: Consider the function:

$$f(x_1, x_2) = 2x_1 + 3x_2^2$$

To compute the partial derivative with respect to x_1, we treat x_2 as a constant:

$$\frac{\partial f}{\partial x_1} = \frac{\partial}{\partial x_1}(2x_1 + 3x_2^2) = 2$$

Similarly, the partial derivative with respect to x_2 is:

$$\frac{\partial f}{\partial x_2} = \frac{\partial}{\partial x_2}(2x_1 + 3x_2^2) = 6x_2$$

□

Example 2.11: Now, consider a linear function in d variables, often encountered in machine learning:

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_d) = x_1w_1 + x_2w_2 + \cdots + x_dw_d = \sum_{i=1}^{d} x_iw_i = \mathbf{x}^\top \mathbf{w}$$

Here, $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{w} \in \mathbb{R}^d$ are vectors and since f does not depend on $\mathbf{w}$, it is assumed to be fixed.

The partial derivative of this function with respect to x_i is given by:

$$\frac{\partial f}{\partial x_i} = w_i$$

This means that the rate of change of f with respect to x_i is simply the corresponding weight w_i. □

Example 2.12: Consider the functions:

$$g(z, y) = (z - y)^2 \quad \text{and} \quad f(x) = \exp(x)$$

we are interested in computing the partial derivative of the composed function $g(f(x), y) = (\exp(x) - y)^2$ with respect to x.

To find the partial derivative $\frac{\partial}{\partial x} g(f(x), y)$, we apply the chain rule. Let's compute each part step by step.

1. **Compute $\frac{\partial g}{\partial z}$:**

$$g(z, y) = (z - y)^2 \implies \frac{\partial g}{\partial z} = 2(z - y)$$

2. **Compute $\frac{dz}{dx}$ where $z = f(x) = \exp(x)$:**

$$\frac{dz}{dx} = \frac{d}{dx} \exp(x) = \exp(x)$$

3. **Apply the chain rule:**

$$\frac{\partial}{\partial x} g(f(x), y) = \frac{\partial g}{\partial z} \cdot \frac{dz}{dx} = 2(\exp(x) - y) \cdot \exp(x) = 2\exp(2x) - 2y\exp(x)$$

□

Gradients

In addition to individual partial derivatives, we often work with the *gradient*, which is the vector of all partial derivatives of a function. The gradient of a function $f(\mathbf{x})$ is denoted as $\nabla f(\mathbf{x})$.

For a function $f(x_1, x_2, \ldots, x_d)$, the gradient is defined as:

$$\nabla f(\mathbf{x}) = \Big(\frac{\partial f}{\partial x_1}(\mathbf{x}), \frac{\partial f}{\partial x_2}(\mathbf{x}), \ldots, \frac{\partial f}{\partial x_d}(\mathbf{x}) \Big)^\top$$

Using our previous example $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$, the gradient is:

$$\nabla f(\mathbf{x}) = (w_1, w_2, \ldots, w_d)^\top = \mathbf{w}$$

Thus, the gradient of the linear function $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$ is simply $\mathbf{w}$.

The gradient will be particularly important in learning a good predictor function, where it will help us determine how to adjust our predictor function in order optimize for a particular objective.

Chapter 3

Probability

Probability will help us in formally defining the supervised learning setting and reason about the quality of our learner. Recall that in Chapter 1 we informally defined supervised learning as learning from a randomly sampled batch of labeled data. In Chapter 2 we defined the dataset as a tuple of n feature-label pairs. This dataset satisfied the property of being a batch of labeled data; however, we did not formally define what it means for a dataset to be randomly sampled. By the end of this chapter you should be able to understand that:

1. The dataset $D = ((\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)) \in (\mathcal{X} \times \mathcal{Y})^n$ is a random variable, where the features-label pairs $(\boldsymbol{X}_i, Y_i)$ are sampled idenpendently from the same distribution $\mathbb{P}_{\boldsymbol{X},Y}$ for all $i \in \{1, \ldots, n\}$.

2. The learner $\mathcal{A}(D)$ is a random variable, since it is a function of the random variable D.

3. A predictor $f(\boldsymbol{X})$ is a random variable, since it is a function of the random variable $\boldsymbol{X}$ representing a feature vector.

The probability tools covered in this chapter will also be useful to define the objective of learning in Chapter 4, and when we discuss how to evaluate the quality of our learner in Chapter 7. After this chapter we will have almost finished defining all the parts in the supervised learning setting. What will be left is to define the objective of the learner, which we do next in Chapter 4.

3.1 Probability Theory and Random Variables

Probability theory is as a branch of mathematics that deals with measuring the likelihood of events. It is based on the more foundational field of measure theory. When first learning about probability in an undergraduate class the term measure theory is often not even mentioned as most of the concepts can be understood without it. In these notes we will follow the same approach and not delve into measure theory. However, at times it will cause us to be a bit hand-wavy, and we will try to point out (often with a footnote) when we are doing so[1].

At the heart of probability theory is the concept of an *experiment*. An experiment can be the process of tossing a coin, rolling a die. When carried out, each experiment has an *outcome*, which is an element drawn from a set of predefined options, called the *outcome space*. The outcome of flipping a coin is either tails or heads, and the roll of a die is a number between one and six. Formally the outcome space is represented as a set of outcomes. Each

[1]For a more detailed coverage of probability we refer the reader to [**?**].

outcome can be represented as any object you would like, for instance the outcome of getting heads on a coin can be represented as a the word "Heads". However, we will usually represent outcomes as numbers since we will want to do mathematical operations with them. For example, we might represent the outcome of getting heads on a coin as 1 and the outcome of getting tails as 0. Then, the outcome space of a coin flip is $\{0,1\}$. Similarly we can define the outcome space of a die roll as $\{1,2,3,4,5,6\}$, where each number represents the outcome of the die roll. The outcomes space can also be an uncountably infinite set, such as $[0,\infty)$, which could represent the age of a randomly selected house.

An *event* is a subset of the outcome space[2]. For example, we might want to know the probability of seeing a 1 or a 3 when rolling a die, which would be represented as the event $\{1,3\} \subset \{1,2,3,4,5,6\}$. Another example is $[0,10] \subset [0,\infty)$, which can represent the event that the age of a house is between 0 and 10 years.

What we are interested in is the probability of an event occurring. This probability is defined by the *probaiblity distribution* function (sometimes just called a distribution) which will be denoted as $\mathbb{P}$. A probability distribution is a function that takes as input an event and outputs a number between 0 and 1. The value output by the distribution is the probability of the event occurring. For instance, if the event is getting a heads or tails in a coin flip (i.e. the event is $\{0,1\}$), then the probability of this event is $\mathbb{P}(\{0,1\}) = 1$. A probability of 1 means you will either get a heads or a tails in a coin flip 100% of the time. Naturally, the probability distribution must satisfy some properties that align with our intuition of what probability is. Informally, if the outcome space is $\mathcal{Z}$, then the probability distribution must satisfy the following properties[3]:

1. $P(\mathcal{Z}) = 1$
2. $P(A \cup B) = P(A) + P(B)$ if A and B are disjoint events (i.e. $A \cap B = \varnothing$).

The first property states that the probability of the entire outcome space is 1, which makes sense since the outcome of the experiment must be one of the outcomes in the outcome space. The second property states that the probability of either event A or event B occurring is the sum of the probabilities of each event occurring, if the events do not share any outcomes (i.e. disjoint).

Notice that these properties imply two useful results. The first is that for any event $\mathcal{E}$, it holds that $P(\mathcal{E}^c) = 1 - P(\mathcal{E})$. In words this means that the probability of an event not occurring is 1 minus the probability of the event occurring. To see why this holds notice that by the second and first property $\mathbb{P}(\mathcal{E}) + \mathbb{P}(\mathcal{E}^c) = \mathbb{P}(\mathcal{E} \cup \mathcal{E}^c) = \mathbb{P}(\mathcal{Z}) = 1$, since $\mathcal{E} \cap \mathcal{E}^c = \varnothing, \mathcal{E} \cup \mathcal{E}^c = \mathcal{Z}$ by the definition of the complement. Then rearranging we get $\mathbb{P}(\mathcal{E}^c) = 1 - \mathbb{P}(\mathcal{E})$. The second useful result is that $P(\varnothing) = 0$. This is because the empty set is the complement of the entire outcome space, and so by the first result and the first property we have $P(\varnothing) = 1 - P(\mathcal{Z}) = 1 - 1 = 0$.

[2]An event is not just any subset of the outcome space, it must also belong to something called the event space (or σ-algebra). If the outcome space is countable, then the event space can be defined to contain all subsets of the outcome space (i.e. the power set of the outcome space). While when the outcome space is uncountable the event space cannot contain all the subsets of the outcome space, and something called the Borel σ-algebra is chosen, which still contains any event you will ever want to speak about in this course. For an example of an event not in the Borel σ-algebra search the Cantor set. These details are covered in measure theory and discussed in more detail in Appendix A, but will be unnecessary for these notes.

[3]The second property actually needs to holds for countable unions of disjoint events. The formal definition of what a probability distribution needs to satisfy can be found in Appendix A.

To get a feeling for how probability distributions work, consider the following examples.

Example 3.1: Consider the outcome space $\mathcal{Y} = \{0, 1\}$ which represents the outcome of a fair coin flip. Since it is a fair coin the probability of getting a heads is 0.5 and the probability of getting a tails is 0.5. Formally, the probability distribution evaluated at the event $\{0\}$ representing tails is $\mathbb{P}(\{0\}) = 0.5$ and evaluated at the event $\{1\}$ representing heads is $\mathbb{P}(\{1\}) = 0.5$. Applying the first property we have $\mathbb{P}(\mathcal{Y}) = 1$, which makes sense since the coin flip must result in either heads or tails. Applying the second useful result we have $\mathbb{P}(\varnothing) = 0$. This describes the probability distribution for all possible events in the outcome space. $\square$

Example 3.2: Now, consider the outcome space $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$ which represents the outcome of a fair six-sided die roll. Since it is a fair die the probability of getting any number is $1/6$. Formally, the probability distribution evaluated at the event $\{1\}$ representing rolling a 1 is $\mathbb{P}(\{1\}) = 1/6$ and evaluated at the event $\{2\}$ representing rolling a 2 is $\mathbb{P}(\{2\}) = 1/6$, and so on. We also have $\mathbb{P}(\mathcal{X}) = 1$ and $\mathbb{P}(\varnothing) = 0$. However, there are many more events in this outcome space than the previous example, and to define the probability distribution function we must specify the probability of each event. You might notice that we can make use of second property to define the probability of any event in the outcome space. In particular, the probability of any event $\mathcal{E} \subset \mathcal{X}$ is the sum of the probabilities of the events containing each of the outcomes in $\mathcal{E}$. For example, if $\mathcal{E} = 1, 3, 5$ then $\mathbb{P}(\mathcal{E}) = \mathbb{P}(\{1\}) + \mathbb{P}(\{3\}) + \mathbb{P}(\{5\}) = 1/6 + 1/6 + 1/6 = 1/2$. This trick of calculating the probability of an event by summing the probabilities of the events containing each of the outcomes in the event will be discussed further in the next section. $\square$

You might notice that the notation of a distribution taking as input an event is intuitively a bit misleading. For instance, from the previous dice roll example when we write $\mathbb{P}(\{1, 3, 5\})$ we do not mean to say that the probability of getting the numbers 1, 3, or 5 all at the same time is $1/2$. We only ever get one outcome from the dice roll. This outcome is of course random, and so when we write $\mathbb{P}(\{1, 3, 5\})$ we mean the probability that the random outcome produced by the dice roll is either 1, 3, or 5. As such, we use what is called a *random variable* to represent the random outcome of an experiment. A random variable is always associated with a probability distribution which describes the probability of the random variable taking a value in a given event. For instance, if we let X be the random variable representing the outcome of a die roll, then we write $\mathbb{P}(X \in \{1, 3, 5\}) = \mathbb{P}(\{1, 3, 5\})$ to mean the probability that the random variable X takes on the value 1, 3, or 5.

For the purposes of this course we will think of random variables as elements of the outcome space, and will write things like $X \in \{1, 2, 3, 4, 5, 6\}$[4]. We will use capital letters to denote random variables, and the value of the random variable will be denoted by the lowercase version of the letter. Some shorthand notation is often used to represent the probability of a random variable taking on a value in an event. Let X be a random variable representing the outcome of a die roll, and its associated probability distribution be $\mathbb{P}$. We have already seen that the probability of the random variable X being an odd number is written as $\mathbb{P}(X \in \{1, 3, 5\})$ which is defined as $\mathbb{P}(\{1, 3, 5\})$. If we are interested in the

[4]A random variable is actually a function (satisfying certain properties) from one outcome space to another outcome space. Again, this is covered in measure theory and some details are explained in Appendix A.

probability of X taking a value greater than 4, we can write $\mathbb{P}(X > 4) = \mathbb{P}(X \in \{5, 6\}) = \mathbb{P}(\{5, 6\})$. Or, if we are interested in the probability of X taking the value 4, then we can write $\mathbb{P}(X = 4) = \mathbb{P}(X \in \{4\}) = \mathbb{P}(\{4\})$. This last case will be especially important to remember since writing $X = 4$ would seem to imply that the probability distribution function can take as input an outcome instead of an event. This is not the case, since $\mathbb{P}(X = 4)$ is defined as $\mathbb{P}(\{4\})$, which makes it clear that $\mathbb{P}$ always takes as input events and never outcomes. Of course $X \in \{4\}$ is to be interpreted as the event that the random variable X is the outcome of 4, however, mathematically it is important because $\mathbb{P}$ is defined to only take as input events.

Another note on notation is that $X \sim \mathbb{P}$ is often used to denote that the random variable X has the probability distribution $\mathbb{P}$, or sometimes read as X is sampled according to $\mathbb{P}$.

In the next section we will discuss a convenient way to define a probability distribution when the outcome space is large or even infinite. However, to do this it will be useful to distinguish between two types of random variables: discrete and continuous. A random variable is discrete if it takes values from:

- a countable outcome space, or
- an uncountable outcome space, but there is a countable event that has a probability of 1.

Some examples of the first case, when the outcome space is countable, are the random variables representing the outcome of a die roll or a coin flip. To give some intuition for the second case we give a more detailed examples.

Example 3.3: Suppose we are again working with a fair six-sided die roll. The outcome space is $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$, and the probability distribution $\mathbb{P}$ satisfies that $\mathbb{P}(X = 1) = \cdots = \mathbb{P}(X = 6) = 1/6$ since the die is fair. If we let $X \in \mathcal{X}$ be the random variable representing the outcome of the die roll, then clearly X is a discrete random variable, since outcome space $\mathcal{X}$ is finite and thus countable.

Now, suppose we change the outcome space to $\mathcal{X} = \mathbb{R}$. Here we should think of the newly added outcomes as you would expect (ex: $-5 \in \mathbb{R}$ is the outcome the die roll was -5, which is of course impossible, but we can speak about it). We need to define the probability distribution $\mathbb{P}$ now. Naturally, we will keep $\mathbb{P}(X = 1) = \cdots = \mathbb{P}(X = 6) = 1/6$, which means $\mathbb{P}(X \in \{1, 2, 3, 4, 5, 6\}) = \mathbb{P}(X = 1) + \cdots + \mathbb{P}(X = 6) = 1$. This implies that $\mathbb{P}(X \in \{1, 2, 3, 4, 5, 6\}^c) = 1 - \mathbb{P}(X \in \{1, 2, 3, 4, 5, 6\}) = 0$ (i.e. the probability of the die roll being any number other than 1, 2, 3, 4, 5, or 6 is 0). Notice that the event {1, 2, 3, 4, 5, 6} is countable since it has a finite cardinality $|\{1, 2, 3, 4, 5, 6\}| = 6$. Also, notice that $\mathbb{P}(\{1, 2, 3, 4, 5, 6\} = 1$. Thus, the random variable X is still discrete, even though the outcome space is uncountable. $\square$

The above example demonstrates a general extension rule that you can always take a random variable defined on a countable outcome space and extend it to an uncountable outcome space, by setting the probability of the event containing all the new outcomes to 0. But, you can still reason about the probabilities as if the outcome space was countable, since the probabilities of events from the original countable outcome space remain the same. This extension can be used when we are working with features in machine learning that belong to a countable set. For example, we can use a random variable to represent the number of rooms feature of a house. The number of rooms feature is an element of the natural numbers, which is countable. However, for implementation reasons (which we will see in later sections) we will often want to represent the feature as 1-dimensional vector which is just a real number. Using the extension rule we can have a feature represented as a real number, but still have the probability of the feature taking a specific value (ex: 3 rooms) be non-zero. This will be in stark contrast to continuous random variables, which we define next.

A random variable is continuous if it takes values from:

- an uncountable outcome space, and the probability of any single outcome is 0.

The intuition behind the probability of a single outcome being 0 will be discussed in the next section when we define probability density functions. For now, we will just give an example of a continuous random variable.

Example 3.4: Let $Z \in [0, \infty)$ be a random variable representing the age of a house. Here the outcome space $[0, \infty)$ is uncountable. Z is a continuous random variable if the probability of the houses age $z \in [0, \infty)$ is zero. Formally, this means that $\mathbb{P}(Z = z) = 0$ for all $z \in [0, \infty)$. Also, by the first property of the probability distribution we have $\mathbb{P}(Z \in [0, \infty)) = 1$[5]. It is also valid if the probability of the house being between 0 and 10 years is $\mathbb{P}(Z \in [0, 10]) = 0.5$, since $[0, 10]$ is not an event containing only a single number, but instead is uncountably infinite. □

3.2 Defining Distributions

In this section we introduce some helpful functions, called probability mass functions and probability density functions. These functions allow us to define the probability distribution over any outcome space in a simple way. They are also very useful when calculating the probabilities of random variables taking on values in events. It is convenient to separately consider discrete and continuous random variables.

3.2.1 Probability mass functions for discrete random variables

Let $\mathcal{X}$ be a countable outcome space. A function $p : \mathcal{X} \to [0, 1]$ is called a *probability mass function* (pmf) if

$$\sum_{x \in \mathcal{X}} p(x) = 1.$$

The probability of any event $\mathcal{E} \subset \mathcal{X}$ is defined as

$$\mathbb{P}(X \in \mathcal{E}) \overset{\text{def}}{=} \sum_{x \in \mathcal{E}} p(x).$$

[5]At this point a careful reader might wonder how this is possible. If the second property of a probability distribution indicates that the probability of the union of disjoint events is the sum of their probabilies. And, if you write $[0, \infty)$ as the union of all the numbers in $[0, \infty)$, wouldn't the probability of $[0, \infty)$ just be the sum $\sum_{z \in [0, \infty)} \mathbb{P}(Z = z) = 0$? Alas, the devil is in the details. It turns out that the second property of a probability distribution only holds for countable unions of disjoint events, while the set $[0, \infty)$ cannot be written as a countable union of numbers. The details of this, again, can be found in a course on measure theory, and are not needed for these notes.

For discrete random variables notice that $\mathbb{P}(X = x) = p(x)$ for each outcome $x \in \mathcal{X}$, by the above definition. The conditions on the pmf p ensure that we get valid probabilities, including that $\mathbb{P}(X \in \mathcal{E}) \in [0, 1]$ and that $\mathbb{P}(X \in \mathcal{X}) = 1$.

Exercise 3.1: Prove that for any pmf p, defined above, we have $\mathbb{P}(X \in \mathcal{X}) = 1$. □

Example 3.5: Consider a roll of a fair six-sided die (i.e. $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$), then what is the probability that the outcome is a number greater than 4?

First, because the die is fair, we know that $p(x) = \frac{1}{6}$ for all $x \in \mathcal{X}$. Now, let $\mathcal{E} \subset \mathcal{X}$ be the event that the outcome is greater than 4; i.e., $\mathcal{E} = \{5, 6\}$. Thus,

$$\mathbb{P}(X \in \mathcal{E}) = \sum_{x \in \mathcal{E}} p(x) = \frac{1}{3}.$$

□

To specify $\mathbb{P}$, therefore, we need to determine how to specify the pmf, (i.e. the probability of each discrete outcome). The pmf is often specified as a table of probability values. For example, to model the probability of a birthday for each day in the year, one could have a table of 365 values between zero and one, as long as the probabilities sum to 1. These probabilities could be computed from data about individuals birthdays, using counts for each day and normalizing by the total number of people in the population to estimate the probability of seeing a birthday on a given day. Such a table of values is very flexible, allowing precise probability values to be specified for each outcome. There are, however, a few useful named pmfs that have a (more restricted) functional form.

The *Bernoulli distribution* derives from the concept of a Bernoulli trial, an experiment that has two possible outcomes: success and failure. In a Bernoulli trial, a success occurs with probability $\alpha \in [0, 1]$ and, thus, failure occurs with probability $1 - \alpha$. A toss of a coin (heads/tails), a basketball game (win/loss), or a roll of a die (even/odd) can all be seen as Bernoulli trials. The outcome space $\mathcal{X}$ consists of two elements and we define the probability of one of them as α. More specifically, $\mathcal{X} = \{0, 1\}$, where 0 represents a failure and 1 represents a success. The pmf is:

$$p(x) = \begin{cases} \alpha & x = \text{success} \\ 1 - \alpha & x = \text{failure} \end{cases}$$

where $\alpha \in [0, 1]$ is a parameter. We can compactly write the Bernoulli distribution as

$$p(x) = \alpha^x (1 - \alpha)^{1-x} \tag{3.1}$$

for $x \in \mathcal{X}$. The Bernoulli distribution is often written Bernoulli(α). As we will see, a common setting where we use the Bernoulli is for binary classification, say where we try to predict whether a patient has the flu (outcome 0) or does not have the flu (outcome 1).

The *discrete uniform distribution* for discrete sample spaces is defined over a finite set of outcomes each of which is equally likely to occur. Let $\mathcal{X} = \{1, \ldots, n\}$; then for all $x \in \mathcal{X}$

$$p(x) = \frac{1}{n}.$$

The uniform distribution has a parameter $n \in \mathbb{N}$ that defines the size of the sample space and the pmf. We refer to this distribution as Uniform(n). We will see later that the uniform distribution can also be defined for continuous random variables.

3.2.2 Probability density functions for continuous random variables

The treatment of continuous probability spaces is analogous to that of discrete spaces, with probability density functions (pdfs) replacing probability mass functions and integrals replacing sums. In defining pdfs, however, we will not be able to use tables of values, and will be restricted to functional forms. The main reason for this difference stems from the fact that it no longer makes sense to measure the probability of an event containing a single outcome.

Consider the age of a randomly selected house which we represent as a number in $[0, \infty)$, which recall is an uncountable outcome space. It would not make a lot of sense to ask the probability that the age is exactly 3.14159625 years; realistically, the probability of such a precise event is vanishingly small. In fact, the probability of seeing precisely that age is zero, because the event $\{3.14159625\}$ is an event containing a single outcome. Instead, we will have to consider the probabilities of intervals, like $[3, 4]$ or $[3.14, 3.15]$.

Let $\mathcal{X}$ be an uncountable outcome space. A function $p : \mathcal{X} \to [0, \infty)$ is called a *probability density function* (pdf) if

$$\int_{\mathcal{X}} p(x)dx = 1$$

where the integral is over the whole space $\mathcal{X}$. For example, if $\mathcal{X} = [a, b]$, then $\int_{\mathcal{X}} p(x)dx = \int_a^b p(x)dx$. The probability of an event $\mathcal{E} \subset \mathcal{X}$ is defined as

$$\mathbb{P}(X \in \mathcal{E}) \overset{\mathsf{def}}{=} \int_{\mathcal{E}} p(x)dx.$$

Notice that the definition of the pdf is not restricted to having a range $[0, 1]$, but rather to $[0, \infty)$. For pmfs, the probability of an event with a single outcome $\{x\}$ is the value of the pmf at the sample point x (i.e. $\mathbb{P}(X = x) = \mathbb{P}(X \in \{x\}) = p(x)$). Since probability distributions $\mathbb{P}$ are restricted to the range $[0, 1]$, this implies pmfs must also be restricted to that range. In contrast, the value of a pdf at point x is not a probability; it can actually be greater than 1. Rather, $p(x)$ reflects the density at x. While the density itself is not a probability, it indicates how likely it is to see points near x when we integrate over a small region around x. Also, the probability of an event containing a single outcome is zero, because the integral over a single point is zero.

A natural confusion is how p can integrate to 1, but actually have values larger than 1. The reason for this is that $p(x)$ can be (much) larger than 1, as long as its only for a very small interval. Consider the small interval $\mathcal{E} = [x, x + \Delta x]$, with probability

$$\mathbb{P}(\mathcal{E}) = \int_x^{x+\Delta x} p(x)dx \approx p(x)\Delta x.$$

A potentially large value of the density function is compensated for by the small interval Δx to result in a number between 0 and 1. So, even if $p(x)$ is a million, and the density of points in the small interval or ball around x is large, the probability of an event must still be ≤ 1. The density does indicate that there is high likelihood around that point. By having a huge density around x, this suggests that the density for other points is zero or near zero and that the pdf is extremely peaked around x.

Figure 3.1: Selection of a random number (x) from the unit interval $[0, 1]$.

Unlike pmfs, we cannot so easily define pdfs p to flexibly provide specific probabilities for each outcome with a table of probabilities. Rather, for pdfs, we will usually use a known pdf that satisfies the required properties. Further, unlike the discrete case, we will never write $\mathbb{P}(X = x)$, because that would be zero. Rather, we will typically write $\mathbb{P}(X \in A)$, such as by writing $\mathbb{P}(X \in [0.1, 0.2])$ or $\mathbb{P}(X \leq 5)$. We highlight three pdfs here, that will be used throughout these notes.

The *continuous uniform distribution* is defined by an equal value of a probability density function over a finite interval in $\mathbb{R}$. Thus, for $\mathcal{X} = [a, b]$ the uniform probability density function $\forall x \in [a, b]$ is defined as

$$p(x) \overset{\text{def}}{=} \frac{1}{b-a}.$$

The continuous uniform distribution has two parameters $a, b \in \mathbb{R}$. We will refer to this distribution as $\text{Uniform}(a, b)$.

The *Gaussian distribution* or normal distribution is one of the most frequently used probability distributions. It has outcome space $\mathcal{X} = \mathbb{R}$, and two parameters, $\mu \in \mathbb{R}$ and $\sigma > 0$. The pdf is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

As we will discuss next, for a random variable that is Gaussian distributed, the parameter μ is the mean or expected value and σ^2 is the variance. We will refer to this distribution as $\text{Gaussian}(\mu, \sigma^2)$ or $\mathcal{N}(\mu, \sigma^2)$. When the mean is zero, and the variance is 1 (unit variance), this Gaussian is called the *standard normal.* This specific Gaussian has a name because it is so frequently used.

The *Laplace distribution* is similar to the Gaussian, but is more peaked around the mean. It also has outcome space $\mathcal{X} = \mathbb{R}$, and two parameters $\mu \in \mathbb{R}$ and $b > 0$. The pdf is

$$p(x) = \frac{1}{2b} e^{-\frac{1}{b}|x-\mu|}$$

We will refer to this distribution as $\text{Laplace}(\mu, b)$.

Example 3.6: Consider selecting a number between 0 and 1 uniformly randomly (shown in Fig. 3.1). What is the probability that the number is greater than or equal to $\frac{3}{4}$ or less than and equal to $\frac{1}{4}$?

We know that $\mathcal{X} = [0, 1]$. The distribution is defined by the uniform pdf, $p(x) = \frac{1}{b-a} = 1$ where $a = 0, b = 1$ define the interval for the outcome space. We define the event of interest as $\mathcal{E} = \left[0, \frac{1}{4}\right] \cup \left[\frac{3}{4}, 1\right]$ and calculate its probability as

$$\begin{aligned}
\mathbb{P}(\mathcal{E}) &= \int_0^{1/4} p(x)dx + \int_{3/4}^{1} p(x)dx && \triangleright\ p(x) = 1 \\
&= \left(\frac{1}{4} - 0\right) + \left(1 - \frac{3}{4}\right) \\
&= \frac{1}{2}.
\end{aligned}$$

Note that since the probability of an event containing a single outcome is 0, there is no difference in integration if we consider open or closed intervals (since for instance $[0, \frac{1}{4}] = [0, \frac{1}{4}) \cup \{\frac{1}{4}\}$). Therefore, the probability of $[0, \frac{1}{4}) \cup [\frac{3}{4}, 1]$ would still be $\frac{1}{2}$. □

3.3 Multivariate Random Variables

The examples so far have dealt with, what are sometimes called, univariate random variables, because they only have one random variable. Sometimes it is useful to talk about the probability of multiple random variables, which is what multi-variate random variables allow us to do. A multivariate random variable is a tuple of random variables. Much of the above development extends to multivariate random variables because the definition of outcome spaces and probability distributions was introduced in a general way. In this section, we discuss several new notions that only arise when there are multiple random variables, including joint distributions, conditional distributions, marginal distributions and dependence between random variables.

To build some intuition, let us start with two random variables X and Y with outcome spaces $\mathcal{X}$ and $\mathcal{Y}$ respectively. The tuple containing the two random variables is called a multi-variate random variable, and we will denote it as $Z = (X, Y)$. The outcome space of Z is $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, which is the Cartesian product of the outcome spaces of X and Y. Since Z is a random variable, it has a probability distribution $\mathbb{P}$, which is called the *joint distribution* of X and Y.

Example 3.7: Consider the random variables $X \in \{0, 1\}$ and $Y \in \{0, 1\}$ where X is the outcome of a coin flip and Y is the outcome of a different coin flip. Then $Z = (X, Y)$ is a multivariate random variable with outcome space $\mathcal{Z} = \{0, 1\} \times \{0, 1\}$. It represents the outcome of two coin flips. □

Example 3.8: Consider the random variables $X \in \mathbb{N}$ and $Y \in [0, \infty)$ where X represents the number of rooms in a house and Y is the age of a house. Then $Z = (X, Y)$ is a multivariate random variable with outcome space $\mathcal{Z} = \mathbb{N} \times [0, \infty)$. It represents the number of rooms and the age of a house. □

If X and Y are discrete, the random variable Z is also considered discrete. Since Z is a discrete random variable its distribution is defined through a *joint pmf* $p : \mathcal{Z} \to [0, 1]$. The probability of any event $\mathcal{E} \subset \mathcal{Z}$ is

$$\mathbb{P}(Z \in \mathcal{E}) = \sum_{z \in \mathcal{E}} p(z).$$

		Y	
		no arthritis	arthritis
X	young	$p(\text{young}, \text{no arthritis}) = 1/2$	$p(\text{young}, \text{arthritis}) = 1/100$
	old	$p(\text{old}, \text{no arthritis}) = 1/10$	$p(\text{old}, \text{arthritis}) = 39/100$

Table 3.1: The joint pmf p for random variables X and Y.

Example 3.9: Consider an example where $X \in \mathcal{X} = \{\text{young}, \text{old}\}$ and $Y \in \mathcal{Y} = \{\text{no arthritis}, \text{arthritis}\}$. Then $Z = (X, Y)$ is a multivariate random variable with outcome space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y} = \{(\text{young}, \text{no arthritis}), (\text{young}, \text{arthritis}), (\text{old}, \text{no arthritis}), (\text{old}, \text{arthritis})\}$. Suppose that the joint[6] pmf is[7]

$$p(x, y) = \begin{cases} 1/2 & \text{if } x = \text{young}, y = \text{no arthritis} \\ 1/100 & \text{if } x = \text{young}, y = \text{arthritis} \\ 1/10 & \text{if } x = \text{old}, y = \text{no arthritis} \\ 39/100 & \text{if } x = \text{old}, y = \text{arthritis}. \end{cases}$$

To build intuition about the joint pmf it can be useful to represent it with a table as shown in Table 3.1. Notice that the joint pmf sums to 1 over all possible outcomes, as it should (i.e. $\sum_{z \in \mathcal{Z}} p(z) = 1$). However, the rows do not sum to 1 nor do the columns.

Another useful way to visualize the joint pmf is to represent its value for each outcome $(x, y) \in \mathcal{Z}$ as an area in a two-dimensional square with unit area, as shown in Fig. 3.2. We can see that the area taken up by the outcome (young, no arthritis) is $1/2$, which is a much larger area than the area taken up by (young, arthritis) which is only $1/100$. The benefit of working with discrete random variables is that the pmf can be interpreted as the giving the probability of an outcome, since $\mathbb{P}(Z = z) = p(z)$ for any $z \in \mathcal{Z}$. Thus, the regions in Fig. 3.2 are exactly the probabilities of each outcome $(x, y) \in \mathcal{Z}$[8]. $\square$

If instead X and Y are continuous, the random variable Z is also considered continuous and its distribution is defined through a *joint pdf* $p : \mathcal{Z} \to [0, \infty)$. The probability of any event $\mathcal{E} \subset \mathcal{Z}$ is

$$\mathbb{P}(Z \in \mathcal{E}) \overset{\text{def}}{=} \int_{\mathcal{E}} p(z) dz.$$

There is a common notational convention that is used when dealing with multivariate random variables and the event $\mathcal{E} \subset \mathcal{Z}$ has a specific form. In particular, if $\mathcal{E} = \mathcal{E}_x \times \mathcal{E}_y$ where $\mathcal{E}_x \subset \mathcal{X}$ and $\mathcal{E}_y \subset \mathcal{Y}$, then we can write $\mathbb{P}(Z \in \mathcal{E}) = \mathbb{P}(X \in \mathcal{E}_x, Y \in \mathcal{E}_y)$. Intuitively, this means that $\mathcal{E}$ is the event that considers every possible pair of outcomes (x, y) where $x \in \mathcal{E}_x$ and $y \in \mathcal{E}_y$. The notation $\mathbb{P}(X \in \mathcal{E}_x, Y \in \mathcal{E}_y)$ should then be read as the probability that X is in $\mathcal{E}_x$ and Y is in $\mathcal{E}_y$. A useful property of $\mathcal{E}$ being of the form $\mathcal{E}_x \times \mathcal{E}_y$ is that the

[6]Pun intended.

[7]This is synthetic data and does not represent any real data.

[8]Remember that when we say "the probability of an outcome" we really mean the probability of the event that contains that outcome.

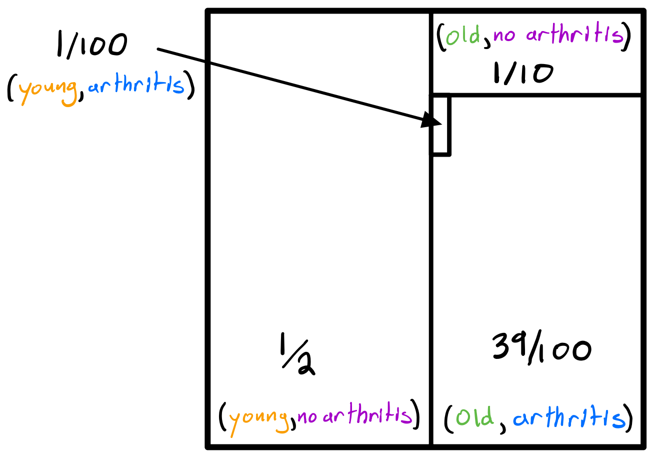


Figure 3.2: The area of each rectangle represents the value of the joint pmf $p(x, y)$ for each outcome $(x, y) \in \mathcal{Z}$.

probability of this event can be written as

$$\mathbb{P}(X \in \mathcal{E}_x, Y \in \mathcal{E}_y) = \mathbb{P}(Z \in \mathcal{E}) = \begin{cases} \sum_{z \in \mathcal{E}} p(z) = \sum_{x \in \mathcal{E}_x} \sum_{y \in \mathcal{E}_y} p(x, y) & \text{if } Z \text{ is discrete} \\ \int_{\mathcal{E}} p(z) dz = \int_{\mathcal{E}_x} \int_{\mathcal{E}_y} p(x, y) dy dx & \text{if } Z \text{ is continuous.} \end{cases}$$

In these notes we will almost always work with events $\mathcal{E}$ that are of the form $\mathcal{E}_x \times \mathcal{E}_y$.

Example 3.10: We continue with the previous arthritis example. Suppose we are interested in the probability that a person is young X = young and either has or does not have arthritis $Y \in \{\text{no arthritis}, \text{arthritis}\}$. Notice that this event is of the form $\mathcal{E} = \{\text{young}\} \times \{\text{no arthritis}, \text{arthritis}\} = \{(\text{young}, \text{no arthritis}), (\text{young}, \text{arthritis})\}$. Thus, to calculate the probability of this event we can write

$$\begin{aligned} \mathbb{P}(X = \text{young}, Y \in \{\text{no arthritis}, \text{arthritis}\}) &= \sum_{x \in \{\text{young}\}} \sum_{y \in \{\text{no arthritis}, \text{arthritis}\}} p(x, y) \\ &= p(\text{young}, \text{no arthritis}) + p(\text{young}, \text{arthritis}) \\ &= 1/2 + 1/100 = 51/100. \end{aligned}$$

□

So far we have only discussed multivariate random variables that are tuples of two random variables, but we can generalize this to tuples of any number of random variables and all of the above concepts extend in a straightforward way. In particular, if we have a multivariate random variable $X = (X_1, \ldots, X_d)$ with outcome space $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$ and pmf of pdf p (depending on if X is discrete or continuous), then the probability of some event $\mathcal{E} \subset \mathcal{X}$ is

$$\mathbb{P}(X \in \mathcal{E}) = \begin{cases} \sum_{x \in \mathcal{E}} p(x) & \text{if } X \text{ is discrete} \\ \int_{\mathcal{E}} p(x) dx & \text{if } X \text{ is continuous.} \end{cases}$$

If $\mathcal{E} = \mathcal{E}_1 \times \cdots \times \mathcal{E}_d$ where $\mathcal{E}_i \subset \mathcal{X}_i$ for all $i \in \{1, \dots, d\}$, then we can write $\mathbb{P}(X \in \mathcal{E}) = \mathbb{P}(X_1 \in \mathcal{E}_1, \dots, X_d \in \mathcal{E}_d)$ and the probability of event $\mathcal{E}$ is

$$\mathbb{P}(X_1 \in \mathcal{E}_1, \dots, X_d \in \mathcal{E}_d) = \begin{cases} \sum\limits_{x_1 \in \mathcal{E}_1} \cdots \sum\limits_{x_d \in \mathcal{E}_d} p(x_1, \dots, x_d) & \text{if } X \text{ is discrete} \\ \int\limits_{\mathcal{E}_1} \cdots \int\limits_{\mathcal{E}_d} p(x_1, \dots, x_d) dx_1 \cdots dx_d & \text{if } X \text{ is continuous.} \end{cases}$$

3.3.1 Marginal Distributions

Given a joint distribution over random variables, one would hope that we could extract more specific probabilities, like the distribution over just one of those variables, which is called the *marginal distribution.* As with all the probability distributions we have seen so far, the marginal distribution can be discrete or continuous and is defined through the marginal pmf or pdf respectively. Consider again the case where $Z = (X, Y) \in \mathcal{X} \times \mathcal{Y} = \mathcal{Z}$ is a multivariate random variable with joint pmf or pdf p (depending on if Z is discrete or continuous). The marginal pmf or pdf of X is defined for any $x \in \mathcal{X}$ as

$$p_X(x) \overset{\text{def}}{=} \begin{cases} \sum_{y \in \mathcal{Y}} p(x, y) & \text{if } X, Y \text{ are discrete} \\ \int_{\mathcal{Y}} p(x, y) dy & \text{if } X, Y \text{ are continuous.} \end{cases}$$

Roughly speaking, the marginal distribution of X is the distribution of X if we ignore the value of Y. Then, the marginal probability distribution of X is defined for any event $\mathcal{E}_X \subset \mathcal{X}$ as

$$\mathbb{P}_X(X \in \mathcal{E}_X) = \begin{cases} \sum_{x \in \mathcal{E}_X} p_X(x) & \text{if } X \text{ is discrete} \\ \int_{\mathcal{E}_X} p_X(x) dx & \text{if } X \text{ is continuous.} \end{cases}$$

The exact same definitions apply for the marginal distribution of Y.

Example 3.11: We continue with the arthritis example. Suppose we want to calculate the probability that a person is young, regardless of whether they have arthritis or not. To calculate this probability we can use the marginal distribution of X as follows

$$\begin{aligned} \mathbb{P}_X(X = \text{young}) &= p_X(\text{young}) \\ &= \sum_{y \in \mathcal{Y}} p(\text{young}, y) \\ &= p(\text{young}, \text{no arthritis}) + p(\text{young}, \text{arthritis}) \\ &= {}^1\!/_2 + {}^1\!/_{100} = {}^{51}\!/_{100}. \end{aligned}$$

The careful reader might notice that this is the same probability as in Example 3.10. This is not a coincidence. A young person either does or does not have arthritis, so summing up over these two possible cases factors out that variable. Notice that this is exactly what was done in Example 3.10, since the event for Y was the whole outcome space $\mathcal{E}_Y = \mathcal{Y}$.

To continue building our intuition we can again use the idea of areas as probabilities (that we introduced in Fig. 3.2) to visualize the marginal distribution of X. To get the probability of X = young we can sum up all the areas that correspond to the outcome

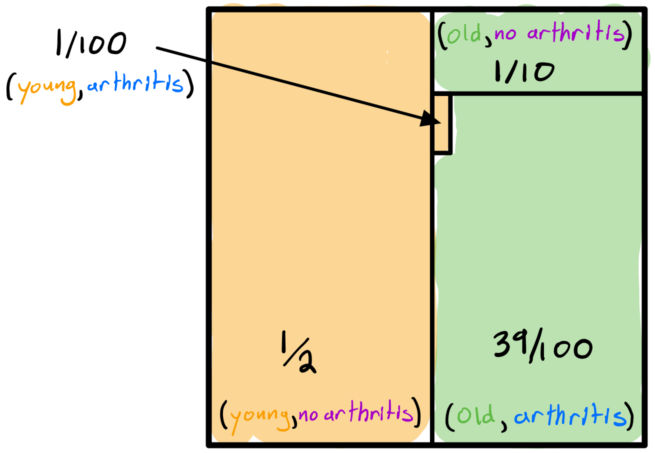


Figure 3.3: The orange area represents the value $p_X(\textit{young}) = {}^{51}\!/_{100}$*, while the green area represents the value* $p_X(\textit{old}) = {}^{49}\!/_{100}$*.*

(young, y) for all $y \in \mathcal{Y}$. This is represented as the orange region in the figure, and we can see that the area of this region is ${}^{51}\!/_{100}$, matching our calculation above. We can also use this approach to quickly calculate the probability of $X = \text{old}$, which is just the sum of all the areas corresponding to the outcome (old, y) for all $y \in \mathcal{Y}$. The area of this region is ${}^{49}\!/_{100}$. The calculation $\mathbb{P}(X = \text{old}) = 1 - \mathbb{P}(X = \text{young}) = {}^{49}\!/_{100}$ shows that our visual approach is indeed correct.

We could repeat the same process to calculate the marginal distribution of Y and get the probability of $Y = \text{no arthritis}$ or $Y = \text{arthritis}$. □

In general, if we have a multivariate random variable $X = (X_1, X_2, \ldots, X_d)$ with outcome space $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$ and joint pmf or pdf p (depending on if X is discrete or continuous), then the marginal pmf or pdf of X_i is defined for any $x_i \in \mathcal{X}_i, i \in \{1, \ldots, n\}$ as

$$p_{X_i}(x_i) \stackrel{\text{def}}{=} \begin{cases} \sum\limits_{x_1 \in \mathcal{X}_1} \cdots \sum\limits_{x_{i-1} \in \mathcal{X}_{i-1}} \sum\limits_{x_{i+1} \in \mathcal{X}_{i+1}} \cdots \sum\limits_{x_d \in \mathcal{X}_d} p(x_1, \ldots, x_d) & \text{if } X \text{ is discrete} \\ \int\limits_{\mathcal{X}_1} \cdots \int\limits_{\mathcal{X}_{i-1}} \int\limits_{\mathcal{X}_{i+1}} \cdots \int\limits_{\mathcal{X}_d} p(x_1, \ldots, x_d)\, dx_1 \ldots dx_{i-1} dx_{i+1} \ldots dx_d & \text{if } X \text{ is continuous.} \end{cases}$$

3.3.2 Conditional Probability and Independence

In many scenarios, we are interested in the probability of one random variable given that another random variable takes on a specific value. This leads us to the concept of a *conditional probability distribution*. Just as with joint and marginal distributions, conditional distributions are defined through the *conditional pmf* for discrete variables and the *conditional pdf* for continuous variables.

Suppose we have two random variables X and Y with a joint pmf or pdf $p(x, y)$ and marginal pmfs or pdfs $p_X(x)$ and $p_Y(y)$. For discrete or continuous random variables, the conditional pmf or pdf of Y given $X = x$ is calculated as

$$p_{Y|X}(y|x) \overset{\text{def}}{=} \frac{p(x,y)}{p_X(x)} \quad \text{for all } y \in \mathcal{Y} \text{ such that } p_X(x) > 0.$$

When we condition on $X = x$, the conditional pmf or pdf $p_{Y|X}(y|x)$ is a function solely of y, with x fixed. As before, the conditional distribution of Y given $X = x$ is defined for any event $\mathcal{E}_Y \subset \mathcal{Y}$ through the conditional pmf or pdf as

$$\mathbb{P}_{Y|X}(Y \in \mathcal{E}_Y | X = x) \overset{\text{def}}{=} \begin{cases} \sum_{y \in \mathcal{E}_Y} p_{Y|X}(y|x) & \text{if } Y \text{ is discrete} \\ \int_{\mathcal{E}_Y} p_{Y|X}(y|x) dy & \text{if } Y \text{ is continuous.} \end{cases}$$

All of the above also applies to the conditional distribution of X given $Y = y$.

Example 3.12: Continuing with the arthritis example, suppose we want to find the probability that a person has arthritis given that they are young.

First, recall the joint probabilities:

$$p(\text{young}, \text{no arthritis}) = 1/2, \quad p(\text{young}, \text{arthritis}) = 1/100.$$

We have already calculated the marginal pmf of X at $X =$ young in Example 3.11 to be

$$p_X(\text{young}) = 51/100.$$

Thus, the conditional probability of Y given $X =$ young is

$$\begin{aligned} \mathbb{P}_{Y|X}(Y = \text{arthritis}|X = \text{young}) &= p_{Y|X}(\text{arthritis}|\text{young}) \\ &= \frac{p(\text{young}, \text{arthritis})}{p_X(\text{young})} = \frac{1/100}{51/100} = \frac{1}{51}, \\ \mathbb{P}_{Y|X}(Y = \text{no arthritis}|X = \text{young}) &= p_{Y|X}(\text{no arthritis}|\text{young}) \\ &= \frac{p(\text{young}, \text{no arthritis})}{p_X(\text{young})} = \frac{1/2}{51/100} = \frac{50}{51}. \end{aligned}$$

We can visuallize this again using the idea of areas as probabilities, as shown in Fig. 3.4. The shape on the right represents all the area that remains when conditioning on $X =$ young. We can then see that for this shape, the area corresponding to $Y =$ arthritis is $1/51$, while the area corresponding to $Y =$ no arthritis is $50/51$. □

Product Rule

The *product rule* allows us to express the joint distribution of two or more random variables in terms of their marginal and conditional distributions. Specifically, for two random variables X and Y, the joint pmf or pdf can be written as the product of the marginal pmf or pdf of one variable and the conditional pmf or pdf of the other variable given the first.

$$p(x, y) = p_X(x) \cdot p_{Y|X}(y|x).$$

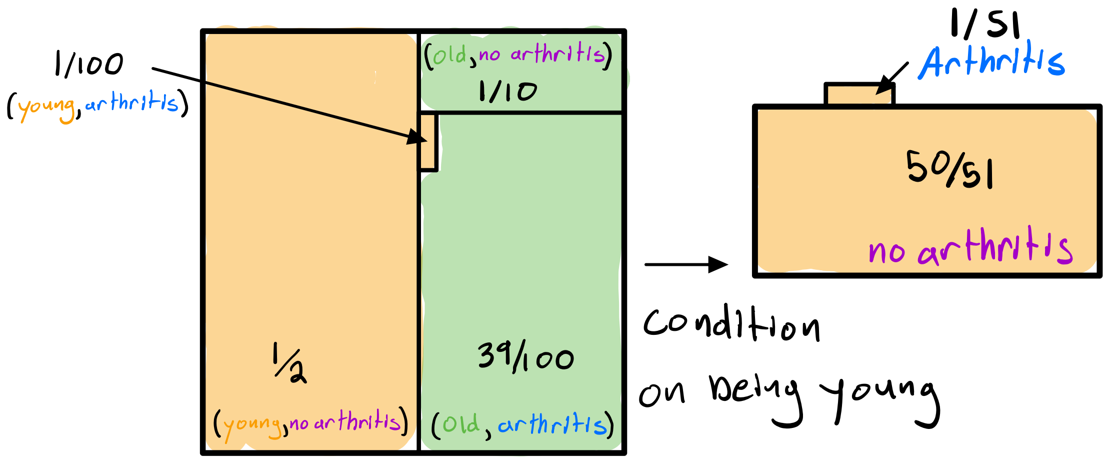


Figure 3.4: The shape on the right represenets all the area that remains when conditioning on $X =$ young.

This rule generalizes to more than two random variables. For a multivariate random variable $X = (X_1, X_2, \ldots, X_d)$, the joint pmf/pdf can be expressed as

$$p(x_1, x_2, \ldots, x_d) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_1, x_2) \cdots p(x_d|x_1, x_2, \ldots, x_{d-1}).$$

Example 3.13: Using the arthritis example, the joint pmf can be expressed using the product rule. For instance,

$$\begin{aligned} p(\text{young}, \text{arthritis}) &= p_X(\text{young}) \cdot p_{Y|X}(\text{arthritis}|\text{young}) \\ &= {}^{51}/_{100} \cdot {}^{1}/_{51} \\ &= {}^{1}/_{100}, \end{aligned}$$

which matches the original joint pmf. □

Bayes' Rule

Bayes' rule provides a way to invert conditional pmfs or pdfs, allowing us to express $p_{X|Y}(x|y)$ in terms of $p_{Y|X}(y|x)$ and the marginals.

$$p_{X|Y}(x|y) = \frac{p_{Y|X}(y|x) \cdot p_X(x)}{p_Y(y)}.$$

Example 3.14: Suppose we want to find the probability that a person is young given that they have arthritis. Using Bayes' rule:

$$\begin{aligned} \mathbb{P}_{X|Y}(X = \text{young}|Y = \text{arthritis}) &= p_{X|Y}(\text{young}|\text{arthritis}) \\ &= \frac{p_{Y|X}(\text{arthritis}|\text{young}) \cdot p_X(\text{young})}{p_Y(\text{arthritis})}. \end{aligned}$$

We have already the terms in the numerator in previous examples, giving us:

$$p_{Y|X}(\text{arthritis}|\text{young}) \cdot p_X(\text{young}) = \frac{1}{51} \cdot \frac{51}{100} = \frac{1}{100},$$

From the denominator we have:

$$\begin{aligned} p_Y(\text{arthritis}) &= p(\text{young}, \text{arthritis}) + p(\text{old}, \text{arthritis}) \\ &= {}^{1}\!/_{100} + {}^{39}\!/_{100} \\ &= {}^{40}\!/_{100}. \end{aligned}$$

Thus,

$$\mathbb{P}(X = \text{young}|Y = \text{arthritis}) = \frac{{}^{1}\!/_{51} \cdot {}^{51}\!/_{100}}{{}^{40}\!/_{100}} = \frac{{}^{1}\!/_{100}}{{}^{40}\!/_{100}} = \frac{1}{40}.$$

□

Independence

Two random variables X and Y are said to be *independent* if the occurrence of one does not affect the probability distribution of the other. Formally, X and Y are independent if

$$p_{Y|X}(y|x) = p_Y(y) \quad \text{for all } x \in \mathcal{X}, y \in \mathcal{Y}.$$

Alternatively, by the product rule, independence can be expressed as

$$p(x, y) = p_{Y|X}(y|x) \cdot p_X(x) = p_Y(y) \cdot p_X(x) \quad \text{for all } x \in \mathcal{X}, y \in \mathcal{Y}.$$

Similarly, for distributions, X and Y are independent if

$$\mathbb{P}(Y \in \mathcal{E}_Y, X \in \mathcal{E}_X) = \mathbb{P}(Y \in \mathcal{E}_Y) \cdot \mathbb{P}(X \in \mathcal{E}_X) \quad \text{for all events } \mathcal{E}_X \subset \mathcal{X}, \mathcal{E}_Y \subset \mathcal{Y}.$$

Independence is critical in machine learning. If two variables are independent, this has important modeling implications. For example, if feature X and target Y are independent, then X is not useful for predicting Y and so is not a useful feature.

Example 3.15: Consider two independent coin flips represented by random variables X and Y, each taking values in $\{0, 1\}$ with $p_X(0) = p_X(1) = {}^{1}\!/_{2}$ and $p_Y(0) = p_Y(1) = {}^{1}\!/_{2}$. Since the coin flips are independent, the joint pmf is

$$p(x, y) = p_X(x) \cdot p_Y(y) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} \quad \text{for all } x, y \in \{0, 1\}.$$

Thus, knowing the outcome of X does not change the probability distribution of Y, and vice versa. □

Example 3.16: In contrast, consider the arthritis example where X represents age and Y represents arthritis status. The joint pmf does not factor into the product of the marginals:

$$p(\text{young}, \text{arthritis}) = {}^{1}\!/_{100} \neq p_X(\text{young}) \cdot p_Y(\text{arthritis}) = {}^{51}\!/_{100} \cdot {}^{40}\!/_{100} = {}^{2040}\!/_{10000}.$$

This inequality indicates that X and Y are dependent; knowing a person's age affects the probability of having arthritis. $\square$

Notational Remark: For joint, marginal, and conditional distributions, probability mass functions (pmfs) and probability density functions (pdfs), the subscripts are sometimes omitted when the context makes it clear which distribution is being referred to. For example, instead of writing $p_{Y|X}(y|x)$ for the conditional pmf, we may simply write $p_{Y|X}(y|x)$ when it's clear that X and Y are the variables in question. Similarly, $p_X(x)$ and $p_Y(y)$ may be written as $p(x)$ and $p(y)$ respectively when there is no ambiguity.

Mixed Multivariate Random Variables

In some cases, a multivariate random variable consists of both continuous and discrete components. For example, consider a random variable X that is continuous and a random variable Y that is discrete. In such scenarios, p is neither a probability mass function (pmf) nor a probability density function (pdf). Instead, to calculate probabilities involving both X and Y, we use the product rule, expressing p as the product of the marginal pdf of X and the conditional pmf of Y given X:

$$p(x,y) = p_{Y|X}(y|x) \cdot p_X(x),$$

where $p_X(x)$ is a pdf and $p_{Y|X}(y|x)$ is a pmf.

Example 3.17: [Probability of wine] Consider a scenario where $X \in [0, 900]$ represents the amount of a chemical in a wine and $Y \in \{0, 1\}$ represents whether the wine is Barolo ($Y = 1$) or not ($Y = 0$). Assume that the amount of chemical X is uniformly distributed over $[0, 900]$, and the probability that the wine is Barolo given $X = x$ follows a Bernoulli distribution with parameter $x/900$. This means that p_X is a marginal pdf defined as

$$p_X(x) = {}^1\!/_{900} \quad \text{for } x \in [0, 900],$$

and $p_{Y|X}$ is a conditional pmf defined as

$$p_{Y|X}(y|x) = \begin{cases} x/900 & \text{if } y = 1, \\ 1 - x/900 & \text{if } y = 0. \end{cases}$$

To calculate the probability that $X \in [0, 50]$ and $Y = 1$, we compute:

$$\begin{aligned}
\mathbb{P}(X \in [0,50], Y = 1) &= \int_0^{50} \left(\sum_{y \in \{1\}} p(x,y) \right) dx \\
&= \int_0^{50} p(1|x) \cdot p(x) \, dx \\
&= \int_0^{50} \frac{1}{900} \cdot \frac{x}{900} \, dx \\
&= \frac{1}{810000} \frac{x^2}{2} \bigg|_0^{50} \\
&= \frac{1}{810000} \cdot \frac{2500}{2} = \frac{1}{648}.
\end{aligned}$$

Thus, the probability that the amount of chemical X is between 0 and 50 and the wine is Barolo ($Y = 1$) is $\frac{1}{648}$. $\square$

3.4 Representing Features, Labels, and Datasets as Random Variables

The concepts from the previous sections have now equipped us with the necessary tools to represent the features, labels, and datasets that we encounter in machine learning as random variables. In particular, we can represent a dataset of size $n \in \mathbb{N}$ as the following multivariate random variable

$$D = (Z_1, \dots, Z_n) \in \mathcal{Z}^n.$$

Each Z_i represents a feature-label pair, where $Z_i = (\mathbf{X}_i, Y_i) \in \mathcal{X} \times \mathcal{Y} = \mathcal{Z}$. The random variable $\mathbf{X}_i \in \mathbb{R}^d$ is often called the feature vector and denoted with a boldface to indicate that it is a vector. While $Y_i \in \mathcal{Y}$ is the label.

As we have learned, each random variable is associated with a probability distribution. We will use $\mathbb{P}_D$ the represent the probability distribution of the dataset D, and $\mathbb{P}_{Z_i}$ to represent the marginal probability distribution of the feature-label pair Z_i.

In supervised learning, the following assumptions are made:

1. $(\boldsymbol{X}_i, Y_i)$ are independent for all $i \in \{1, \dots, n\}$.
2. $\mathbb{P}_{Z_i} = \cdots = \mathbb{P}_{Z_n} = \mathbb{P}_Z$. Which means that all the feature-label pairs have the same distribution, which we denote with $\mathbb{P}_Z$.

These two assumptions together are often referred to as the *i.i.d. assumption*, which stands for independent and identically distributed. The i.i.d assumption implies that the probability of any event $\mathcal{E} = \mathcal{E}_1 \times \cdots \times \mathcal{E}_n \in \mathcal{Z}^n$ is

$$\begin{aligned}\mathbb{P}_D(D \in \mathcal{E}) = \mathbb{P}_D(Z_1 \in \mathcal{E}_1, \dots, Z_n \in \mathcal{E}_n) &= \mathbb{P}_{Z_1}(Z_1 \in \mathcal{E}_1) \cdots \mathbb{P}_{Z_n}(Z_n \in \mathcal{E}_n) \\ &= \mathbb{P}_Z(Z \in \mathcal{E}_1) \cdots \mathbb{P}_Z(Z \in \mathcal{E}_n).\end{aligned}$$

The first equality holds since the feature-label pairs are independent, and the second equality holds since they are identically distributed.

All of the above will often be communicated in a more concise way by writing

$$D = \{(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)\} \in (\mathcal{X} \times \mathcal{Y})^n$$

where $(\boldsymbol{X}_i, Y_i) \sim \mathbb{P}_{\boldsymbol{X},Y}$ are independent for all $i \in \{1, \dots, n\}$. This is to be read as "D contains n independent samples of feature-label pairs $(\boldsymbol{X}_i, Y_i)$, each drawn from the same distribution $\mathbb{P}_{\boldsymbol{X},Y}$".

3.5 Functions of Random Variables

A function of a random variable is itself a random variable. Let $X \in \mathcal{X}$ be a random variable with pmf or pdf p and let $f : \mathcal{X} \to \mathcal{Y}$ be a function. Then, $f(X)$ is a random variable with outcome space $\mathcal{Y}$.

Example 3.18: [Fair six-sided die] Let $X \in \mathcal{X}$ be a random variable representing the outcome of a fair six-sided die. It has outcome space $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$ and pmf $p(x) = 1/6$ for all $x \in \mathcal{X}$. Let $f(X) = X^2$ be a function of X, which is also a random variable. The outcome space of $f(X)$ is $\{1, 4, 9, 16, 25, 36\}$. Sometimes a function of a random variable is assigned a new symbol, such as $Y = f(X)$. The pmf of Y is $p_Y(y) = p_{f(X)}(y) = 1/6$ for all $y \in \{1, 4, 9, 16, 25, 36\}$. In this case $p_Y(x^2) = p(x)$ for all $x \in \{1, 2, 3, 4, 5, 6\}$. However, this is not always the case, as we will see in the next example. □

Example 3.19: [Payout from a slot machine] Let $X \in \mathcal{X}$ be a random variable representing the payout from a slot machine. It has an outcome space $\mathcal{X} = [-10, 10]$ and pdf $p(x) = 1/20$ for all $x \in \mathcal{X}$. Let $Y = f(X) = X^2$ be a function of X. The outcome space of Y is $[0, 100]$ and the pmf of Y is $p_Y(y) = p_{f(X)}(y) = \frac{1}{20\sqrt{y}}$ for all $y \in [0, 100]$. Notice that $p_Y(x^2) \neq p(x)$ for all $x \in [-10, 10]$. □

In general the pmf or pdf of $f(X)$ is not simply $p(x)$ as we have seen in the previous example. Fortunately, in these notes we will mainly be interested in the expected value of functions of random variables (discussed in Section 3.6), in which case working with $p(x)$ will turn out to be sufficient. As such, we avoid discussing how to calculate the pmf or pdf of $f(X)$ in these notes.

3.5.1 The Predictor and Learner are Functions of Random Variables

In Chapter 2 we discussed how the predictor and learner are functions. And in Section 3.4 we discussed how datasets and feature-label pairs are random variables. As such, the predictor and learner are functions of random variables and thus are themselves random variables.

Example 3.20: [Predictor] Let $f : \mathcal{X} \to \mathcal{Y}$ be a predictor, where $\mathcal{X}$ is the set of feature vectors and $\mathcal{Y}$ is the set of labels. Let $\boldsymbol{X} \in \mathcal{X}$ be a random variable representing a feature vector. Then $f(\boldsymbol{X})$ is a random variable representing the predicted label and has outcome space $\mathcal{Y}$. □

Example 3.21: [Learner] Let $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \{f | f : \mathcal{X} \to \mathcal{Y}\} = \mathcal{F}$ be a learner, where $(\mathcal{X} \times \mathcal{Y})^n$ is the set of datasets and $\mathcal{F}$ is the set of predictors. Let $D \in (\mathcal{X} \times \mathcal{Y})^n$ be a random variable representing a dataset. Then $\mathcal{A}(D)$ is a random variable representing the learned predictor and has outcome space $\mathcal{F}$.

Notice that in this example the random variable $\mathcal{A}(D)$ has an outcome space that is a set of functions $\mathcal{F}$. This is perfectly fine, since random variables can have outcome spaces that are sets of any objects. □

The reason it is important to think of the predictor and learner as random variables is that it allows us to use the tools of probability theory to reason about them. In particular, we will be able to talk about things like the probability that predictor f outputs some label y or the probability that learner $\mathcal{A}$ outputs some predictor f.

3.6 Expectation and Variance

The *expected value*, or *mean*, of a random variable $X \in \mathcal{X}$ is its average value if you sample according to its distribution infinitely many times. It is not necessarily the value we expect to see most frequently, that is called the mode. To calculate the expected value of a random variable, the random variable must have an outcome space $\mathcal{X} \subset \mathbb{R}$ that only contains numbers. The reason for this is that if $\mathcal{X} = \{\text{cat}, \text{dog}\}$, it is not clear how to calculate the average of a cat and a dog.

More precisely, if $X \in \mathcal{X} \subset \mathbb{R}$ is a random variable with pmf or pdf p, then the expected value of X is

$$\mathbb{E}[X] \overset{\text{def}}{=} \begin{cases} \sum_{x \in \mathcal{X}} xp(x) & \text{if } X \text{ is discrete} \\ \int_{\mathcal{X}} xp(x)dx & \text{if } X \text{ is continuous} \end{cases}$$

Example 3.22: [Fair six-sided die] Let $X \in \mathcal{X}$ be a random variable representing the outcome of a fair six-sided die. It has outcome space $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$ and pmf $p(x) = 1/6$ for all $x \in \mathcal{X}$. The expected value of X is

$$\mathbb{E}[X] = \sum_{x \in \mathcal{X}} x \cdot p(x) = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = 3.5.$$

Thus, the expected value of a fair six-sided die is 3.5. Notice how the expected value is not a value that the die can actually take on, but is the average value if you roll the die infinitely many times. □

Example 3.23: [Unfair coin] Let $X \in \mathcal{X}$ be a random variable representing the outcome of an unfair coin flip, with distribution Bernoilli(α). It has outcome space $\mathcal{X} = \{0, 1\}$ and pmf $p(x) = \alpha^x(1-\alpha)^{1-x}$ for all $x \in \mathcal{X}$. The expected value of X is

$$\mathbb{E}[X] = \sum_{x \in \mathcal{X}} x \cdot p(x) = 0 \cdot \alpha^0(1-\alpha)^1 + 1 \cdot \alpha^1(1-\alpha)^0 = \alpha.$$

Thus, the expected value of an unfair coin flip with probability α of landing heads is α. Notice again how the expected value is not a value that the coin can actually take on, but is the average value if you flip the coin infinitely many times. □

Example 3.24: [Normal distribution] Let $X \in \mathcal{X}$ be a random variable representing a Normal distribution with mean μ and variance σ^2. It has outcome space $\mathcal{X} = \mathbb{R}$ and pdf $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$. The expected value of X is

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x \cdot p(x)\, dx = \int_{-\infty}^{\infty} x \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \mu.$$

Thus, the naming convention of calling the parameter μ the mean of the Normal distribution is justified. The last equality above is not trivial to show, thus we will not prove it here. □

3.6.1 Expectation of Functions of Random Variables

In general, we may be interested in the expected value of functions of random variables. If X is a random variable and $f : \mathcal{X} \to \mathbb{R}$ is a function, then the expected value of $f(X)$ is

$$\mathbb{E}[f(X)] \overset{\text{def}}{=} \begin{cases} \sum_{x \in \mathcal{X}} f(x)p(x) & \text{if } X \text{ is discrete} \\ \int_{\mathcal{X}} f(x)p(x)dx & \text{if } X \text{ is continuous.} \end{cases}$$

Notice that it was important that the codomain of f is $\mathbb{R}$ so that we can calculate the average of the values that f takes on.

Example 3.25: [Payout from a slot machine] Let $X \in \mathcal{X}$ be a random variable representing the payout from a slot machine. It has an outcome space $\mathcal{X} = [-10, 10]$ and pdf $p(x) = 1/20$ for all $x \in \mathcal{X}$. Let $f(X) = X^2$ be a function of X. The expected value of $f(X)$ is

$$\mathbb{E}\left[f(X)\right] = \int_{\mathcal{X}} f(x)p(x)dx = \int_{-10}^{10} x^2 \cdot \frac{1}{20}\,dx = \frac{1}{20}\frac{x^3}{3}\bigg|_{-10}^{10} = \frac{1}{20}\left(\frac{1000}{3} - \frac{-1000}{3}\right) = \frac{2000}{60}.$$

In Example 3.18 we said that sometimes we represent a function of a random variable with a new symbol, such as $Y = f(X)$. Using this notation one should wonder why the expected value of Y was not calculated by using its pdf $p_Y(y) = \frac{1}{20\sqrt{y}}$. First, let us calculate the expected value of Y using its pdf:

$$\mathbb{E}\left[Y\right] = \int_{\mathcal{Y}} y \cdot p_Y(y)dy = \int_{0}^{100} y \cdot \frac{1}{20\sqrt{y}}\,dy = \frac{1}{20}\frac{2}{3} \cdot y^{3/2}\bigg|_{0}^{100} = \frac{2000}{60}.$$

We see that calculating the expected value of Y using its pdf gives the same result as calculating the expected value of $f(X)$ using the pdf of X. This is a result that holds in general. Often it is hard to calculate the pdf of a function of a random variable; luckily, the expected value can be calculated using the pdf of the original random variable. This is a useful property of expected values, and justifies why we often only need to know the pdf of the original random variable. $\square$

Example 3.26: [Fair six-sided die] Let $X \in \mathcal{X}$ be a random variable representing the outcome of a fair six-sided die. It has outcome space $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$ and pmf $p(x) = 1/6$ for all $x \in \mathcal{X}$. Let $f(X) = X^2$ be a function of X. The expected value of $f(X)$ is

$$\mathbb{E}\left[f(X)\right] = \sum_{x \in \mathcal{X}} f(x) \cdot p(x) = 1^2 \cdot \frac{1}{6} + 2^2 \cdot \frac{1}{6} + 3^2 \cdot \frac{1}{6} + 4^2 \cdot \frac{1}{6} + 5^2 \cdot \frac{1}{6} + 6^2 \cdot \frac{1}{6} = 15.17.$$

Thus, the expected value of the square of a fair six-sided die is 15.17. $\square$

3.6.2 Variance of Random Variables

There is a special function of a random variable that is often of interest, the *variance*. Roughly speaking, it measures how much the values of a random variable deviate from the expected value. The variance of a random variable X is defined as

$$\text{Var}\left[X\right] \overset{\text{def}}{=} \mathbb{E}\left[(X - \mathbb{E}\left[X\right])^2\right] = \mathbb{E}\left[X^2\right] - \left(\mathbb{E}\left[X\right]\right)^2.$$

The last equality is a useful way to calculate the variance, as it only requires the expected value of X and the expected value of X^2. It can be shown by expanding the definition of the variance and using the properties in Section 3.7.

Example 3.27: [Unfair coin] Let $X \in \mathcal{X}$ be a random variable representing the outcome of an unfair coin flip, with distribution Bernoilli(α). It has outcome space $\mathcal{X} = \{0, 1\}$ and pmf $p(x) = \alpha^x(1-\alpha)^{1-x}$ for all $x \in \mathcal{X}$. In Example 3.23 we showed the expected value of X is $\mathbb{E}[X] = \alpha$. We can calculate the variance of X using the defintion of variance as

$$\begin{aligned}
\operatorname{Var}[X] &= \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \\
&= \sum_{x\in\mathcal{X}} (x - \mathbb{E}[X])^2 \cdot p(x) \\
&= (0-\alpha)^2 \cdot \alpha^0(1-\alpha)^1 + (1-\alpha)^2 \cdot \alpha^1(1-\alpha)^0 \\
&= \alpha^2(1-\alpha) + (1-\alpha)^2\alpha = \alpha(1-\alpha).
\end{aligned}$$

We can also calculate the variance using the formula $\operatorname{Var}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.

$$\operatorname{Var}[X] = \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2 = 0^2 \cdot \alpha^0(1-\alpha)^1 + 1^2 \cdot \alpha^1(1-\alpha)^0 - \alpha^2 = \alpha - \alpha^2 = \alpha(1-\alpha).$$

As expected, the two methods give the same result. Thus, the variance of an unfair coin flip with probability α of landing heads is $\alpha(1-\alpha)$. □

Example 3.28: [Normal distribution] Let $X \in \mathcal{X}$ be a random variable representing a Normal distribution with mean μ and variance σ^2. It has outcome space $\mathcal{X} = \mathbb{R}$ and pdf $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$. We can calculate the variance of X using the defintion of variance as

$$\begin{aligned}
\operatorname{Var}[X] &= \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \\
&= \int_{-\infty}^{\infty} (x-\mu)^2 \cdot p(x)\, dx \\
&= \int_{-\infty}^{\infty} (x-\mu)^2 \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx. = \sigma^2.
\end{aligned}$$

The last equality is not trivial to show, thus we will not prove it here. The important take away is that the variance of a Normal distribution is equal to its variance parameter σ^2. □

3.6.3 Expectation of Multivariate Random Variables

All of the above can be extended to multivariate random variables. We cover the case of two random variables, but the extension to more random variables follows a similar pattern. Let $X \in \mathcal{X}, Y \in \mathcal{Y}$ be random variables with joint pmf or pdf $p(x, y)$. Let $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ be a function. Then the expected value of $f(X, Y)$ is

$$\mathbb{E}[f(X,Y)] \overset{\text{def}}{=} \begin{cases} \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} f(x,y) \cdot p(x,y) & \text{if } X \text{ and } Y \text{ are discrete} \\ \int_{\mathcal{X}} \int_{\mathcal{Y}} f(x,y) \cdot p(x,y)\, dx\, dy & \text{if } X \text{ and } Y \text{ are continuous.} \end{cases}$$

Example 3.29: [Two fair six-sided dice] Let $X \in \mathcal{X}, Y \in \mathcal{Y}$ be random variables representing the outcomes of two fair six-sided dice. They have outcome spaces $\mathcal{X} = \mathcal{Y} = \{1, 2, 3, 4, 5, 6\}$ and joint pmf $p(x, y) = 1/36$ for all $x, y \in \mathcal{X}$. Let $f(X, Y) = X + Y$ be a function of X and Y. The expected value of $f(X, Y)$ is

$$\begin{aligned}\mathbb{E}\left[f(X,Y)\right] &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} f(x,y)\cdot p(x,y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}}(x+y)\cdot\frac{1}{36} \\ &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} x\cdot\frac{1}{36} + \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} y\cdot\frac{1}{36} = 7.\end{aligned}$$

Thus, the expected value of the sum of two fair six-sided dice is 7. □

If X is continuours and Y is discrete, or vice versa, then we must use the product rule to express $p(x, y) = p_X(x)p_{Y|X}(y|x)$. In these two cases the expected value of $f(X, Y)$ is

$$\mathbb{E}\left[f(X,Y)\right] \overset{\text{def}}{=} \begin{cases}\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} f(x,y)p_{Y|X}(y|x)p_X(x) & \text{if } X \text{ is discrete and } Y \text{ is continuous} \\ \int_{\mathcal{X}}\int_{\mathcal{Y}} f(x,y)p_{Y|X}(y|x)p_X(x)\,dx\,dy & \text{if } X \text{ is continuous and } Y \text{ is discrete.}\end{cases}$$

Example 3.30: [Expected loss of predicting wine] Consider the same scenario as in Example 3.17, where $X \in [0, 900]$ represents the amount of a chemical in a wine and $Y \in \{0, 1\}$ represents whether the wine is Barolo ($Y = 1$) or not ($Y = 0$). Assume that X is uniformly distributed over $[0, 900]$, and the probability that the wine is Barolo given $X = x$ follows a Bernoulli distribution with parameter $x/900$.

Let $f(X) = {}^{X}\!/_{900}$, and define $\ell(f(X), Y) = (f(X)-Y)^2$. We aim to compute $\mathbb{E}[\ell(f(X), Y)]$.

$$\begin{aligned}\mathbb{E}[\ell(f(X),Y)] &= \int_{\mathcal{X}}\sum_{y\in\mathcal{Y}} \ell(f(x),y)p(y|x)p(x)\,dx \\ &= \int_0^{900}\left(\sum_{y\in\{0,1\}}\left(\frac{x}{900}-y\right)^2 p(y|x)\right)p(x)\,dx \\ &= \int_0^{900}\left(\left(\frac{x}{900}-0\right)^2\left(1-\frac{x}{900}\right)+\left(\frac{x}{900}-1\right)^2\left(\frac{x}{900}\right)\right)\frac{1}{900}\,dx \\ &= \frac{1}{900}\int_0^{900}\frac{x}{900}\left(1-\frac{x}{900}\right)dx \\ &= \frac{1}{900^2}\int_0^{900} x - \frac{1}{900^3}\int_0^{900} x^2 dx \\ &= \frac{1}{900^2}\frac{x^2}{2}\Big|_0^{900} - \frac{1}{900^3}\frac{x^3}{3}\Big|_0^{900} \\ &= \frac{1}{2}-\frac{1}{3}=\frac{1}{6}.\end{aligned}$$

We will see this type of calcultation again in the next chapter. To slightly foreshadow what is to come, the function f can be thought as a predictor while ℓ will be something we call a loss function, which measures how close the prediction $f(X)$ is to the true label Y. As such, the expectation we calculated is the expected loss of the predictor f. If we had different predictors, we could compare them by calculating their expected loss and picking the one with the lowest expected loss. □

3.6.4 Conditional Expectation

The expected value of a random variable given some information is called the *conditional expectation*. Let $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$ be random variables with the conditional pmf or pdf of Y given $X = x$ denoted by $p(y|x)$. If $f : \mathcal{Y} \to \mathbb{R}$ is a function of Y, then the conditional expectation of $f(Y)$ given $X = x$ for some $x \in \mathcal{X}$ is

$$\mathbb{E}\left[f(Y)|X = x\right] \overset{\text{def}}{=} \begin{cases} \sum_{y \in \mathcal{Y}} f(y)p(y|x) & \text{if } Y \text{ is discrete} \\ \int_{\mathcal{Y}} f(y)p(y|x)\,dy & \text{if } Y \text{ is continuous.} \end{cases}$$

Example 3.31: [Wine depending on the amount of a chemical] We continue with the wine example from Example 3.17. Suppose we are interested in what type of wine we should expect given that the amount of the chemical is $x = 900$. We would then choose $f(Y) = Y$ to just be the identity function and calculate $\mathbb{E}\left[f(Y)|X = 900\right]$ as follows

$$\mathbb{E}\left[f(Y)|X = 900\right] = \sum_{y \in \mathcal{Y}} yp(y|900) = 0 \cdot p(0|900) + 1 \cdot p(1|900) = 1 \cdot {}^{900}/_{900} = 1.$$

This means that given that the amount of the chemical is 900, we should expect the wine to be Barolo (represented as $Y = 1$). Intuitively, this makes sense since (by the definition of Bernoilli(${}^{900}/_{900}$)) with probability 1 the wine is Barolo given that the amount of the chemical is 900. □

Example 3.32: [Coin flip depending on a dice roll] Let $X \in \{1, 2, 3, 4, 5, 6\}$ be a random variable representing the outcome of a fair six-sided die, and let $Y \in \{0, 1\}$ be a random variable representing the outcome of an unfair coin flip. Assume the conditional distribution of Y given $X = x$ is Bernoilli(${}^{x^2}/_{36}$). Let $f(Y) = 10Y$. Suppose we were interested in the conditional expectation of $f(Y)$ given a three was rolled. This can be calculated as

$$\mathbb{E}\left[f(Y)|X = 3\right] = \sum_{y \in \mathcal{Y}} f(y)p(y|3) = 10 \cdot 0 \cdot \left(1 - \frac{3^2}{36}\right) + 10 \cdot 1 \cdot \frac{3^2}{36} = 10 \cdot \frac{3^2}{36} = \frac{15}{6}$$

□

3.7 Properties of expectations and variances

We state some useful properties of expectations and variances, without proof. Consider random variables, X and Y. For a constant $c \in \mathbb{R}$, it holds that:

1. $\mathbb{E}[cX] = c\mathbb{E}[X]$
2. $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$ ▷ linearity of expectation
3. $\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X]$ ▷ law of total expectation
4. $\text{Var}[c] = 0$ ▷ the variance of a constant is zero
5. $\text{Var}[cX] = c^2\text{Var}[X]$.

In addition, if X and Y are independent random variables, it holds that:

6. $\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y]$
7. $\mathrm{Var}[X+Y] = \mathrm{Var}[X] + \mathrm{Var}[Y]$.

Exercise 3.2: Show the third property of expectations, $\mathbb{E}[\mathbb{E}[X]] = \mathbb{E}[X]$. □

3.8 Probability Review Exercises

Phew, that was a lot about probability. This section includes more exercises to help you get used to all this new terminology and new concepts. Answers to these solutions can be found in Appendix 12.

Exercise 3.3: You roll a fair six-sided die once.

1. What is the sample space for rolling the dice?
2. What is the probability of observing an even number?
3. What is the probability of not observing 3 when you roll the dice?
4. What is the probability of neither observing 3 nor 4 when you roll the dice?

Exercise 3.4: In a Bernoulli trial, the probability of success was 0.7. What is the probability of failure?

Exercise 3.5: Assume X is a continuous random variable with a uniform distribution on the interval $[-5, 5]$.

1. What is $P(X \leq 4.5)$?
2. What is $P(-3 \leq X \leq 3)$?

Exercise 3.6: Two discrete random variables X and Y have the following joint distribution,

	$x = 1$	$x = 2$	$x = 3$
$y = 1$	$\frac{3}{18}$	$\frac{1}{18}$	$\frac{1}{18}$
$y = 2$	$\frac{1}{18}$	$\frac{3}{18}$	$\frac{1}{18}$
$y = 3$	$\frac{2}{18}$	c	$\frac{3}{18}$

1. What is c?
2. What is $p(x = 3)$?
3. What is $p(y = 1)$?

Exercise 3.7: Two continuous random variables X and Y have the following joint pdf,

$$p(x,y) = \begin{cases} \frac{cx^2}{y^2} & y \geq 1, 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

1. Write the outcome space for X and the outcome space for Y?
2. Write the joint outcome space for the two-dimensional random variable (X, Y).
3. What is c?
4. Calculate $p(y|x = 0.5)$.

Chapter 4

Supervised Learning

We are finally ready to formally present the supervised learning setting. In supervised learning, we are given a random dataset $D = ((\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)) \in (\mathcal{X} \times \mathcal{Y})^n$ where each $(\boldsymbol{X}_i, Y_i)$ is an independent and identically distributed according to $\mathbb{P}_{\boldsymbol{X},Y}$ for all $i \in \{1, \ldots, n\}$. $\boldsymbol{X}_i$ is often called a features, and Y_i is often called a label or target. We will assume that the features are always represented as a vector $\boldsymbol{X}_i \in \mathbb{R}^d$ for some $d \in \mathbb{N}$.

Example 4.1: [Predicting house prices] If we are trying to predict house prices, then $\boldsymbol{X}_i \in \mathbb{R}^3$ could be a vector of three features such as the number of bedrooms, the number of floors, and the age of the house. $Y_i \in \mathbb{R}$ would be the price of the house. □

Example 4.2: [Predicting the type of wine] If we are trying to predict the type of wine, then $\boldsymbol{X}_i \in \mathbb{R}^2$ could be a vector of two features such as the alcohol content and the acidity. $Y_i \in \{A, B, C, D\}$ would be the type of wine. □

Example 4.3: [Classifying the type of image based on pixel values] If we are trying to classify the type of image, then $\boldsymbol{X}_i \in \mathbb{R}^{400}$ could be a vector of pixel values of a 20×20 image. $Y_i \in \{\text{cat}, \text{dog}, \text{bird}\}$ would be the type of image. □

It is not always clear what the features and labels are. For example, in the case of predicting house prices, we could have a dataset where the features are the price of the house and the number of bedrooms, and the label is the age of the house. Usually the features are the information that is easy to collect, while the label is harder to collect, and thus we want to predict the label from the features. In the case of the pixel images, it is likely easy to collect lots of images from the internet, but it is hard to label them.

We have defined the data we recieve; however, we have not defined what learning well means. This will be the focus of the next section.

4.1 Defining the Learning Objective

In Chapter 1, we discussed how learning is the process of taking experience (the dataset) and using it to produce knowledge (the predictor). In Chapter 2 we explained how a program that is doing the learning can be represented by a function $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \{f | f : \mathcal{X} \to \mathcal{Y}\}$ called the learner. The learner takes as input a random dataset D and outputs a predictor f. Roughly speaking, a good learner is then one that outputs a good predictor. This objective can be written as

Objective 1 (Informal) *Define a learner $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \{f | f : \mathcal{X} \to \mathcal{Y}\}$ such that the predictor $\hat{f}$ is good, where $\mathcal{A}(D) = \hat{f}$.*

To develop some intuition of what a good predictor is we will consider the example of predicting house prices based on the number of rooms. Here, the number of rooms would be the feature and the price would be the label. Suppose we are given an arbitrary predictor $f : \mathcal{X} \to \mathcal{Y}$. Next, we are presented with the number of rooms $\boldsymbol{X} = 2$ of some random house, but we are not given the price $Y = 300$. We would likely say the predictor is good if $f(2) = 300$. However, consider the case where we are presented with the number of rooms 2 of some other random house, but we are not given its price of 400. Now, it seems less clear what a good predictor would do. If the predictor outputs $f(2) = 400$, then it predicts the price of the second house correctly, but is wrong for the first house. Perhaps a predictor that outputs $f(2) = 350$ is better, but why should we think that being 50 off from both house prices is better than being 100 off from one house price and 0 off from the other? We can image considering this example unrolling for infinitely more random houses.

This example was trying to illustrate two important things that need to be considered when evaluating the quality of a predictor. First, the feature vector $\boldsymbol{X}$ and label Y are randomly drawn from the distribution $\mathbb{P}_{\boldsymbol{X},Y}$, so the predictor could potentially be tasked with predicting the label based on any feature vector. Second, we need to consider how to measure how close a prediction $f(\boldsymbol{X})$ is to the true label Y. To address the first problem, we will choose to evaluate the predictor in expectation over all possible feature-label pairs. To address the second problem, we will measure the closeness of the prediction to the true label using a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. However, the choice of loss function will depend on the problem. For now we keep the loss function abstract and will discuss concrete examples in the next section.

Formally, a good predictor f should have a small expected loss. The expected loss of a predictor f is defined as

$$L(f) = \mathbb{E}[\ell(f(\boldsymbol{X}), Y)]$$

where the expectation is taken over the distribution $\mathbb{P}_{\boldsymbol{X},Y}$. Notice how $\ell(f(\boldsymbol{X}), Y)$ is a random variable, since it is a function of the random feature-label pair $(\boldsymbol{X}, Y)$. Using this definition, we can attempt to define the learning objective more formally.

Objective 2 (Almost formal) *Define a learner $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \{f | f : \mathcal{X} \to \mathcal{Y}\}$ such that $L(\hat{f})$ is small, where $\mathcal{A}(D) = \hat{f}$.*

We write that this almost formal because there is still some ambiguity. In particular, the careful reader might notice that the dataset D is random, so the predictor $\mathcal{A}(D) = \hat{f}$, is also random, since the learner $\mathcal{A}$ is a function of the random variable D. This means that if the value of D changes, then $\hat{f}$ might also change. So the question becomes, for which datasets D do we want the predictor to have a small expected loss? Perhaps we can define the learner such that it outputs the best predictor for all possible datasets. This turns out to be an unachievable goal. Instead we will ask the learner to output a good predictor in expectation over all datasets. We can formally write the objective now as

Objective 3 (Formal) *Define a learner $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \{f | f : \mathcal{X} \to \mathcal{Y}\}$ such that $\mathbb{E}\left[L(\mathcal{A}(D))\right]$ is small.*

Notice how $L(\mathcal{A}(D))$ is a random variable, since it is a function of the random variable $\mathcal{A}(D)$, which itself is a function of the random variable D. This objective can seem complicated. To ease the analysis of how to achieve it, at times we will assume the dataset D is not random, and instead is fixed. To denote that we are dealing with a fixed dataset we will write $\mathcal{D}$ instead of D. Doing this will help us study the learner $\mathcal{A}$ in isolation from the randomness of the dataset D. For instance, we can study how to define $\mathcal{A}(\mathcal{D}) = \hat{f}$ such that $L(\hat{f})$ is small for a fixed dataset $\mathcal{D}$.

In Section 4.3 we discuss the approach we will take to achieving the above objective. First, we discuss two common types of supervised learning problems. Namely, regression and classification.

4.2 Regression and Classification

The distinction between regression and classification is based on the type of label $Y \in \mathcal{Y}$. In regression, there is a notion of order between the labels Y in the set of labels $\mathcal{Y}$, while in classification there is no such notion of order. Usually, the set of labels $\mathcal{Y}$ is either $\mathbb{R}$ or an interval in $\mathbb{R}$ for regression, and a finite set for classification.

Example 4.4: [Regression] We provide some examples of sets of labels $\mathcal{Y}$ for regression.

- Predicting house prices: $\mathcal{Y} = [0, \infty)$ represents the set of all possible prices of a house. Alternatively, we could write $\mathcal{Y} = \mathbb{R}$, where of course the price of a house cannot be negative. There is a notion of order between the prices of houses, since a house that costs \$200k is closer in prices to a house that costs \$300k than to a house that costs \$1M.

- Predicting stock prices: $\mathcal{Y} = [0, \infty)$ represents the set of all possible prices of a stock. There is a notion of order between the prices of stocks, since a stock that costs \$200 is closer in prices to a stock that costs \$300 than to a stock that costs \$1,000.

- Predicting energy consumption of a factory: $\mathcal{Y} = \mathbb{R}$ represents the set of all possible energy consumption values. There is a notion of order between the energy consumption values, since a factory that consumes 100 kWh is closer in energy consumption to a factory that consumes 200 kWh than to a factory that consumes 1,000 kWh.

- Prediction the temperature tomorrow: $\mathcal{Y} = \mathbb{R}$ represents the set of all possible temperatures. There is a notion of order between the temperatures, since a temperature of 20° is closer in temperature to a temperature of 25° than to a temperature of 30°.

□

Example 4.5: [Classification]

- Predicting the type of wine: $\mathcal{Y} = \{A, B, C, D\}$ represents the set of possible types of wine. There is no notion of order between the types of wine, since there is no way to say that type A is closer to type B than to type C.

- Predicting the type of image: $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{bird}\}$ represents the set of possible types of images. There is no notion of order between the types of images, since there is no way to say that a cat is closer to a dog than to a bird.

- Predicting the type of email: $\mathcal{Y} = \{\text{spam}, \text{not spam}\}$ represents the set of possible types of emails. There is no notion of order since there are only two types of emails.

- Predicting the type of disease: $\mathcal{Y} = \{\text{cancer}, \text{diabetes}, \text{flu}\}$ represents the set of possible types of diseases. There is no notion of order between the types of diseases.

□

The reason it is important to distinguish between regression and classification is that the loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ will depend on the type of label Y. For regression we use loss functions that measure how close the prediction $f(\boldsymbol{X})$ is to the true label Y. Some common choices are the absolute loss

$$\ell(f(\boldsymbol{X}), Y) = |f(\boldsymbol{X}) - Y|$$

and the squared loss

$$\ell(f(\boldsymbol{X}), Y) = (f(\boldsymbol{X}) - Y)^2.$$

For classification there is no notion of closeness between the prediction and the true label, so we use loss functions that measure how well the prediction matches the true label. The common choice here is the zero-one loss

$$\ell(f(\boldsymbol{X}), Y) = \begin{cases} 0 & \text{if } f(\boldsymbol{X}) = Y \\ 1 & \text{if } f(\boldsymbol{X}) \neq Y. \end{cases}$$

4.3 Empirical Risk Minimization

In this section we will discuss how we can define a learner $\mathcal{A}(\mathcal{D}) = \hat{f}$ such that $L(\hat{f})$ is small for a fixed dataset $\mathcal{D}$. Before getting into the details we make a small notational remark. Untill now we have called $L(f)$ the expected loss of the predictor f; however, another common name for $L(f)$ is the *risk* of the predictor f. Throughout the rest of the notes we will use the term risk and expected loss interchangeably.

Returning to our goal of making $L(\hat{f})$ small. One might naively think that we can simply calculate $L(f)$ for all possible predictors f and choose the one with the smallest loss. However, this idea quickly runs into a problem. In particular, we cannot calculate $L(f)$ since we do not know the distribution $\mathbb{P}_{\boldsymbol{X},Y}$. Recall that we are only given a dataset $\mathcal{D}$ and not the distribution $\mathbb{P}_{\boldsymbol{X},Y}$.

To address this problem we will use the dataset $\mathcal{D}$ to estimate the risk $L(f)$ of a predictor f. This estimate will be denoted as $\hat{L}(f)$ and since it is based on data it is common to call this estimte the *empirical risk* of the predictor f. If we can ensure that the empirical risk $\hat{L}(f)$ is a good estimate of the risk $L(f)$, then we can simply choose the predictor $\hat{f}$ that minimizes the empirical risk. However, we need to be careful about the set of predictors we are considering. It turns out that if the set of predictors is too large or unstructured we will run into two problems. The first is it will be computaitonally infeasible to search through all of the predictors to find the one with the smallest empirical risk. The second is that our estimate $\hat{L}(f)$ will be a poor estimate of the risk $L(f)$. To address these problems we will restrict the set of predictors to a class of functions $\mathcal{F}$ which is just a set of predictors that is a strict subset of all possible predictors $\{f | f : \mathcal{X} \to \mathcal{Y}\}$.

This approach has the obvious name of *empirical risk minimization* (ERM). More precisely, the ERM approach is defined as follows.

Definition 1 (Empirical Risk Minimization) *Given a fixed dataset $\mathcal{D}$ and function class $\mathcal{F}$, the empirical risk minimization approach does the following:*

1. ***Estimation:*** *Use $\mathcal{D}$ to estimate the risk $L(f)$ for any predictor $f \in \mathcal{F}$. This estimate is called the empirical risk and is denoted as $\hat{L}(f)$.*
2. ***Optimization:*** *Choose the predictor $\hat{f}$ to be the predictor $f \in \mathcal{F}$ that minimizes the empirical risk $\hat{L}(f)$.*

We have defined ERM for a fixed dataset $\mathcal{D}$; however, the definition is exactly the same for a random dataset D, except now the predictor $\hat{f}$ is random.

Notice that ERM is indeed a learner $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \{f | f : \mathcal{X} \to \mathcal{Y}\}$, since it takes as input a dataset $\mathcal{D}$ and outputs a predictor $\hat{f} \in \mathcal{F} \subset \{f | f : \mathcal{X} \to \mathcal{Y}\}$. The ERM approach is a general learner that can be applied to any supervised learning problem. In the next chaper we will discuss how to do the estimation step of ERM. While in Chapter 6 we will discuss how to do the optimization step of ERM.

We present some high-level examples of what the ERM approach looks like for some common supervised learning problems.

Example 4.6: [ERM for house price prediction] Suppose we are trying to predict house prices based on the number of rooms. The set of predictors $\mathcal{F}$ could be the set of all lines of the form $f(x) = w \cdot x + b$. The empirical risk $\hat{L}(f)$ could be estimated by calculating the average squared loss over the dataset $\mathcal{D}$. The optimization step would then be to find the values of w and b that minimize the empirical risk. The predictor $\hat{f}$ would then be the line that best fits the data. $\square$

Example 4.7: [ERM for image classification] Suppose we are trying to classify images of cats and dogs based on the pixel values. The set of predictors $\mathcal{F}$ could be the set of all neural networks of a fixed architecture[1]. The empirical risk $\hat{L}(f)$ could be estimated by calculating the average zero-one loss over the dataset $\mathcal{D}$. The optimization step would then be to find the weights of the neural network that minimize the empirical risk. The predictor $\hat{f}$ would then be the neural network that best classifies the images. $\square$

[1]You should think of a neural network with a fixed architecture as just some complicated predictor function.

Chapter 5

Estimation

Estimation is the proces of using data to infer the value of an unknown quantity. In this chapter our goal will be to introduce a way of performing the estimation step in ERM. In particular, how we can estimate the expected loss $L(f)$ of a predictor f. We begin by introducing estimation through the simpler example of estimating the mean of coin flip. Then, we show how this can be generalized to estimating the expected loss of a predictor.

5.1 Estimating the Mean of a Coin Flip

Suppose you have an unfair coin that lands on heads with probability α. This can be modeled as a random variable $X \in \{0, 1\}$ where $X = 1$ if the coin lands on heads and $X = 0$ if the coin lands on tails. X is a Bernoulli random variable with parameter α. This means its probability mass function is given by

$$p(x) = \begin{cases} \alpha & \text{if } x = 1 \\ 1 - \alpha & \text{if } x = 0 \end{cases}$$

Since X is a random variable, we can compute its expected value, which is given by

$$\mathbb{E}[X] = 1 \cdot \alpha + 0 \cdot (1 - \alpha) = \alpha$$

This means that the expected value of X is equal to the probability that the coin lands on heads. Now suppose we do not know the value of α. The question becomes: how can we estimate α? One way to do this is to flip the coin n times and take the average of the outcomes. Formally this idea can be expressed as follows. Let $Z = (X_1, \ldots, X_n) \in \{0, 1\}^n$ be multivariate random variable representing the outcomes of n coin flips. Each X_i is to be thought of as representing a single flip of the same biased coin. This means each X_i is independent and identically distributed (i.i.d.) with the same distribution Bernoulli(α).

The estimate of α using an average of the n coin flips can be written as a function

$$g(Z) = g(X_1, \ldots, X_n) = \frac{1}{n} \sum_{i=1}^{n} X_i$$

This function g is called the *sample mean* estimate of α. Notice how g is a function of the random variable Z, thus it is also a random variable. It is common to use the notation $\bar{X}$ to denote the sample mean estimate $g(Z)$. Another convention is to use a hat to denote the estimate of a parameter. In this case we would write $\hat{\alpha} = g(Z)$, to denote that $\hat{\alpha}$ is the estimate of α. In particular,

$$\hat{\alpha} = \bar{X} = g(Z),$$

and we will use this notation interchangeably. If we were to change the quantity we are estimating, we would change the symbol the hat is applied to accordingly.

Since our estimate $\hat{\alpha}$ is a random variable, we can take its expected value. This is given by

$$\mathbb{E}[\hat{\alpha}] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X] = \frac{1}{n}\sum_{i=1}^{n}\alpha = \alpha.$$

The second equality follows from the fact that the expected value of a constant times a random variable is the constant times the expected value of the random variable (Property 1). The third equality follows from the linearity of expectation (Property 2). The fourth equality follows from the fact that each X_i is identically distributed, which implies that $\mathbb{E}[X_i] = \mathbb{E}[X]$. This means that the expected value of our estimate $\hat{\alpha}$ is equal to the true value of α.

To get some intuition of what this means, consider the following example.

Example 5.1: Let everything be as defined above with $\alpha = 4/5$. Suppose we flip the coin $n = 5$ times and call this an experiment. Each time we perform this experiment, we get a specific outcome of the coin flips. We can of course perform this experiment multiple times. Suppose we perform the experiment 4 times and get the following outcomes:

Experiment	Outcome z	Estimate $\hat{\alpha}$
1	$(1, 0, 1, 1, 1)$	$4/5$
2	$(0, 1, 0, 1, 1)$	$3/5$
3	$(1, 1, 1, 1, 1)$	$5/5$
4	$(0, 1, 0, 0, 0)$	$1/5$

Notice how for a particular experiment, the estimate $\hat{\alpha}$ can be different. What the expected value of $\hat{\alpha}$ tells us is that if we were to perform the experiment an infinite number of times, the average of all the estimates would be equal to the true value of α.

You might have noticed that the estimate $\hat{\alpha}$ is equal to $g(z)$ here, which is not a random variable since it is a function of the specific outcome z. This highlights an important abuse of notation. In particular, it is common to use the same notation $\hat{\alpha}$ to denote both the random variable $g(Z)$ and the specific value $g(z)$. The context should make it clear which one is being referred to. $\square$

The above example illustrates how each outcome of n coin flips can vary from one experiment to another. This naturally leads us to calculating the variance of our estimate $\hat{\alpha}$. The variance of $\hat{\alpha}$ is given by

$$\text{Var}[\hat{\alpha}] = \text{Var}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n^2}\text{Var}\left[\sum_{i=1}^{n} X_i\right] = \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}[X_i] = \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}[X] = \frac{1}{n}\text{Var}[X].$$

The second equality follows from the fact that the variance of a constant times a random variable is the constant squared times the variance of the random variable (Property 5). The third equality follows from the fact that the X_i are independent, so the variance of the sum is the sum of the variances (Property 7). The fourth equality follows from the fact that the X_i are identically distributed, which implies that $\text{Var}[X_i] = \text{Var}[X]$.

This result tells us that the variance of our estimate $\hat{\alpha}$ decreases as we increase the number of coin flips n. This is intuitive since the more coin flips we have, the more information we have about the true value of α. An important thing to notice is that none of the above results were specific to the Bernoulli distribution. In fact, the same results hold for any distribution, as long as the n random variables are independent and identically distributed. We will now generalize this to estimating the expected loss of a predictor.

5.2 Estimating the Expected Loss of a Predictor

To show how we can estimate the expected loss of a predictor, we will follow nearly the same steps as for the coin flip example. Let $(\boldsymbol{X}, Y) \in \mathcal{X} \times \mathcal{Y}$ be a random variable representing a feature-label pair, with distribution $\mathbb{P}_{\boldsymbol{X},Y}$. If we have a predictor $f : \mathcal{X} \to \mathcal{Y}$, then $f(\boldsymbol{X})$ is also a random variable. Similarly, for a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, the loss $\ell(f(\boldsymbol{X}), Y)$ is a random variable. Recall that we do not know the expected loss $L(f) = \mathbb{E}[\ell(f(\boldsymbol{X}), Y)]$. Thus, we would like to estimate it. To do this we can use the sample mean estimate based on n feature-label pairs.

Formally, let $D = ((\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)) \in (\mathcal{X} \times \mathcal{Y})^n$ be a random variable representing a dataset of n feature-label pairs. As we have discussed in the last chapter, we assume that each $(\boldsymbol{X}_i, Y_i)$ is independent and identically distributed with distribution $\mathbb{P}_{\boldsymbol{X},Y}$. The sample mean estimate of $L(f)$ is given by

$$\hat{L}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\boldsymbol{X}_i), Y_i).$$

As discussed in the previous section, this expected value and variance of $\hat{L}(f)$ is

$$\mathbb{E}[\hat{L}(f)] = L(f), \quad \text{and} \quad \mathrm{Var}[\hat{L}(f)] = \frac{1}{n}\mathrm{Var}[\ell(f(\boldsymbol{X}), Y)].$$

This shows that the estimate $\hat{L}(f)$ has some desirable properties. In particular, the expected value of $\hat{L}(f)$ is equal to the true value of $L(f)$, and the variance of $\hat{L}(f)$ decreases as we increase the number of feature-label pairs n. This means that as we get more data, our estimate of the expected loss of a predictor becomes more accurate.

Notice that the relationship to the coin flip example is quite clear. In particular the feature-label pairs random loss $\ell(f(\boldsymbol{X}_i), Y_i)$ is analogous to the coin flip random variable X_i. The expected loss $L(f)$ is analogous to the probability of the coin landing on heads α, and the sample mean estimate $\hat{L}(f)$ is analogous to the sample mean estimate $\hat{\alpha}$.

Notice that so far we have considered the expected loss of a single predictor f. Recall that in ERM we need to an estimate of $L(f)$ for all the predictors f in our function class $\mathcal{F}$. The question then becomes: if we use $\hat{L}(f)$ based on the same dataset D, then is it a good estimate of $L(f)$ for all $f \in \mathcal{F}$? The answer to this not so simple and it will depend on how large n is and how complex the function class $\mathcal{F}$ is. This is a topic we will explore in Chapter 7. For now, we will assume that $\hat{L}(f)$ is a good estimate of $L(f)$ for all $f \in \mathcal{F}$, and move on to the optimization step of ERM in the next chapter.

Example 5.2: Suppose we are working with features representing the number of rooms in a house and labels representing the price of the house. Let $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_5, y_5)\} = ((2, 300), (1, 150), (3, 400), (2, 200), (1, 100))$ be a fixed dataset of 5 feature-label pairs. Let $\ell(\hat{y}, y) = (\hat{y} - y)^2$ be the squared loss function. Suppose we have two different predictors

$f_1(\mathbf{x}) = 100\mathbf{x}$ and $f_2(\mathbf{x}) = 200\mathbf{x}$. We can calculate the expected loss of each predictor using the dataset $\mathcal{D}$.

$$\begin{aligned}
\hat{L}(f_1) &= \frac{1}{5}\sum_{i=1}^{5}(100\mathbf{x}_i - y_i)^2 \\
&= \frac{1}{5}[(200-300)^2 + (100-150)^2 + (300-400)^2 + (200-200)^2 + (100-100)^2] \\
&= 4500.
\end{aligned}$$

$$\begin{aligned}
\hat{L}(f_2) &= \frac{1}{5}\sum_{i=1}^{5}(200\mathbf{x}_i - y_i)^2 \\
&= \frac{1}{5}[(400-300)^2 + (200-150)^2 + (600-400)^2 + (400-200)^2 + (200-100)^2] \\
&= 20500.
\end{aligned}$$

This shows that the expected loss of f_1 is lower than the expected loss of f_2. If we were using ERM and the function class $\mathcal{F}$ only contained f_1 and f_2, then we would choose f_1 as our predictor $\hat{f}$. $\square$

Chapter 6

Optimization

Optimization is the process of finding the best solution from a set of possible solutions. Usually this means minimizing or maximizing some function. In this chapter our goal will be to introduce some ways of performing the optimization step in ERM. In particular, we will discuss how we can find the predictor $f \in \mathcal{F}$ that minimizes $\hat{L}(f)$, which is our estimate of the expected loss $L(f)$. Similar to last chapter, we will begin with some examples of minimizing simpler functions and then move on to minimizing $\hat{L}(f)$.

6.1 Optimization Basics

Let us start by considering the problem of minimizing a function $g : \mathcal{W} \to \mathbb{R}$. The minimum value of $g(w)$ over all $w \in \mathcal{W}$ is denoted by

$$\min_{w \in \mathcal{W}} g(w).$$

If we want the $w \in \mathcal{W}$ that achieves this minimum, we can write

$$w^* = \operatorname*{argmin}_{w \in \mathcal{W}} g(w).$$

The value w^* is called the *minimizer* of $g(w)$. Clearly, $\min_{w \in \mathcal{W}} g(w) = g(w^*)$.

Example 6.1: Let $g(w) = w^2$ and $\mathcal{W} = \mathbb{R}$. The function g is shown in Fig. 6.1. Visually we can see that $\min_{w \in \mathcal{W}} g(w) = 0$ (shown as a purple point in the figure) and

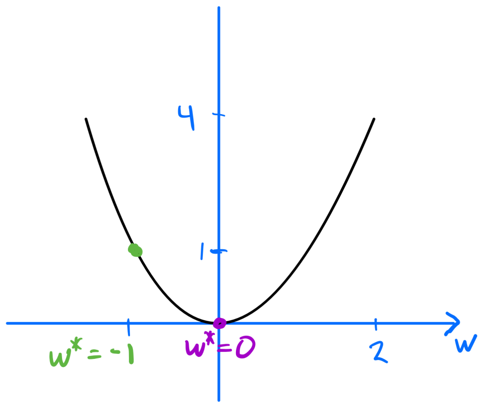


Figure 6.1: The function $g(w) = w^2$.

$w^* = \operatorname{argmin}_{w \in \mathcal{W}} g(w) = 0$. Notice that $g(w^*) = g(0) = 0 = \min_{w \in \mathcal{W}} g(w)$.

Suppose we changed the domain to $\mathcal{W} = \{-1, 2\}$. Then $\min_{w \in \mathcal{W}} g(w) = 1$ (shown as a green point in the figure) and $w^* = -1$. Again, $g(w^*) = g(-1) = 1 = \min_{w \in \mathcal{W}} g(w)$. However, notice that changing the domain $\mathcal{W}$ from $\mathbb{R}$ to $\{-1, 2\}$ changed the minimizer w^* and the minimum value. This example highlights the importance of the domain $\mathcal{W}$ when performing optimization. □

So far we have only spoken about minimizing functions. The reason for this is that there is a relationship between minimizing a function and maximizing a function. The minimizer of $g(w)$ is the maximizer of $-g(w)$ and vice versa. Formally, this means that

$$w^* = \operatorname*{argmin}_{w \in \mathcal{W}} g(w) = \operatorname*{argmax}_{w \in \mathcal{W}} -g(w).$$

This can be seen in both the plots in Fig. 6.2. Thus, if we want the maximizer of a function

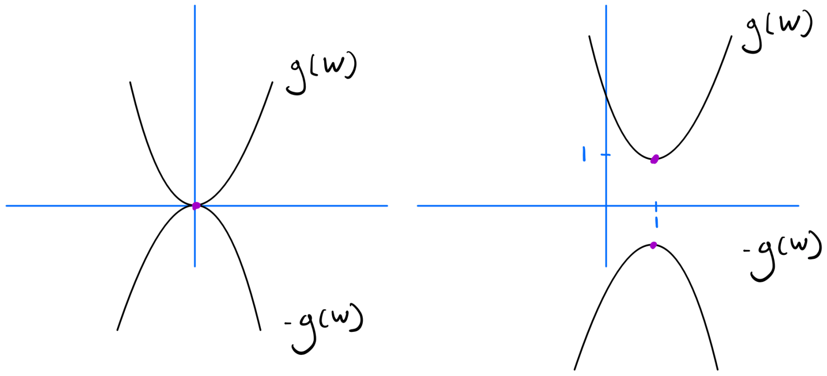


Figure 6.2: Left: the function $g(w) = w^2$ and $-g(w) = -w^2$. Right: the function $g(w) = (w-1)^2 + 1$ and $-g(w) = -(w-1)^2 - 1$.

$h(w)$ we can define $-g(w) = h(w)$ and find the minimizer of $g(w)$.

Notice, however, that the minimum value of $g(w)$ is not necessarily the maximum value of $-g(w)$. For example, consider the function $g(w) = (w-1)^2 + 1$, which is shown on the right in Fig. 6.2. The minimum value of $g(w)$ is 1 and the minimizer is $w^* = 1$. However, the maximum value of $-g(w)$ is -1 and the maximizer is $w^* = 1$. Thus, the minimum value of $g(w)$ is not the maximum value of $-g(w)$. It can be shown that instead, the minimum value of $g(w)$ is the negative of the maximum value of $-g(w)$. Formally, this means that

$$\min_{w \in \mathcal{W}} g(w) = -\left(\max_{w \in \mathcal{W}} -g(w)\right).$$

This should not be a surprise. Since w^* is the minimizer of $g(w)$ and maximizer of $-g(w)$, it implies that $g(w^*) = \min_{w \in \mathcal{W}} g(w)$ and $-g(w^*) = \max_{w \in \mathcal{W}} -g(w)$. Thus, $\min_{w \in \mathcal{W}} g(w) = g(w^*) = -(-g(w^*)) = -(\max_{w \in \mathcal{W}} -g(w))$.

We have not yet discussed how to find the minimizer of a function. The process to do so depends on the function $g(w)$ and the domain $\mathcal{W}$. If $\mathcal{W}$ is discrete (not continuous), then we can simply evaluate $g(w)$ for all $w \in \mathcal{W}$ and find the w that achieves the minimum. We have already seen this in Example 6.1 when $\mathcal{W} = \{-1, 2\}$. If $\mathcal{W}$ is continuous, then we can sometimes use derivatives to find the minimizer. For the remainder of this section we will focus on the second case, as is typical in machine learning.

6.2 Continuous Optimization

In this section we will discuss how to find the minimizer of function $g(w)$ over a continuous domain $\mathcal{W}$. Some examples of continuous domains are $\mathcal{W} = \mathbb{R}$ or an interval $\mathcal{W} = [a, b]$. To find the minimizer of $g(w)$ we will have to make another assumption about $g(w)$. We will assume that $g(w)$ is convex and twice differentiable. A function $g(w)$ is twice differentiable if its second derivative exists for all $w \in \mathcal{W}$. A function $g(w)$ is convex it its second derivative is non-negative. Formally, $g(w)$ is convex if

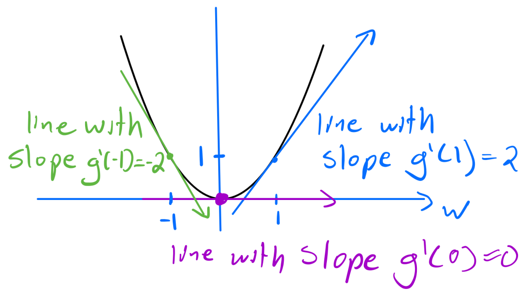
$$g''(w) \geq 0, \quad \forall w \in \mathcal{W}.$$

We also assumed that $g(w)$ is twice differentiable so that its second derivative exists and we can use it to determine if $g(w)$ is convex as we described above. Roughly speaking, a function is convex if it is bowl shaped. An example of a convex function is $g(w) = w^2$, which we have already seen in Example 6.1.

The reason we assume $g(w)$ is convex is because it makes finding the minimizer easier. In particular, if $g(w)$ is convex, then:

1. If $\mathcal{W} = \mathbb{R}$, the minimizer of $g(w)$ is the w that satisfies $g'(w) = 0$.
2. If $\mathcal{W} = [a, b]$, the minimizer of $g(w)$ is the $w \in \mathbb{R}$ that satisfies $g'(w) = 0$ if it is an element of $[a, b]$. Otherwise, the minimizer is a or b.

Example 6.2: [Convex with $\mathcal{W} = \mathbb{R}$] Let $g(w) = w^2$ and $\mathcal{W} = \mathbb{R}$. Then, $g'(w) = 2w$, and $g''(w) = 2$. Thus, $g(w)$ is convex, since $g''(w) \geq 0$. To find the minimizer of $g(w)$ we solve $g'(w) = 0$. This gives $2w = 0$ or $w = 0$. Thus, the minimizer of $g(w)$ is $w^* = 0$. This aligns with what we saw in Example 6.1.



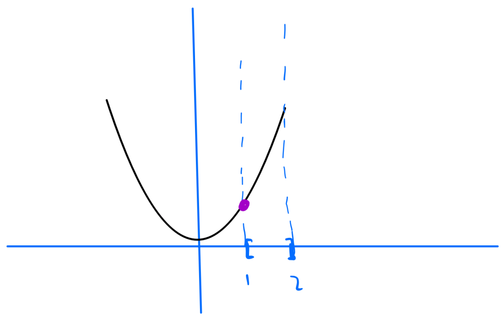


Figure 6.3: The function $g(w) = w^2$. Left: optimizing over $\mathcal{W} = \mathbb{R}$ and showing derivative $g'(w) = 2w$ at $w = -1, 0, 1$. Right: optimizing over $\mathcal{W} = [1, 2]$.

The plot on the left in Fig. 6.3 shows a visual interpretation of this example. The derivative $g'(w) = 2w$ can be visualized as a line with slope $2w$ passing through the point $g(w)$. For $w = -1$ the slope is -2 and for $w = 1$ the slope is 2, which are shown as the green and blue lines in the plot respectively. While, for $w = 0$ the slope is 0 and is shown as the purple line in the plot. Notice that when the derivative is 0 the function is at a minimum, which is why we solve $g'(w) = 0$ to find the minimizer.

We can also use the left plot to build some intuition about the second derivative. The second derivative $g''(w) = 2$ represents the constant rate of change of the derivative $g'(w) = 2w$. Since $g''(w) \geq 0$, the derivative $g'(w)$ is increasing linearly with w. In the plot, we can see this: at $w = -1$, $g'(w) = -2$; at $w = 0$, $g'(w) = 0$; and at $w = 1$, $g'(w) = 2$. The constant positive second derivative indicates that the function $g(w)$ is convex (bowl-shaped) everywhere. □

Example 6.3: [Convex with $\mathcal{W} = [1, 2]$] Now let $g(w) = w^2$ and $\mathcal{W} = [1, 2]$. From the previous example we know that the minimizer of $g(w)$ is $w^* = 0$. However, $w^* = 0$ is not in the domain $\mathcal{W} = [1, 2]$. Thus, we must consider the endpoints of the domain. Since $g(1) = 1$ and $g(2) = 4$, we know that the minimizer of $g(w)$ is $w^* = 1$. The plot on the right in Fig. 6.3 shows a visual interpretation of this example. □

Example 6.4: [Non-convex] Let $g(w) = w^3$ and $\mathcal{W} = \mathbb{R}$. Then, $g'(w) = 3w^2$ and $g''(w) = 6w$. Since $g''(w) = 6w$ is negative for $w < 0$, $g(w)$ is not convex. Thus, we cannot use the derivative to find the minimizer of $g(w)$.

However, suppose we tried to use the derivative to find the minimizer. We would solve $g'(w) = 0$ to get $3w^2 = 0$ or $w = 0$. This is visualized in the plot in Fig. 6.4. We can see that $w = 0$ is not the minimizer of $g(w)$. □

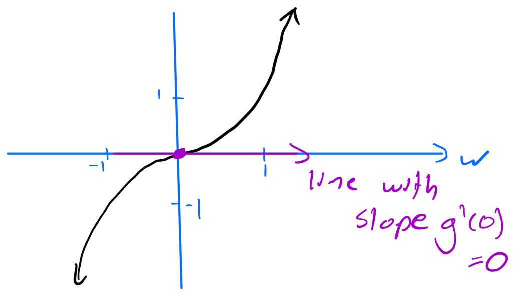


Figure 6.4: The function $g(w) = w^3$.

Example 6.5: [Complex Non-convex] Let $\mathcal{W} = \mathbb{R}$ and define $g(w)$ as the function in the plot in Fig. 6.5. We use this example to illustrate that one can visually check that the function is not convex. Intuitively, we can see that the function is not bowl shaped. Formally, since the function has multiple minimizers and maximizers (shown as the purple points), it's second derivative must be negative at some points. Thus, the function is not convex. □

6.2.1 Optimization Over Multidimensional Domains

In this section we will discuss how to find the minimizer of a function $g(\mathbf{w})$ over a multi-dimensional domain $\mathcal{W}$ such as $\mathbb{R}^d$. Again, we will assume that $g(\mathbf{w})$ is convex and twice differentiable. Checking if $g(\mathbf{w})$ is convex is more complicated in higher dimensions. In these notes we will always indicate if a function is convex or not over a multidimensional

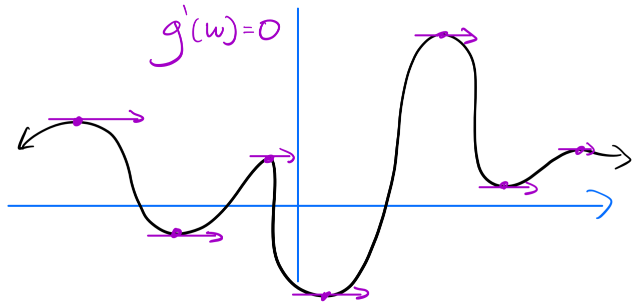


Figure 6.5: A complicated non-convex function.

domain. The minimizer of $g(\mathbf{w})$ is written as

$$\mathbf{w}^* = (w_1^*, \ldots, w_d^*) = \operatorname*{argmin}_{\mathbf{w} \in \mathcal{W}} g(\mathbf{w}).$$

If $\mathcal{W} = \mathbb{R}^d$, then each w_j^* can be found by solving $\frac{\partial g}{\partial w_j}(\mathbf{w}) = 0$ for all $j \in \{1, \ldots, d\}$. This should remind you of the one-dimensional case where we solved $g'(w) = 0$, except now we have to solve d equations.

Example 6.6: Let $g(\mathbf{w}) = w_1^2 + w_2^2$ and $\mathcal{W} = \mathbb{R}^2$. Then, $\frac{\partial g}{\partial w_1}(\mathbf{w}) = 2w_1$ and $\frac{\partial g}{\partial w_2}(\mathbf{w}) = 2w_2$. To find the minimizer of $g(\mathbf{w})$ we solve $\frac{\partial g}{\partial w_1}(\mathbf{w}) = 0$ and $\frac{\partial g}{\partial w_2}(\mathbf{w}) = 0$. This gives $w_1 = 0$ and $w_2 = 0$. Thus, the minimizer of $g(\mathbf{w})$ is $\mathbf{w}^* = (0, 0)$. The function $g(\mathbf{w})$ is shown in Fig. 6.6. It is again a bowl shaped function (since it is convex), and its minimizer is at the bottom of the bowl. □

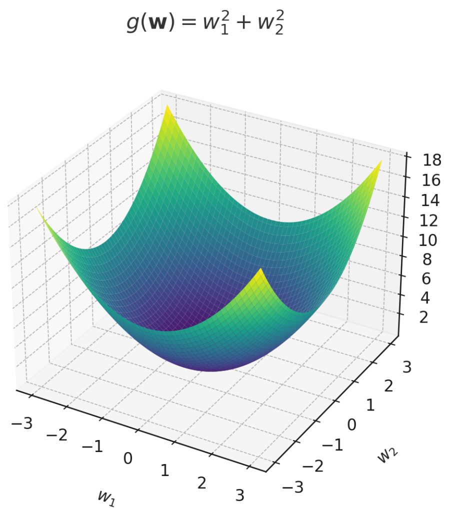


Figure 6.6: The function $g(\mathbf{w}) = w_1^2 + w_2^2$.

6.3 Closed Form Solution for Linear Regression

Recall that the optimization step of ERM was to find the predictor $f \in \mathcal{F}$ that minimizes $\hat{L}(f)$. Using the notation from the previous sections we can write this as

$$\hat{f} = \operatorname*{argmin}_{f \in \mathcal{F}} \hat{L}(f) \quad \text{where} \quad \hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i).$$

Notice the similarity between this and the minimization of $g(w)$ over $w \in \mathcal{W}$. Now $g(w) = \hat{L}(f)$ and $\mathcal{W} = \mathcal{F}$. If $\hat{L}(f)$ is convex and twice differentiable, and $\mathcal{F}$ is continuous, then we can use the techniques from the previous sections to find $\hat{f}$. Next, we try to select $\mathcal{F}, \ell, \mathcal{X}, \mathcal{Y}$, such that this is the case.

We will focus on optimization in the regression setting. In particular, we will consider $\mathcal{Y} = \mathbb{R}$. The set of feature vectors $\mathcal{X}$ will be $\mathbb{R}^d$. We assume we are given a fixed dataset $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)) \in (\mathcal{X} \times \mathcal{Y})^n$. We have to select a function class $\mathcal{F}$, which contains all the predictors we will be optimizing over. We will select $\mathcal{F}$ to be the set of linear functions, hence the name *linear regression.* Formally, the function class is

$$\mathcal{F} = \{f | f : \mathcal{X} \to \mathcal{Y} \text{ and } f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b \text{ for some } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}.$$

In words, $\mathcal{F}$ contains all the predictors of the form $f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b$ for some $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$. The term $\mathbf{w}$ is called the *weight vector* or the *weights* and b is called the *bias.* The bias is a constant that shifts the prediction up or down. Take the case of $d = 1$. Then, the predictor is $f(x) = wx + b$. The term b shifts the line up or down, while w controls the slope of the line.

Writing the bias as a seperate term is often mathematically inconvenient. As such, it is common to include the bias as an extra dimension in the weight vector $\mathbf{w}$. Formally, we can write $\mathbf{w} = (w_0, w_1, \ldots, w_d)$, where we have used w_0 to represent the bias. We must also change the feature vector by adding a 1 to the beginning. Formally, $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_d)$. Then, the predictor is $f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} = w_0 + w_1x_1 + \ldots + w_dx_d$. Notice how the predictor is the same as before, but now it can be written in a more compact form as just the dot product of $\mathbf{x}$ and $\mathbf{w}$. Now $\mathbf{w}$ is in $\mathbb{R}^{d+1}$ instead of $\mathbb{R}^d$, and we will simply refer to it as the weight vector. Similarly $\mathbf{x}$ is in $\mathbb{R}^{d+1}$ instead of $\mathbb{R}^d$. It is important to note that now $\mathcal{X} = \mathbb{R}^{d+1}$ and whenever we write $\mathbf{x} \in \mathcal{X}$ we always assume that $\mathbf{x}$ is of the form $(1, x_1, \ldots, x_d)$ (i.e. the first element is 1). Similarly, all the feature vectors $\mathbf{x}_i$ in the dataset $\mathcal{D}$ are of the form $(x_{i0} = 1, x_{i1}, \ldots, x_{id}) \in \mathbb{R}^{d+1}$ for $i \in \{1, \ldots, n\}$. With this change the function class $\mathcal{F}$ is now

$$\mathcal{F} = \{f | f : \mathcal{X} \to \mathcal{Y} \text{ and } f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} \text{ for some } \mathbf{w} \in \mathbb{R}^{d+1}\}.$$

In the previous sections we only discussed optimization over domains $\mathcal{W}$ that contained numbers or vectors. However, $\mathcal{F}$ is a set of functions. We will address this difference by making a helpful observation. Notice that $\hat{f}$, which is the minimizer of $\hat{L}(f)$, is a function in $\mathcal{F}$. That means that $\hat{f} = \mathbf{x}^T\hat{\mathbf{w}}$ for some $\hat{\mathbf{w}} \in \mathbb{R}^{d+1}$. Thus, our goal can be restated as the optimization problem

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^{d+1}} \hat{L}(\mathbf{w}) \quad \text{where} \quad \hat{L}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{x}_i^\top\mathbf{w}, y_i).$$

We have abused notation slightly by passing a vector $\mathbf{w}$ to $\hat{L}$ instead of a function f and defining $\hat{L}(\mathbf{w})$ differently from $\hat{L}(f)$. We will, again, trust that the definition will be clear from the context. What is important in the above display is that the minimization is now over $\mathbb{R}^{d+1}$ instead of $\mathcal{F}$. If we set $\mathcal{W} = \mathbb{R}^{d+1}$ and $g(\mathbf{w}) = \hat{L}(\mathbf{w})$, then this expression is of the same form as the optimization problems in Section 6.2.1.

What is left is to ensure that $\hat{L}(\mathbf{w})$ is convex and twice differentiable. We will not show this, but, if we select the squared loss function $\ell(\hat{y}, y) = (\hat{y} - y)^2$, then $\hat{L}(\mathbf{w})$ is convex and twice differentiable. This sets us up to use the techniques from Section 6.2.1 to find $\hat{\mathbf{w}}$.

6.3.1 Minimizing the Estimate of the Expected Loss

Our goal is to find $\hat{w} = (w_0, w_1, \ldots, w_d)$, which is the minimizer of $\hat{L}(\mathbf{w})$. We can do this by solving $\partial \hat{L}(\mathbf{w})/\partial w_j = 0$ for all $j \in \{0, \ldots, d\}$.

Let us begin by first writing out $\hat{L}(\mathbf{w})$.

$$\begin{aligned}\hat{L}(\mathbf{w}) &= \frac{1}{n}\sum_{i=1}^{n} \ell(\mathbf{x}_i^\top \mathbf{w}, y_i) \\ &= \frac{1}{n}\sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n} \left(x_{i0}w_0 + \ldots + x_{id}w_d - y_i\right)^2 .\end{aligned}$$

If we define $\hat{L}_i(\mathbf{w}) = (x_{i0}w_0 + \ldots + x_{id}w_d - y_i)^2$, then we can write

$$\hat{L}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \hat{L}_i(\mathbf{w}).$$

Now we are ready to take the partial derivative of $\hat{L}(\mathbf{w})$ with respect to w_j, for all $j \in \{0, \ldots, d\}$. Due to the linearity of the derivative (Section 2.6.2) we can take the derivative of each term in the sum separately to get

$$\frac{\partial \hat{L}}{\partial w_j}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial \hat{L}_i}{\partial w_j}(\mathbf{w}).$$

We now focus on the term $\partial \hat{L}_i(\mathbf{w})/\partial w_j$ where $i \in \{1, \ldots, n\}$. To help us we will use the chain rule (Eq. (2.2)), by setting $u_i = x_{i0}w_0 + \ldots + x_{id}w_d - y_i, g(u_i) = u_i^2$. Then,

$$\frac{\partial \hat{L}_i}{\partial w_j}(\mathbf{w}) = \frac{\partial g}{\partial u_i}\frac{\partial u_i}{\partial w_j}(\mathbf{w}).$$

Each of the partial derivatives on the right hand side can be computed as follows.

$$\frac{\partial g}{\partial u_i} = 2u_i, \quad \text{and} \quad \frac{\partial u_i}{\partial w_j} = x_{ij}.$$

Using these two derivatives and plugging in the definition of u_i we get

$$\frac{\partial \hat{L}_i}{\partial w_j}(\mathbf{w}) = 2(x_{i0}w_0 + \ldots + x_{id}w_d - y_i)x_{ij} = 2x_{ij}\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right).$$

We can now plug this result back into the expression for $\partial \hat{L}/\partial w_j$ to get

$$\frac{\partial \hat{L}}{\partial w_j}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} 2x_{ij}\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right).$$

Recall that we need to set this derivative to 0 for each $j \in \{0, \ldots, d\}$. Doing this and rearranging for a specific $j \in \{0, \ldots, d\}$, we have

$$\frac{1}{n}\sum_{i=1}^{n} 2x_{ij}\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right) = 0 \quad\implies\quad \sum_{i=1}^{n} x_{ij}\mathbf{x}_i^\top \mathbf{w} = \sum_{i=1}^{n} x_{ij}y_i.$$

Thus we have $d+1$ equations, one for each $j \in \{0, \ldots, d\}$. We can explicitly write out these equations as

$$\begin{gathered}\sum_{i=1}^{n} x_{i0}\mathbf{x}_i^\top \mathbf{w} = \sum_{i=1}^{n} x_{i0}y_i \\ \vdots \\ \sum_{i=1}^{n} x_{id}\mathbf{x}_i^\top \mathbf{w} = \sum_{i=1}^{n} x_{id}y_i.\end{gathered}$$

To solve this system of equations it will be helpful to write it in matrix form.

$$\begin{bmatrix} \sum_{i=1}^n x_{i0}x_{i0} & \sum_{i=1}^n x_{i0}x_{i1} & \cdots & \sum_{i=1}^n x_{i0}x_{id} \\ \sum_{i=1}^n x_{i1}x_{i0} & \sum_{i=1}^n x_{i1}x_{i1} & \cdots & \sum_{i=1}^n x_{i1}x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_{id}x_{i0} & \sum_{i=1}^n x_{id}x_{i1} & \cdots & \sum_{i=1}^n x_{id}x_{id} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n x_{0i}y_i \\ \sum_{i=1}^n x_{1i}y_i \\ \vdots \\ \sum_{i=1}^n x_{di}y_i \end{bmatrix}$$

$$\mathbf{A}\mathbf{w} = \mathbf{b}.$$

Here, $\mathbf{A}_{jk} = \sum_{i=1}^n x_{ij}x_{ik}$ is the (j,k)-th element of the matrix $\mathbf{A}$, and $b_j = \sum_{i=1}^n x_{ij}y_i$ is the j-th element of the vector $\mathbf{b}$. Similarly, w_j is the j-th element of the weight vector $\mathbf{w}$.

When implementing this in code, it can be helpful to write the matrix $\mathbf{A}$ and vector $\mathbf{b}$ in terms of the feature vectors $\mathbf{x}_i$. This can be done by writing $\mathbf{A} = \sum_{i=1}^n \mathbf{x}_i\mathbf{x}_i^\top$ and $\mathbf{b} = \sum_{i=1}^n \mathbf{x}_i y_i$. Here, $\mathbf{x}_i\mathbf{x}_i^\top$ is something called the outer product of $\mathbf{x}_i$ and $\mathbf{x}_i$ and is a $d+1$ by $d+1$ matrix defined as

$$\mathbf{x}_i\mathbf{x}_i^\top = \begin{bmatrix} x_{i0} \\ x_{i1} \\ \vdots \\ x_{id} \end{bmatrix} \begin{bmatrix} x_{i0} & x_{i1} & \cdots & x_{id} \end{bmatrix} = \begin{bmatrix} x_{i0}^2 & x_{i0}x_{i1} & \cdots & x_{i0}x_{id} \\ x_{i1}x_{i0} & x_{i1}^2 & \cdots & x_{i1}x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ x_{id}x_{i0} & x_{id}x_{i1} & \cdots & x_{id}^2 \end{bmatrix}.$$

The important thing is that we have reduced the problem of finding $\hat{\mathbf{w}}$ to solving the linear system $\mathbf{A}\mathbf{w} = \mathbf{b}$. This system of equations has the following closed form solution.

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{b},$$

if $\mathbf{A}$ is invertible, where $\mathbf{A}^{-1}$ is the inverse of $\mathbf{A}$. This is called the closed form solution for linear regression. When implementing this in code, there are standard libraries that can be used to compute the inverse of a matrix.

Thus, the weight vector $\hat{\mathbf{w}}$ that minimizes $\hat{L}(\mathbf{w})$ is $\hat{\mathbf{w}} = \mathbf{A}^{-1}\mathbf{b}$.

6.3.2 Defining the Learner

Now that we have covered the estimation and optimization steps of ERM, we can define the learner. Recall that learner $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \mathcal{F}$ is a function that takes as input a dataset $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)) \in (\mathcal{X} \times \mathcal{Y})^n$ and outputs a predictor $f \in \mathcal{F}$. For the case of linear regression, with things as defined above, the defintion of the learner becomes quite simple. In particular, the learner is defined as

$$\mathcal{A}(\mathcal{D}) = \hat{f} \quad \text{where} \quad \hat{f}(\mathbf{x}) = \mathbf{x}^T \hat{\mathbf{w}} \quad \text{and} \quad \hat{\mathbf{w}} = \mathbf{A}^{-1}\mathbf{b}.$$

Example 6.7: Suppose we are interested in predicting house prices based on the number of rooms in a house. In this case $d = 1$, $\mathcal{X} = \mathbb{R}^{1+1}$, and $\mathcal{Y} = \mathbb{R}$. We are given a dataset $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$ where $\mathbf{x}_i = (x_{i0} = 1, x_{i1})$ and y_i is the price of the i-th house. Our ERM learner $\mathcal{A}(\mathcal{D})$ will output a predictor $\hat{f}(\mathbf{x}) = \mathbf{x}^T\hat{\mathbf{w}}$, where $\hat{\mathbf{w}} = \mathbf{A}^{-1}\mathbf{b}$. This predictor $\hat{f}$ will be a line that best fits the data. A sketch of what $\hat{f}$ might look like is shown in Fig. 6.7. Each black point in the plot represent a feature-label pair from the dataset $\mathcal{D}$. □

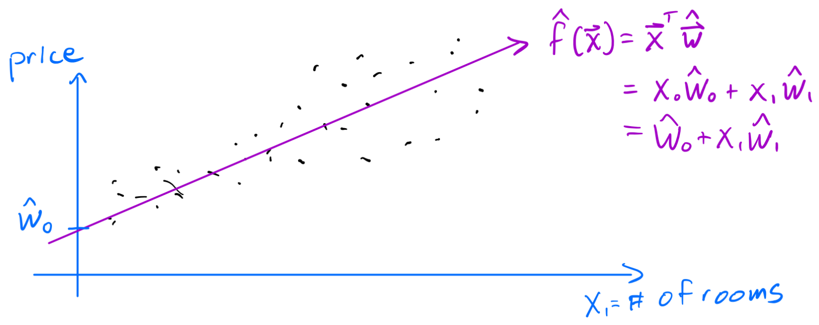


Figure 6.7: The best fit line for a synthetic house price dataset.

In pseudocode, the learner $\mathcal{A}$ can be written as

Algorithm 1: Closed Form Linear Regression Learner

```
1: input: $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$
2: $\mathbf{A} \leftarrow \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top$
3: $\mathbf{b} \leftarrow \sum_{i=1}^{n} \mathbf{x}_i y_i$
4: $\hat{\mathbf{w}} \leftarrow \mathbf{A}^{-1}\mathbf{b}$
5: return $\hat{f}(\mathbf{x}) = \mathbf{x}^T \hat{\mathbf{w}}$
```

6.4 Optimization with Gradient Descent

To motivate and introduce gradient descent we take a step back from machine learning and consider finding the minimizer of a function $g : \mathcal{W} \to \mathbb{R}$.

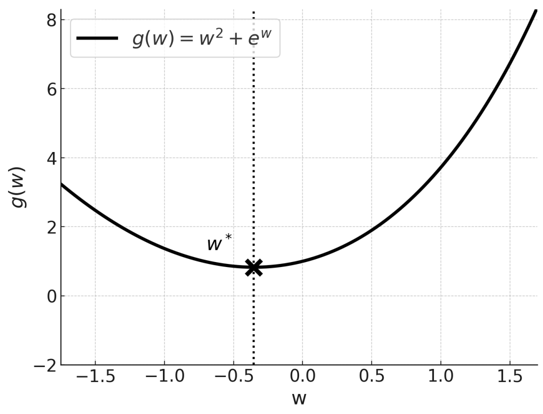


Figure 6.8: The function $g(w) = w^2 + e^w$.

Example 6.8: Let $g(w) = w^2 + e^w$ and $w \in \mathcal{W} = \mathbb{R}$. The function is shown in Fig. 6.8. We want to find $w^* = \operatorname{argmin}_{w \in \mathbb{R}} g(w)$, which is depicted as the black X in the plot. We would like to use the optimization techniques from Section 6.2, so we check if $g(w)$ is convex. The first derivative of $g(w)$ is $g'(w) = 2w + e^w$ and the second derivative is $g''(w) = 2 + e^w$. Since $g''(w) = 2 + e^w$ is always positive, $g(w)$ is convex. Alternatively, we could have just looked at the plot and seen that $g(w)$ is bowl shaped.

Now, we can find the minimizer of $g(w)$ by solving $g'(w) = 0$. This gives $2w + e^w = 0$ or $2w = -e^w$, which does not have a closed form solution[1]. Although, we can see from the plot that $w^* \approx -0.35$ we do not have a mathematical expression for w^*. □

The above example highlights that in general we cannot find the minimizer of a function $g(w)$ in closed form. This is where gradient descent comes in.

6.4.1 Second-Order Gradient Descent (Newton's Method)

Assume that $g(w)$ is convex. Since $w^* = \operatorname{argmin}_{w \in \mathcal{W}} g(w)$ does not have a closed form solution in general, we can instead approximate $g(w)$ with a convex function that does have a closed form solution. A quadratic function can be chosen to always be convex and always has a closed form solution. Thus, we will use a quadratic function to approximate $g(w)$. To do this we will use the second-order Taylor expansion of $g(w)$ at a point w_0, given by

$$g_{w^{(0)}}(w) = g(w^{(0)}) + g'(w^{(0)})(w - w^{(0)}) + \frac{1}{2} g''(w^{(0)})(w - w^{(0)})^2.$$

We have that $g(w) \approx g_{w^{(0)}}(w)$ when w is close to $w^{(0)}$.

Example 6.9: With things defined as in Example 6.8, we can use the second-order Taylor expansion to approximate $g(w)$. We choose $w^{(0)} = 1$. Then,

$$g_{w^{(0)}}(w) = g(1) + g'(1)(w - 1) + \frac{1}{2} g''(1)(w - 1)^2.$$

[1]A closed form solution is an expression that only uses a finite combination of elementary functions. Elementary functions include polynomials, exponentials, logarithms, trigonometric functions, and their inverses.

Calculating $g(1) = 1 + e$, $g'(1) = 2 + e$, and $g''(1) = 2 + e$, we get

$$g_{w^{(0)}}(w) = 1 + e + (2 + e)(w - 1) + \frac{1}{2}(2 + e)(w - 1)^2 = \left(1 + \frac{1}{2}e\right) w^2 + \frac{1}{2}e.$$

The function $g_{w^{(0)}}(w)$ is shown in Fig. 6.9. Notice how $g_{w^{(0)}}(w)$ is a quadratic function and

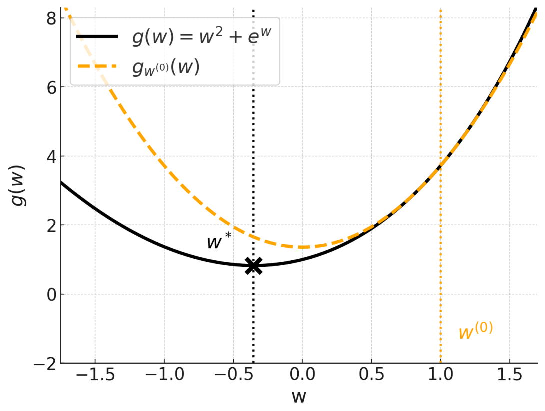


Figure 6.9: The functions $g(w)$ and $g_{w^{(0)}}(w)$.

is a good approximation of $g(w)$ near $w = w^{(0)} = 1$. □

Recall that we are interested in finding the minimizer of $g(w)$. Since $g(w) \approx g_{w^{(0)}}(w)$, the minimizer of $g_{w^{(0)}}(w)$ will be close to the minimizer of $g(w)$. We can find the minimizer of $g_{w^{(0)}}(w)$ by solving $g'_{w^{(0)}}(w) = 0$, since $g_{w^{(0)}}(w)$ is convex. The first derivative of $g_{w^{(0)}}(w)$ is

$$g'_{w^{(0)}}(w) = g'(w^{(0)}) + g''(w^{(0)})(w - w^{(0)}).$$

Setting this to 0 and solving for w gives

$$w = w^{(0)} - \frac{g'(w^{(0)})}{g''(w^{(0)})}.$$

For notational purposes which we will see later, it will be helpful to rename w to $w^{(1)}$. We expect $w^{(1)} \approx w^*$.

Example 6.10: With things defined as in Example 6.9, we can find $w^{(1)}$. Plugging values into $w^{(1)} = w^{(0)} - g'(w^{(0)})/g''(w^{(0)})$ gives

$$w^{(1)} = 1 - \frac{2 + e}{2 + e} = 0.$$

From the plot in Fig. 6.10 we can see how good of an approximation $w^{(1)} = 0$ is to w^*. □

An important observation to make from the above example is that $w^{(1)}$ is closer to w^* than $w^{(0)}$. In general, this will be the case for any $w^{(0)}$ we choose. Thus, we can repeat the same process as above, but with a quadratic approximation $g_{w^{(1)}}$ centered at $w^{(1)}$ instead of $w^{(0)}$. The expression for $g_{w^{(1)}}(w)$ will be

$$g_{w^{(1)}}(w) = g(w^{(1)}) + g'(w^{(1)})(w - w^{(1)}) + \frac{1}{2}g''(w^{(1)})(w - w^{(1)})^2.$$

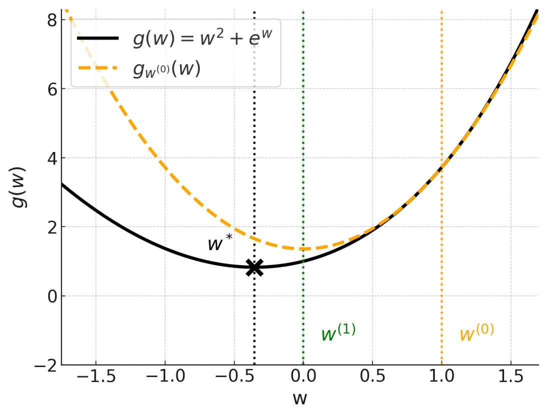


Figure 6.10: The functions $g(w)$ and $g_{w^{(0)}}(w)$ with $w^{(1)} = 0$.

Example 6.11: With things as defined in Example 6.10, $g_{w^{(1)}}(w)$, with $w^{(1)} = 1$ is

$$g_{w^{(1)}}(w) = g(0) + g'(0)w + \frac{1}{2}g''(0)w^2 = 1 + w + \frac{3}{2}w^2.$$

The function $g_{w^{(1)}}(w)$ is shown in Fig. 6.11. Similar to before, we can see that $g_{w^{(1)}}(w)$ is

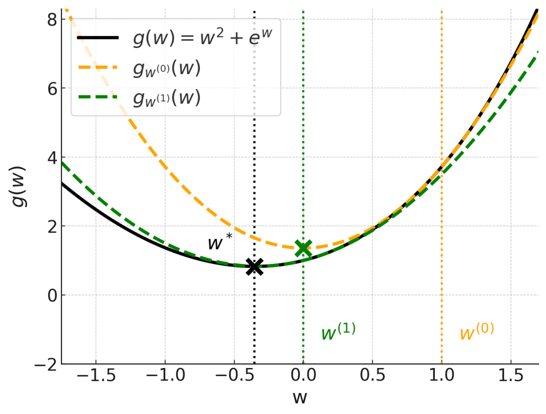


Figure 6.11: The functions $g(w), g_{w^{(0)}}(w)$ and $g_{w^{(1)}}(w)$.

a good approximation of $g(w)$ near $w = w^{(1)} = 0$. □

Solving for the minimizer of $g_{w^{(1)}}(w)$ will give us a new approximation $w^{(2)}$, which will be closer to w^* than $w^{(1)}$. The equation for $w^{(2)}$ will be

$$w^{(2)} = w^{(1)} - \frac{g'(w^{(1)})}{g''(w^{(1)})}.$$

Example 6.12: With things as defined in Example 6.11, we can find $w^{(2)}$. Plugging values into $w^{(2)} = w^{(1)} - g'(w^{(1)})/g''(w^{(1)})$ gives

$$w^{(2)} = 0 - \frac{1}{3} = -\frac{1}{3}.$$

From the plot in Fig. 6.12 we can see $w^{(2)}$ is a better approximation to w^* than $w^{(1)}$. $\square$

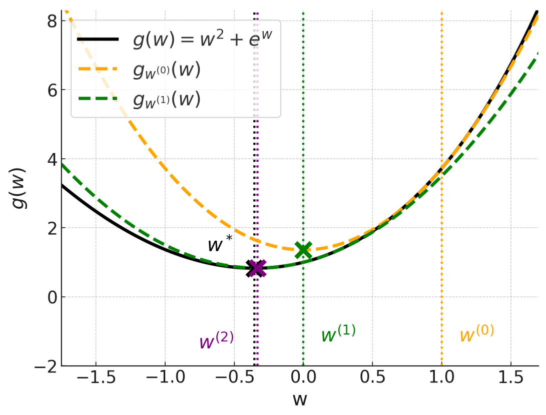


Figure 6.12: The functions $g(w)$ and $g_{w^{(2)}}(w)$ with $w^{(2)} = -1/3$.

We can repeat this process to get $w^{(3)}$, $w^{(4)}$, and so on. In general, the $(t+1)$-th iteration will be

$$w^{(t+1)} = w^{(t)} - \frac{g'(w^{(t)})}{g''(w^{(t)})}.$$

As t increases, $w^{(t)}$ will get closer to w^*. In practice, we would chose the number of iterations T and stop at $w^{(T)}$, or check that $|w^{(t)} - w^{(t-1)}| < \epsilon$ for some small ϵ and stop at t. Thus, we have a method to approximate the minimizer of $g(w)$, even when $g(w)$ does not have a closed form solution. This method is called Newton's method or second-order gradient descent. The name comes froms the fact that we are using a second order approximation of $g(w)$ and the gradient of $g(w)$ is the same as the derivative of $g(w)$, since it is a function of a single variable.

6.4.2 (First-Order) Gradient Descent

Second-order gradient descent is a powerful optimization technique, but it has a drawback. The drawback is that it requires the computation of the second derivative of $g(w)$. This can be computationally expensive, especially when $g(w)$ is a function of many variables. Recall that in second-order gradient descent the update rule is

$$w^{(t+1)} = w^{(t)} - \frac{g'(w^{(t)})}{g''(w^{(t)})}.$$

In first-order gradient descent we replace $1/g''(w^{(t)})$ with a time dependent constant $\eta^{(t)}$. This gives the update rule

$$w^{(t+1)} = w^{(t)} - \eta^{(t)} g'(w^{(t)}).$$

The constant $\eta^{(t)}$ is called the *step size* or *learning rate*. This allows us to avoid computing the second derivative of $g(w)$. However, the tradeoff is that if we select a bad $\eta^{(t)}$ the method may take longer to converge to w^*, or may not converge at all.

Remark: Notice that we call this method "first-order" gradient descent; however, we do not use a first order approximation of $g(w)$. Instead the term first-order here refers to the fact that we are only using the first derivative of $g(w)$. Often when people refer to gradient descent they are referring to first-order gradient descent. As such, we will use the term gradient descent to refer to first-order gradient descent for the remainder of these notes. This is also the reason we have put "first-order" in parentheses in the title of this section.

To help build some intuition behind the first-order gradient descent method, and the role of the step size we go over several examples. First, we begin by reinterpreting the examples from the previous section.

Example 6.13: Recall the function $g(w) = w^2 + e^w$ from Example 6.12. Its first derivative is $g'(w) = 2w + e^w$ and second derivative is $g''(w) = 2 + e^w$. As before we select $w^{(0)} = 1$. If we set $\eta^{(t)} = 1/g''(w^{(t)})$ then the second-order gradient descent update rule can be viewed as a first-order gradient descent update rule.

$$w^{(1)} = w^{(0)} - \eta^{(0)} g'(w^{(0)}).$$

Now, we provide an interpretation of this update rule, through the lense of taking steps in the negative direction of the gradient (which is the same as the derivative in the univariate case). Recall from Section 6.2 that the gradient $g'(w^{(0)})$ is the rate of change of $g(w)$ at $w^{(0)}$. Notice that when the gradient is positive, increasing w will increase $g(w)$, while when the gradient is negative, decreasing w will increase $g(w)$. Conveniently, the sign (or direction) of the gradient tells us if we should increase or decrease w to increase $g(w)$. However, we are interested in minimizing $g(w)$, so we want to decrease $g(w)$. As such, we want to take steps in the negative direction of the gradient (i.e. $-g'(w^{(0)})$). The step size $\eta^{(0)}$ determines how large of a step we take in the negative direction of the gradient.

Thus, we can interpret the first-order gradient descent update rule as starting from $w^{(0)}$ and taking a step of size $\eta^{(0)} g'(w^{(0)})$ in the negative direction of the gradient. Visually, we can see this on the left in Fig. 6.13. The orange dotted line represents the tangent

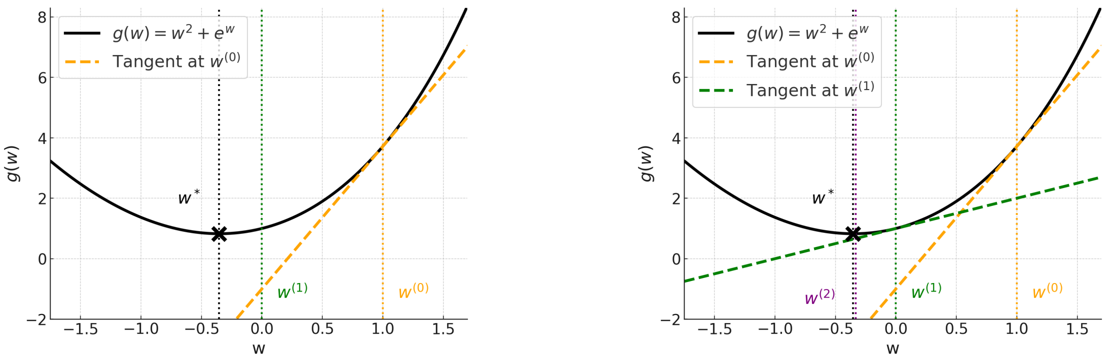


Figure 6.13: Left: Taking a step in the negative direction of the gradient $g'(w)$ at $w^{(0)} = 1$. Right: Taking a step in the negative direction of the gradient $g'(w)$ at $w^{(1)} = 0$.

line to $g(w)$ at $w^{(0)} = 1$. The slope of this line is the gradient $g'(w^{(0)}) = 2 + e$, which is positive. Thus, the negative direction of the gradient is to the left. After taking a step of size $\eta^{(0)} g'(w^{(0)}) = \frac{1}{2+e}(2 + e) = 1$ in the negative direction of the gradient we arrive at $w^{(1)} = 0$, which is indeed left of $w^{(0)} = 1$. The vertical green dotted line depicts $w^{(1)} = 0$.

We can repeat this process to get $w^{(2)}$. We can see this visually on the right in Fig. 6.13. The green dotted line represents the tangent line to $g(w)$ at $w^{(1)} = 0$. The slope of this line is the gradient $g'(w^{(1)}) = 1$, which is positive. Thus, the negative direction of the gradient is again to the left. After, taking a step of size $\eta^{(1)} g'(w^{(1)}) = \frac{1}{3}(1) = 1/3$ in the negative direction of the gradient we arrive at $w^{(2)} = -1/3$, which is indeed left of $w^{(1)} = 0$. The vertical purple dotted line depicts $w^{(2)} = -1/3$. □

The above example provides some intuition behind the name "gradient descent". The gradient gives us the direction that increases the function (the ascent direction). Thus, the negative gradient gives us the direction of descent. The step size $\eta^{(t)}$ determines how large of a step we take in the negative direction of the gradient. In the previous example we chose the step size to be $1/g''(w^{(t)})$, which depended on the second derivative of $g(w)$. At the beginning of this section we explained that the benefit of first-order gradient descent is that it avoids computing the second derivative of $g(w)$. As such, in the next two examples we will explore what happens if we do not choose the step size to be $1/g''(w^{(t)})$.

Example 6.14: [Small step size] We continue with the function $g(w) = w^2 + e^w$ and initial point $w^{(0)} = 1$ from Example 6.13. However, now we select a small step size $\eta^{(t)} = 0.1$ for all $t \in \mathbb{N}$. The first step gives us

$$w^{(1)} = 1 - 0.1(2 \cdot 1 + e^1) \approx 0.528.$$

The second step gives us

$$w^{(2)} = 0.528 - 0.1(2 \cdot 0.528 + e^{0.528}) \approx 0.253.$$

This is visualized in Fig. 6.14. We called this a "small" step size because it is smaller than

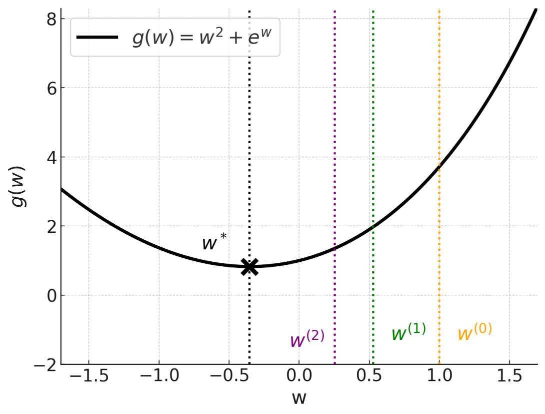


Figure 6.14: Gradient descent with small step size.

$1/g''(w^{(0)}) = 1/(2 + e)$ and $1/g''(w^{(1)}) = 1/3$. We can see that a small step size causes for the steps to be smaller and thus $w^{(2)}$ is further from w^* in this example than it was in Example 6.13. In general, a small step size will cause the method to take longer to converge to w^*. □

Example 6.15: [Large step size] We again continue with the function $g(w) = w^2 + e^w$ and initial point $w^{(0)} = 1$ from Example 6.13. However, now we select a large step size $\eta^{(t)} = 1$

for all $t \in \mathbb{N}$. The first step gives us

$$w^{(1)} = 1 - 1(2 \cdot 1 + e^1) \approx -3.72.$$

The second step gives us

$$w^{(2)} = -3.72 - 1(2 \cdot -3.72 + e^{-3.72}) \approx 3.696.$$

This is visualized in Fig. 6.15. We can see that a large step size causes for the steps to

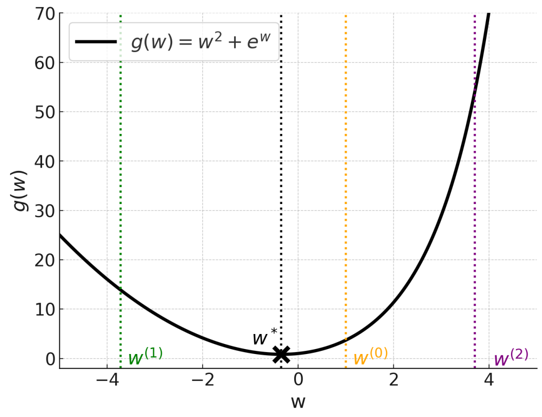


Figure 6.15: Gradient descent with large step size.

be larger, which means we can overshoot w^*. In particular, if we look at the plot we can see that $w^{(1)}$ is to the left of w^*, which means our step from $w^{(0)}$ to $w^{(1)}$ was too large. Similarly, $w^{(2)}$ is to the right of w^*, which means our step from $w^{(1)}$ to $w^{(2)}$ was also too large. In some cases, if the steps are too large the method may not even converge to w^*, and just keep oscillating around it. □

The above two examples highlight the importance of selecting a good step size. If the step size is too small the method may take longer to converge to w^*, while if the step size is too large the method may not converge at all. Selecting a small step is a safe choice, since it will usually converge to w^*, but it may take a long time. In Section 6.4.4 we will discuss some methods for selecting the step size. But first we discuss gradient descent in the multivariate case.

6.4.3 Multivariate Gradient Descent

Let $g : \mathbb{R}^d \to \mathbb{R}$ be a function of $d > 1$ variables. Our goal is to find

$$\mathbf{w}^* = (w_1^*, \ldots, w_d^*) = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}}\, g(\mathbf{w}).$$

If $g(\mathbf{w})$ is convex, then we can use gradient descent to find $\mathbf{w}^*$. The gradient descent update rule is

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)} \nabla g(\mathbf{w}^{(t)}), \quad \text{where} \quad \nabla g(\mathbf{w}) = \left(\frac{\partial g}{\partial w_1}(\mathbf{w}), \ldots, \frac{\partial g}{\partial w_d}(\mathbf{w}) \right)^\top \in \mathbb{R}^d$$

is the gradient of $g(\mathbf{w})$, and $\eta^{(t)}$ is the step size at iteration $t \in \mathbb{N}$. This update rule is the multivariate version of the univariate update rule we saw in Section 6.4.2. The intuition behind the multivariate update rule is the same as the univariate update rule. The gradient $\nabla g(\mathbf{w}^{(t)})$ gives us the direction of ascent of $g(\mathbf{w})$ at $\mathbf{w}^{(t)}$. Thus, the negative gradient $-\nabla g(\mathbf{w}^{(t)})$ gives us the direction of descent. The step size $\eta^{(t)}$ determines how large of a step we take in the negative direction of the gradient.

Remark: Notice that we have not mentioned second-order gradient descent in the multivariate case. The reason for this is that the update rule for second-order gradient descent is more complicated in the multivariate case. The second derivative of $g(\mathbf{w})$ would now have to be replaced with something called the Hessian matrix, which is a d by d matrix of second partial derivatives of $g(\mathbf{w})$. The Hessian matrix is more computationally expensive to compute than the gradient, and as such is not used as often in practice. For the remainder of these notes we will focus on first-order gradient descent.

Example 6.16: Let $g(\mathbf{w}) = g(w_1, w_2) = w_1^2 + w_2^2 + e^{w_1} + e^{w_2}$, where $\mathbf{w} \in \mathbb{R}^2$. Let our initial point be $\mathbf{w}^{(0)} = (1,1)^\top$, and learning rate $\eta^{(t)} = 1/(2+e)$ for all $t \in \mathbb{N}$. The gradient of $g(\mathbf{w})$ is

$$\nabla g(\mathbf{w}) = \left(\frac{\partial g}{\partial w_1}(\mathbf{w}), \frac{\partial g}{\partial w_2}(\mathbf{w})\right)^\top = (2w_1 + e^{w_1}, 2w_2 + e^{w_2})^\top .$$

The first step of gradient descent gives us

$$\mathbf{w}^{(1)} = \mathbf{w}^{(0)} - \eta^{(0)} \nabla g(\mathbf{w}^{(0)}) = (1,1)^\top - \frac{1}{2+e}(2+e, 2+e)^\top = (1,1)^\top - (1,1)^\top = (0,0)^\top .$$

The second step of gradient descent gives us

$$\mathbf{w}^{(2)} = \mathbf{w}^{(1)} - \eta^{(1)} \nabla g(\mathbf{w}^{(1)}) = (0,0)^\top - \frac{1}{2+e}(1,1)^\top = \left(-\frac{1}{2+e}, -\frac{1}{2+e}\right)^\top .$$

This is visualized in Fig. 6.16 by a 3D plot of $g(w_1, w_2)$. We can see that each step of

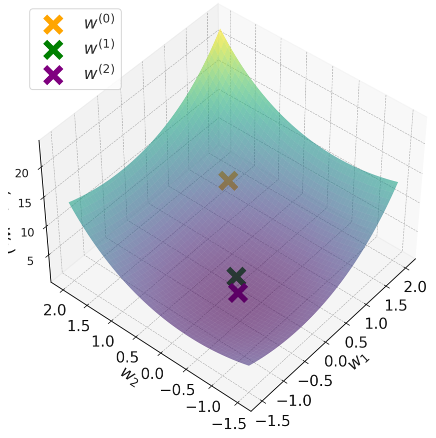


Figure 6.16: Gradient descent for $g(w_1, w_2) = w_1^2 + w_2^2 + e^{w_1} + e^{w_2}$.

gradient descent takes us closer to the minimizer of $g(w_1, w_2)$. To build further intuition, we can visualize the steps in the w_1 and w_2 plane. On the left in Fig. 6.17 we have a contour

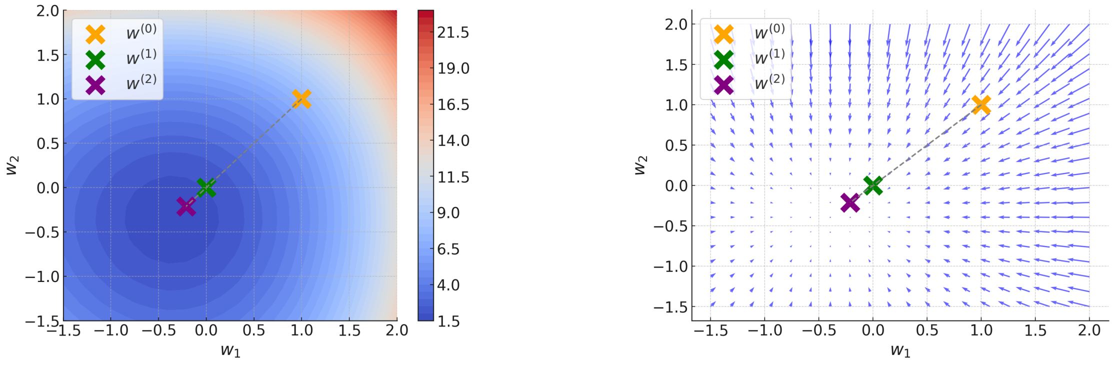


Figure 6.17: Left: Contour plot of $g(w_1, w_2) = w_1^2 + w_2^2 + e^{w_1} + e^{w_2}$. Right: Negative gradient at different points in the w_1 and w_2 plane.

plot of $g(w_1, w_2)$, where the colors of the contours represent the value of $g(w_1, w_2)$. The darkest blue area represents the smallest value of $g(w_1, w_2)$. It can be seen that the steps $w^{(1)}$, and $w^{(2)}$ are moving towards the dark blue region. We can also visualize the negative gradients, as shown on the right in Fig. 6.17. The arrows represent the negative gradient at different points in the w_1 and w_2 plane. We can see that the negative gradients point towards the dark blue region, which is the direction of descent. Also, our steps $w^{(1)}$ and $w^{(2)}$ were taken in the direction of the negative gradient, as expected. □

In general, we will take more than 2 steps of gradient descent to get a better approximation of the minimizer of $g(\mathbf{w})$. We show what taking 9 steps of gradient descent looks like for a more complicated function in the next example.

Example 6.17: Let $g(\mathbf{w}) = g(w_1, w_2) = (1 - w_1)^2 + 100(w_2 - w_1^2)^2$, where $\mathbf{w} \in \mathbb{R}^2$. Let our initial point be $\mathbf{w}^{(0)} = (1.7, -0.8)^\top$, and learning rate $\eta^{(t)} = 1/6$ for all $t \in \mathbb{N}$. We avoid explicitly calculating all 9 steps of gradient descent, and instead visualize the steps in Fig. 6.18. We can see that each step of gradient descent takes us closer to the minimizer

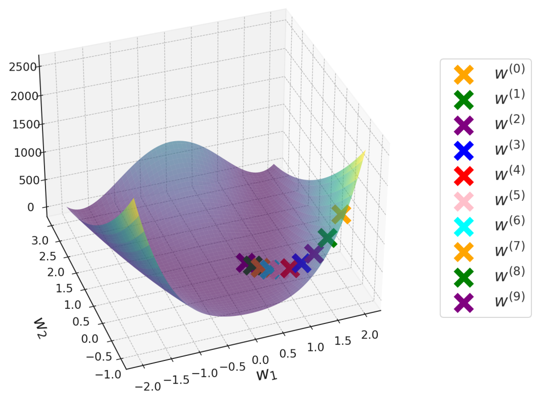


Figure 6.18: Gradient descent for $g(w_1, w_2) = (1 - w_1)^2 + 100(w_2 - w_1^2)^2$.

of $g(w_1, w_2)$. Similar to the previous example, we can visualize the steps in the w_1 and w_2 plane. On the left in Fig. 6.19 we have a contour plot of $g(w_1, w_2)$, where the colors of the

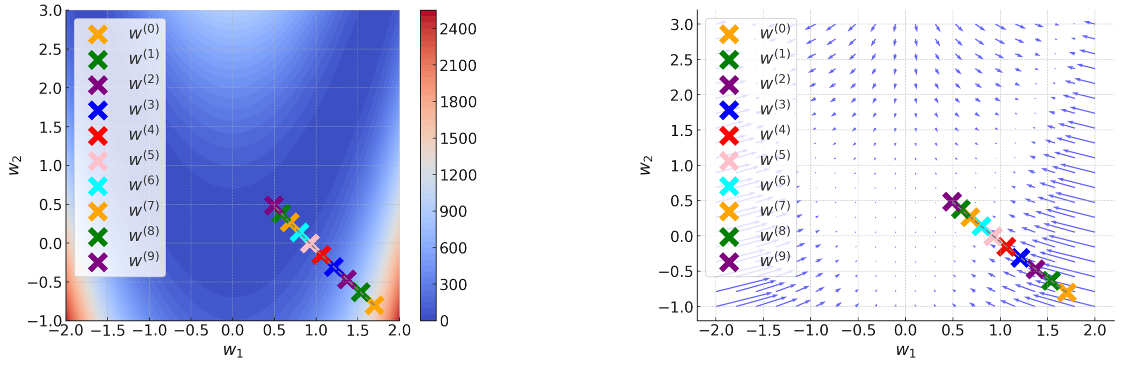


Figure 6.19: Left: Contour plot of $g(w_1, w_2) = (1 - w_1)^2 + 100(w_2 - w_1^2)^2$. Right: Negative gradient at different points in the w_1 and w_2 plane.

contours represent the value of $g(w_1, w_2)$. While on the right we have the negative gradient at different points in the w_1 and w_2 plane. □

In general, if the step size $\eta^{(t)}$ goes to zero as t goes to infinity, then the sequence of parameters $\mathbf{w}^{(t)}$ will approach the minimizer of $g(\mathbf{w})$ as t goes to infinity[2]. The high level intuition for why the step size has to go to zero is that as we get closer to the minimizer of $g(\mathbf{w})$, we need to take smaller steps to avoid overshooting the minimizer. If the step size is kept constant that means we could constantly oscillate around the minimizer, and never actually converge to it.

In the above examples we chose $d = 2$ to be able to visualize the steps of gradient descent. The same intuition applies for $d > 2$, but we cannot visualize the steps.

Exercise 6.1: You are minimizing a function $g : \mathbb{R}^d \to \mathbb{R}$ by using gradient descent with an initial point $\mathbf{w}^{(0)}$ and step size $\eta^{(t)} = \eta$ for all $t \in \mathbb{N}$. You run gradient descent for $T = 10^6$ iterations and get the parameter $\mathbf{w}^{(T)}$. Can you be certain that $\mathbf{w}^{(T)} = \mathbf{w}^* = \text{argmin}_{\mathbf{w} \in \mathbb{R}^d} g(\mathbf{w})$? □

6.4.4 Selecting the Step Size

In Section 6.4.2 we discussed the importance of selecting a good step size. However, we did not discuss how to select the step size. In this section we discuss some common choices for the step size. The following definitions are for all $t \in \mathbb{N}$.

1. **Constant:** One simple method is to select a constant step size

$$\eta^{(t)} = \eta \quad \text{where} \quad \eta \in (0, \infty).$$

2. **Inverse decaying:** Another method is to select a step size that decreases as t increases in an inverse manner

$$\eta^{(t)} = \frac{\eta}{1 + \lambda t} \quad \text{where} \quad \eta, \lambda \in (0, \infty).$$

[2]Actually, not only does the step size have to go to zero as t goes to infinity, but it also has to satisfy some other conditions saying it decreases quickly enough.

The rational behind this method is that as t increases the step size decreases, which allows for smaller steps as we get closer to w^*.

3. **Exponential decaying:** A third method is to select a step size that decreases exponentially as t increases

$$\eta^{(t)} = \eta e^{-\lambda t} \quad \text{where} \quad \eta, \lambda \in (0, 1).$$

This method is similar to the inverse decaying step size, but the step size decreases faster as t increases, due to the exponential term.

4. **Normalized gradient:** A slightly more sophisticated method than the previous ones is to normalize by the gradient

$$\eta^{(t)} = \frac{\eta}{\epsilon + \|\nabla g(\mathbf{w}^{(t)})\|} \quad \text{where} \quad \eta, \epsilon \in (0, \infty),$$

and

$$\|\nabla g(\mathbf{w}^{(t)})\| = \sqrt{\sum_{j=1}^{d} \left(\frac{\partial g}{\partial w_j}(\mathbf{w}^{(t)}) \right)^2}.$$

The value of ϵ is usually set to a small positive number to avoid division by zero (ex: $\epsilon = 10^{-8}$). The term $\|\nabla g(\mathbf{w}^{(t)})\|$ is called the 2-norm of the gradient, which is a measure of the magnitude of the gradient. Dividing by the magnitude of a vector is called normalizing by the vector.

To understand the rational behind this method recall that we take steps $-\eta^{(t)}\nabla g(\mathbf{w}^{(t)})$ in gradient descent. If the gradient $g(\mathbf{w}^{(t)})$ is large (in magnitude), then the step $-\eta^{(t)}\nabla g(\mathbf{w}^{(t)})$ will be large, and if the gradient is small, then the step will be small. To make sure we do no overshoot the minimum with each step, we choose the step size $\eta^{(t)}$ to be inversely proportional to the magnitude of the gradient. This way if the gradient is large, the step size will be small, and if the gradient is small, the step size $\eta^{(t)}$ will be large.

The above methods are just a few of the many methods for selecting the step size. In practice, the step size is often selected through trial and error, by trying different step sizes and seeing which one works best.

Exercise 6.2: Suppose you are using an exponential decaying step size. Is there some way to set the parameters η and λ so that you are using a constant step size? □

6.5 Gradient Descent for Linear Regression

In this section we will apply gradient descent to the linear regression problem we discussed in Section 6.3. Recall that in linear regression we have a dataset

$$\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)) \in (\mathcal{X} \times \mathcal{Y})^n, \quad \text{where} \quad \mathcal{X} = \mathbb{R}^{d+1}, \mathcal{Y} = \mathbb{R}.$$

The function class is the set of all linear functions

$$\mathcal{F} = \{f | f : \mathbb{R}^{d+1} \to \mathbb{R}, \text{ and } f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \text{ for some } \mathbf{w} \in \mathbb{R}^{d+1}\}.$$

The loss function is the squared loss $\ell(\hat{y}, y) = (\hat{y} - y)^2$, where $\hat{y}, y \in \mathbb{R}$. The objective is to find

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^{d+1}} \hat{L}(\mathbf{w}), \quad \text{where} \quad \hat{L}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2.$$

The function $\hat{L}(\mathbf{w})$ is convex, and in Section 6.3 we derived a closed form solution for $\hat{\mathbf{w}}$. Next, we will apply gradient descent to find an approximation of $\hat{\mathbf{w}}$.

6.5.1 Batch Gradient Descent

The gradient descent update rule for $t \in \mathcal{N}$ is

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)} \nabla \hat{L}(\mathbf{w}^{(t)}),$$

where $\eta^{(t)}$ is the step size at iteration $t \in \mathbb{N}$, and $\nabla \hat{L}(\mathbf{w}^{(t)})$ is the gradient of $\hat{L}(\mathbf{w})$ at $\mathbf{w}^{(t)}$. Recall from Section 6.3 that the partial derivative of $\hat{L}(\mathbf{w})$ with respect to w_j is

$$\frac{\partial \hat{L}(\mathbf{w})}{\partial w_j} = \frac{2}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i) x_{ij},$$

Thus, the gradient of $\hat{L}(\mathbf{w})$ is

$$\begin{aligned}
\nabla \hat{L}(\mathbf{w}) &= \left(\frac{2}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i) x_{i1}, \ldots, \frac{2}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i) x_{id} \right)^\top \\
&= \frac{2}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i) \left(x_{i1}, \ldots, x_{id} \right)^\top \\
&= \frac{2}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i) \mathbf{x}_i.
\end{aligned}$$

The gradient descent update rule is

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)} \frac{2}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w}^{(t)} - y_i) \mathbf{x}_i.$$

This is called the batch gradient descent (BGD) update rule. The reason for the name "batch" is that we are using all n data points in $\hat{L}(\mathbf{w})$ to estimate $L(\mathbf{w}) = \mathbb{E}\left[(\mathbf{x}^\top \mathbf{w} - y)^2\right]$. We will see in the next section that there are other versions of gradient descent that do not use all n data points to estimate $L(\mathbf{w})$.

Exercise 6.3: What would the BGD update rule be if we used the loss function $\ell(\hat{y}, y) = (\hat{y} - y)^4$? In the 1-dimensional case, with no bias term (i.e. $w \in \mathbb{R}$), is $\hat{L}(w)$ convex? $\square$

The learner $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \mathcal{F}$ that uses BGD is then defined as

$$\mathcal{A}(\mathcal{D}) = \hat{f} \quad \text{where} \quad \hat{f}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}^{(T)},$$

Algorithm 2: BGD Linear Regression Learner (with a constant step size)

```
1: input: $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$, step size $\eta$, number of epochs $T$
2: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^{d+1}$
3: for $t = 1, \ldots, T$ do
4:     $\nabla \hat{L}(\mathbf{w}) \leftarrow \frac{2}{n} \sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)\mathbf{x}_i$
5:     $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \hat{L}(\mathbf{w})$
6: return $\hat{f}(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$
```

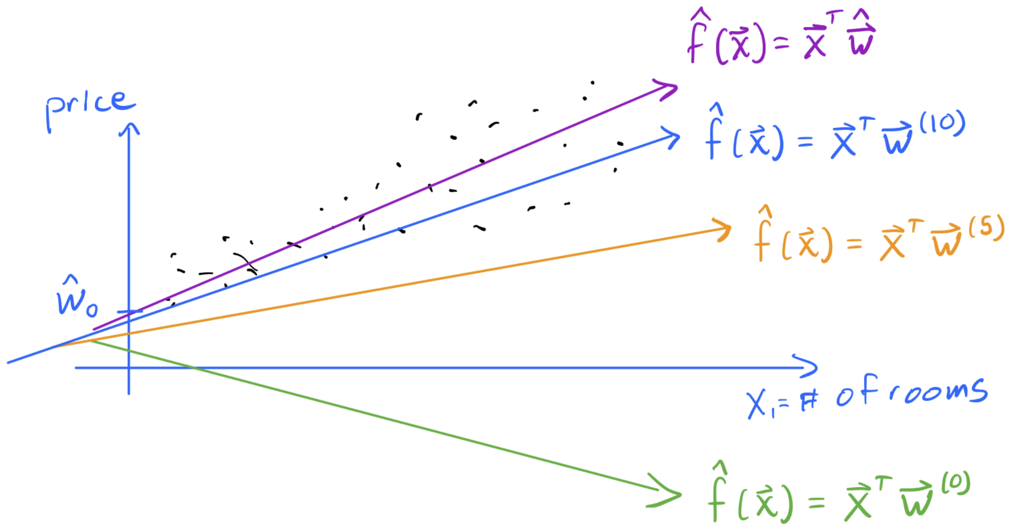
and $\mathbf{w}^{(0)}$ is randomly initialized. The number of iterations $T \in \mathbb{N}$ is often called the number of *epochs*.

In pseudocode, the learner $\mathcal{A}$ is defined as in algorithm 2. In the algorithm 2 we used a constant step size η; however, we could have used any of the step sizes we discussed in Section 6.4.4.

Exercise 6.4: Write the pseudocode for BGD with an exponential decaying step size. □

The larger T is, the closer $\mathbf{w}^{(T)}$ will be to $\hat{\mathbf{w}}$. Notice that unlike the closed form solution from Section 6.3, BGD does not give us an exact solution to the linear regression problem.

Example 6.18: We can visualize the progress of BGD by plotting the predictor $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}^{(t)}$ after $t = 0, 5, 10$ epochs (Fig. 6.20). As the number epochs increases, the BGD predictor $\mathbf{x}^\top \mathbf{w}^{(t)}$ gets closer to the closed form predictor $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \hat{\mathbf{w}}$. □



$\hat{f}(\vec{x}) = \vec{x}^T \hat{\vec{w}}$
price
$\hat{f}(\vec{x}) = \vec{x}^T \vec{w}^{(10)}$
$\hat{f}(\vec{x}) = \vec{x}^T \vec{w}^{(5)}$
$\hat{w}_0$
$x_1 = \#$ of rooms
$\hat{f}(\vec{x}) = \vec{x}^T \vec{w}^{(0)}$

Figure 6.20: The predictor $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}^{(t)}$ after $t = 0, 5, 10$ epochs of BGD compares to the closed form predictor $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \hat{\mathbf{w}}$.

We mentioned in Section 6.4 that gradient descent is useful when a closed form solution does not exist. In this case a closed form solution does exist; however, when we deal with classification problems in Chapter 9 we will see that a closed form solution does not exist, and gradient descent is a useful tool to find an approximate solution. Even though a closed form solution exists for linear regression, BGD can still have an advantage when considering the computational cost.

Computational Cost: Closed Form Solution vs. Batch Gradient Descent

First we should define what we mean by computational cost. We will use the imprecise definition that the computational cost of an algorithm is the total amount of elementary operations the algorithm performs. Elementary operations are operations that can be performed in a constant amount of time by a computer (ex: addition, multiplication, etc.). The "big O" notation will also be useful, which is a way to describe the growth rate of a function. If $f(n)$ is a function of n, then $f(n) = O(g(n))$ if there exists a constant $c > 0$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$. In other words, $f(n)$ grows at most as fast as $g(n)$ for large n. For instance if $f(n) = 3n^2 + 2n + 1$, then $f(n) = O(n^2)$. Multiplying by constants or adding terms like $2n$ that grow slower than n^2 do not change the growth rate of $f(n)$.

Let us now study the computational cost of the closed form solution for linear regression

$$\hat{\mathbf{w}} = A^{-1}\mathbf{b} \quad \text{where} \quad A = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top, \quad \mathbf{b} = \sum_{i=1}^{n} y_i \mathbf{x}_i.$$

Computing A requires $O(d^2 n)$ operations since computing $\mathbf{x}_i \mathbf{x}_i^\top$ requires $O((d+1)^2) = O(d^2)$ operations, and we have to do this for all n data points. Computing $\mathbf{b}$ requires $O(dn)$ operations since computing $y_i \mathbf{x}_i$ requires $O(d+1) = O(d)$ operations, and we have to do this for all n data points. Computing A^{-1} requires $O((d+1)^3) = O(d^3)$ operations, which is well known result for computing the inverse of a $d+1$ by $d+1$ matrix[3]. The matrix multiplication $A^{-1}\mathbf{b}$ requires $O((d+1)^2) = O(d^2)$ operations. The total computational cost of the closed form solution is $O(d^2 n + dn + d^3 + d^2) = O(d^2 n + d^3)$.

Next, let us study the computational cost of BGD for linear regression, defined in algorithm 2. Computing $(\mathbf{x}_i^\top \mathbf{w} - y_i)\mathbf{x}_i$ requires $O(d+1+d+1+d+1) = O(d)$ operations, since we have to compute $\mathbf{x}_i^\top \mathbf{w}$ subtract y_i, and multiply by $\mathbf{x}_i$, each taking $O(d+1)$ operations. Computing $\nabla \hat{L}(\mathbf{w})$ requires $O(dn)$ operations, since we have to compute $(\mathbf{x}_i^\top \mathbf{w} - y_i)\mathbf{x}_i$ for all n data points. The update rule $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \hat{L}(\mathbf{w})$ requires $O(d+1+d+1) = O(d)$ operations, since we have to multiply $\nabla \hat{L}(\mathbf{w})$ by η and subtract from $\mathbf{w}$. Since this update rule is repeated T times, the total computational cost of BGD is $O((dn+d)T) = O(dnT)$.

Thus, the computational cost of BGD is $O(dnT)$, while the computational cost of the closed form solution is $O(d^2 n + d^3)$. If $T < d$, then $O(dnT) < O(d^2 n) \leq O(d^2 n + d^3)$. Which implies that when the number of epochs T is less than the number of features d, BGD is computationally cheaper than the closed form solution. In some machine learning problems the number of features d can be very large, and in these cases BGD can be computationally cheaper than the closed form solution. For example, if you are trying to predict a persons level of risk for a disease based on their genetic information, the number of features d can be in the tens of thousands (representing the expression level of each gene), while running BGD for a few hundred epochs can often be enough to reach a good solution.

[3]There are methods for computing an inverse that are slightly more efficient than $O(d^3)$.

This still may not seem like a big advantage. Next, we introduce another version of gradient descent called mini-batch gradient descent, which can be computationally cheaper than BGD.

6.5.2 Mini-Batch Gradient Descent

In mini-batch gradient descent (MBGD) we use subsets of the dataset to estimate $L(\mathbf{w}) = \mathbb{E}\left[(\mathbf{x}^\top \mathbf{w} - y)^2\right]$, and then take gradient steps based on these estimates. We describe this process in more detail below.

A subset of the dataset is called a *mini-batch*, and the size of the mini-batch is often denoted by $b \in \mathbb{N}$. If the number of data points n is divisible by b, then we can think of the dataset as split into $M = n/b$ mini-batches. Visually, we can see this as follows

$$\mathcal{D} = \left(\underbrace{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_b, y_b)}_{\text{mini-batch 1}}, \underbrace{(\mathbf{x}_{b+1}, y_{b+1}), \ldots, (\mathbf{x}_{2b}, y_{2b})}_{\text{mini-batch 2}}, \ldots, \underbrace{(\mathbf{x}_{(M-1)b+1}, y_{(M-1)b+1}), \ldots, (\mathbf{x}_n, y_n)}_{\text{mini-batch } M} \right).$$

Example 6.19: Let $n = 8, b = 2$ then $M = 8/2 = 4$, and the dataset can be visualized as

$$\mathcal{D} = \left(\underbrace{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2)}_{\text{mini-batch 1}}, \underbrace{(\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4)}_{\text{mini-batch 2}}, \underbrace{(\mathbf{x}_5, y_5), (\mathbf{x}_6, y_6)}_{\text{mini-batch 3}}, \underbrace{(\mathbf{x}_7, y_7), (\mathbf{x}_8, y_8)}_{\text{mini-batch 4}} \right).$$

The number of mini-batches is $M = 4$. □

In general, n is not divisible by b, so we define $M = \text{floor}(\frac{n}{b}) \in \mathbb{N}$, which is just $\frac{n}{b}$ rounded down to the nearest integer. To avoid notational clutter, if n is not divisible by b, we will simply not use the last $n - Mb$ data points from the dataset in the following steps and definitions[4].

For each mini-batch $m \in \{1, \ldots, M\}$ we have a sample mean estimate of the expected loss

$$\hat{L}_m(\mathbf{w}) = \frac{1}{b} \sum_{i=(m-1)b+1}^{mb} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2.$$

Exercise 6.5: Suppose that when n/b was not an integer we did not discard the last $n - Mb$ data points from the dataset, and instead used them in the last mini-batch $M + 1$. What would the sample mean estimate $\hat{L}_{M+1}(\mathbf{w})$ be? □

During an epoch $t \in \mathbb{N}$, the MBGD learner updates the weight vector $\mathbf{w}$ for each mini-batch $m \in \{1, \ldots, M\}$ using the update rule

$$\mathbf{w}^{(t,m+1)} = \mathbf{w}^{(t,m)} - \eta^{(t)} \nabla \hat{L}_m(\mathbf{w}^{(t,m)}).$$

[4]In practice the last $n - Mb$ data points are not discarded, but rather used in the last mini-batch. However, not doing so does not affect the main idea of MBGD and makes the presentation much cleaner for these course notes.

Once the final mini-batch M is reached, the learner updates the weight vector $\mathbf{w}$ for the next epoch $t+1$ using the update rule

$$\mathbf{w}^{(t+1,1)} = \mathbf{w}^{(t,M+1)} - \eta^{(t)} \nabla \hat{L}_1(\mathbf{w}^{(t,M+1)}).$$

This is repeated for T epochs. An important note is that before each epoch t, the dataset $\mathcal{D}$ should be shuffled, so that the order of the data points is random for each epoch. This means that the estimate of the expected loss $\hat{L}_m(\mathbf{w})$ is different for each epoch, since the data points in each mini-batch are different. We will not formally show why this is helpful, but roughly speaking it improves the average $\hat{L}_m(\mathbf{w})$ over all epochs.

The MBGD learner $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^n \to \mathcal{F}$ is defined as

$$\mathcal{A}(\mathcal{D}) = \hat{f} \quad \text{where} \quad \hat{f}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}^{(T,M+1)},$$

and $\mathbf{w}^{(0,1)}$ is randomly initialized. The pseudocode for learner $\mathcal{A}$ is given in algorithm 3

Algorithm 3: MBGD Linear Regression Learner (with a constant step size)

```
1: input:
   $\mathcal{D} = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n))$, step size $\eta$, number of epochs $T$, mini-batch size $b$
2: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^{d+1}$
3: $M \leftarrow \text{floor}(\frac{n}{b})$
4: for $t = 1, \dots, T$ do
5:    randomly shuffle $\mathcal{D}$
6:    for $m = 1, \dots, M$ do
7:       $\nabla \hat{L}_m(\mathbf{w}) \leftarrow \frac{2}{b} \sum_{i=(m-1)b+1}^{mb} (\mathbf{x}_i^\top \mathbf{w} - y_i)\mathbf{x}_i$
8:       $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \hat{L}_m(\mathbf{w})$
9: return  $\hat{f}(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$
```

Notice that there are $T \cdot M$ updates to $\mathbf{w}$ in MBGD. If $b = n$, then MBGD is equivalent to BGD. If $b = 1$, then MBGD is called *stochastic gradient descent* (SGD).

Now, we will compare MBGD to BGD, for the case where $b < n$. In MBGD, notice that the sample mean estimate $\hat{L}(\mathbf{w})$ uses b data points. This is in contrast to the sample mean estimate $\hat{L}(\mathbf{w})$ in BGD, which used all n data points. Let us treat our dataset as a random variable D momentarily. In Chapter 5 we learned that the expected value of $\hat{L}(\mathbf{w})$ is $L(\mathbf{w})$, and for the same reasons, the expected value of $\hat{L}_m(\mathbf{w})$ is also $L(\mathbf{w})$. This is a desirable property, since we want our sample mean estimate to be close to the expected value of the loss. In Chapter 5, we also learned that the variance of $\hat{L}(\mathbf{w})$ scales as $1/n$, where n is the number of data points. Similarly, the variance of $\hat{L}_m(\mathbf{w})$ scales as $1/b$, where b is the size of the mini-batch. Thus, the variance of $\hat{L}(\mathbf{w})$ is smaller than the variance of $\hat{L}_m(\mathbf{w})$, since $n \geq b$. This implies that if we randomly sample a dataset D from the distribution of datasets $\mathbb{P}_D$, then $\hat{L}(\mathbf{w})$ is more likely to be close to $L(\mathbf{w})$ than $\hat{L}_m(\mathbf{w})$. For this reason we say that $\hat{L}(\mathbf{w})$ is a better estimate of $L(\mathbf{w})$ than $\hat{L}_m(\mathbf{w})$.

At this point it may seem that MBGD is strictly worse than BGD, since we are using a worse estimate of $L(\mathbf{w})$. However, to see the advantage of MBGD, we need to consider the computational cost.

Computational Cost: Batch Gradient Descent vs. Mini-Batch Gradient Descent

In Section 6.5.1 we showed that the computational cost of BGD is $O(dnT)$. Next, we will calculate the computational cost of MBGD. The steps are almost identical to BGD, except that we have and extra loop over all the mini-batches. First, the computational cost of computing $(\mathbf{x}_i^\top \mathbf{w} - y_i)\mathbf{x}_i$ is $O(d)$. Computing $\nabla \hat{L}_m(\mathbf{w})$ requires $O(db)$ operations, since we have to compute $(\mathbf{x}_i^\top \mathbf{w} - y_i)\mathbf{x}_i$ for all b data points in the mini-batch. The update rule $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \hat{L}_m(\mathbf{w})$ requires $O(d)$ operations. After the inner loop over all mini-batches, the computational cost is $O((db + d)M) = O(dbM)$. Shuffling the dataset requires $O(n) = O(bM)$ operations. Thus, the total computational cost of MBGD is $O((dbM + bM)T) = O(dbMT)$, since the outer loop over all epochs is repeated T times.

The computational cost of MBGD is $O(dbMT)$, while the computational cost of BGD is $O(dnT)$. But, $O(bM) = O(n)$, which implies that $O(dbMT) = O(dnT)$. This means that for the same number of epochs T the computational cost of MBGD is the same as BGD. However, the number of gradient steps that MBGD takes is $T \cdot M$, while the number of gradient steps that BGD takes is T. As we discussed earlier, the gradient steps in MBGD are based on a worse estimate of $L(\mathbf{w})$ than in BGD. In practice it turns out that the increase in the number of gradient steps in MBGD is usually worth the decrease in the quality of the estimate of $L(\mathbf{w})$. Often, choosing b to be a relatively small number (ex: $b = 32$) will lead to a better solution than BGD, for the same number of epochs T.

6.6 Polynomial Regression

Polynomial regression is just the regression setting, but we select the function class $\mathcal{F}$ to be the set of all polynomial functions up to some degree, instead of just linear functions. To motivate the need for selecting a function class that is more expressive than linear functions, consider the following example.

Example 6.20: Suppose you are trying to predict the price of a car based on its age. The dataset $\mathcal{D}$ could look like that shown on the left in Fig. 6.21. Notice how the price of a

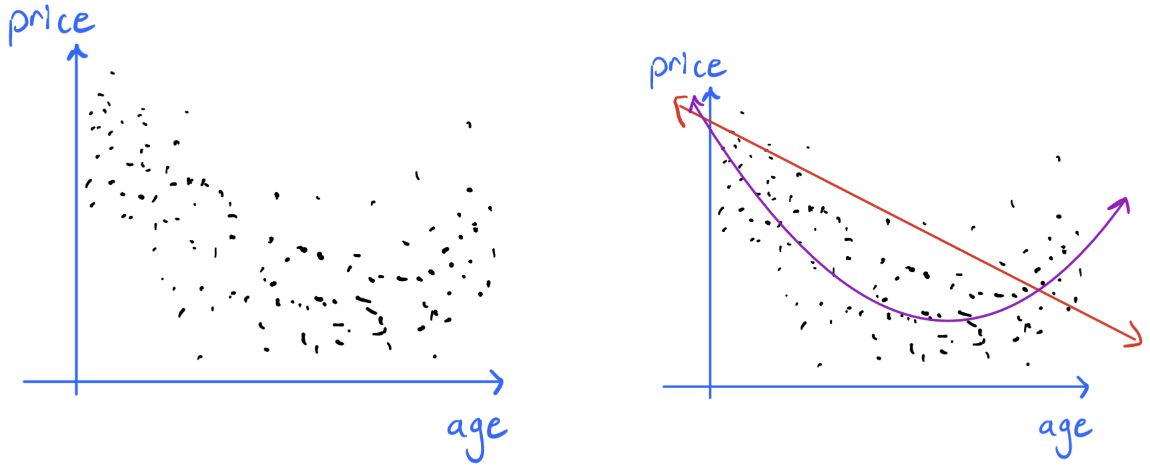


Figure 6.21: Left: Dataset for polynomial regression. Right: Linear and polynomial fit to the dataset.

car is not linearly related to its age. In particular, the price of a car initially decreases with age, but then after a certain age the car is considered an antique and the price increases. A linear function will not be able to capture this relationship. Instead we might want to use a polynomial function to capture this relationship. On the right in Fig. 6.21 we show a linear (red) and polynomial (purple) fit to the dataset. Clearly, using a polynomial function is a better choice in this case.

In this example $d = 1, \mathcal{X} = \mathbb{R}^{1+1} = \mathbb{R}^2$, and $\mathcal{Y} = \mathbb{R}$ since we are in the regression setting. Recall that a linear function is of the form

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} = w_0 + x_1 w_1 \quad \text{where} \quad \mathbf{w} \in \mathbb{R}^2.$$

The function class containing all linear functions is

$$\mathcal{F}_1 = \{f | f : \mathbb{R}^2 \to \mathbb{R}, \text{ and } f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \text{ for some } \mathbf{w} \in \mathbb{R}^2\}.$$

We have used the subscript 1 to denote that this is the function class containing all linear functions, which are polynomial functions of degree 1 or less. The red line in Fig. 6.21 is a linear function, which is an element of $\mathcal{F}_1$.

A polynomial function of degree 2 or less is of the form

$$f(\mathbf{x}) = w_0 + x_1 w_1 + x_1^2 w_2 = \phi_2(\mathbf{x})^\top \mathbf{w},$$

where $\mathbf{w} = (w_0, w_1, w_2)^\top \in \mathbb{R}^3$ and $\phi_2(\mathbf{x}) = (1, x_1, x_1^2)^\top \in \mathbb{R}^3$ is called a *feature map*. Notice that a degree 2 polynomial function is just a linear function of the transformed input $\phi_2(\mathbf{x})$. We can write the function class containing all polynomials of degree 2 or less as

$$\mathcal{F}_2 = \{f | f : \mathbb{R}^2 \to \mathbb{R}, \text{ and } f(\mathbf{x}) = \phi_2(\mathbf{x})^\top \mathbf{w}, \text{ for some } \mathbf{w} \in \mathbb{R}^3\}.$$

The subscript 2 denotes that this is the function class containing all polynomial functions of degree 2 or less. The purple quadratic curve in Fig. 6.21 is a polynomial function of degree 2, which is an element of $\mathcal{F}_2$. Also notice that $\mathcal{F}_1 \subset \mathcal{F}_2$, since a linear function would just have $w_2 = 0$.

If we wanted, we could use an even higher degree polynomial function to fit the dataset. For instance, a polynomial function of degree 3 or less is of the form

$$f(\mathbf{x}) = w_0 + x_1 w_1 + x_1^2 w_2 + x_1^3 w_3 = \phi_3(\mathbf{x})^\top \mathbf{w},$$

where $\mathbf{w} = (w_0, w_1, w_2, w_3)^\top \in \mathbb{R}^4$ and $\phi_3(\mathbf{x}) = (1, x_1, x_1^2, x_1^3)^\top \in \mathbb{R}^4$. The function class containing all polynomials of degree 3 or less is

$$\mathcal{F}_3 = \{f | f : \mathbb{R}^2 \to \mathbb{R}, \text{ and } f(\mathbf{x}) = \phi_3(\mathbf{x})^\top \mathbf{w}, \text{ for some } \mathbf{w} \in \mathbb{R}^4\}.$$

Notice again that $\mathcal{F}_2 \subset \mathcal{F}_3$. We can continue this process for any degree p. □

The above example shows how using a polynomial function can be useful in regression problems. However, the example only considered the case when $d = 1$. The definition of the feature map $\phi_p(\mathbf{x})$ for a polynomial function of degree p is a bit more complicated when $d > 1$. In the next example we show how to define the feature map $\phi_p(\mathbf{x})$ when $d = 2$, and then after the example we give the general definition for any degree p.

Example 6.21: Let $d = 2$, then $\mathcal{X} = \mathbb{R}^{2+1} = \mathbb{R}^3$, and $\mathcal{Y} = \mathbb{R}$. A linear function is of the form

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} = w_0 + x_1 w_1 + x_2 w_2 \quad \text{where} \quad \mathbf{w} \in \mathbb{R}^3.$$

The function class containing all linear functions is

$$\mathcal{F}_1 = \{f | f : \mathbb{R}^3 \to \mathbb{R}, \text{ and } f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \text{ for some } \mathbf{w} \in \mathbb{R}^3\}.$$

A polynomial function of degree 2 or less is of the form

$$f(\mathbf{x}) = w_0 + x_1 w_1 + x_2 w_2 + x_1^2 w_3 + x_1 x_2 w_4 + x_2^2 w_5 = \phi_2(\mathbf{x})^\top \mathbf{w},$$

where $\mathbf{w} = (w_0, w_1, w_2, w_3, w_4, w_5)^\top \in \mathbb{R}^6$ and $\phi_2(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)^\top \in \mathbb{R}^6$. Notice how the feature map $\phi_2(\mathbf{x})$ considers all possible products of the input features x_1, x_2 up to degree 2. The function class containing all polynomials of degree 2 or less is

$$\mathcal{F}_2 = \{f | f : \mathbb{R}^3 \to \mathbb{R}, \text{ and } f(\mathbf{x}) = \phi_2(\mathbf{x})^\top \mathbf{w}, \text{ for some } \mathbf{w} \in \mathbb{R}^6\}.$$

A polynomial function of degree 3 or less is of the form

$$\begin{aligned} f(\mathbf{x}) &= w_0 + x_1 w_1 + x_2 w_2 + x_1^2 w_3 + x_1 x_2 w_4 + x_2^2 w_5 + x_1^3 w_6 + x_1^2 x_2 w_7 + x_1 x_2^2 w_8 + x_2^3 w_9 \\ &= \phi_3(\mathbf{x})^\top \mathbf{w}, \end{aligned}$$

where

$$\begin{aligned} \mathbf{w} &= (w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9)^\top \in \mathbb{R}^{10}, \\ \phi_3(\mathbf{x}) &= (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2, x_1^3, x_1^2 x_2, x_1 x_2^2, x_2^3)^\top \in \mathbb{R}^{10}. \end{aligned}$$

The function class containing all polynomials of degree 3 or less is

$$\mathcal{F}_3 = \{f | f : \mathbb{R}^3 \to \mathbb{R}, \text{ and } f(\mathbf{x}) = \phi_3(\mathbf{x})^\top \mathbf{w}, \text{ for some } \mathbf{w} \in \mathbb{R}^{10}\}.$$

□

In general, the feature map $\phi_p(\mathbf{x})$ for a polynomial function of degree p considers all possible products of the input features $x_0, x_1, \ldots, x_d$ up to degree p. This means $\phi_p : \mathbb{R}^{d+1} \to \mathbb{R}^{\bar{p}}$ where $\bar{p} = \binom{d+p}{p}$. The equation for $\bar{p}$ can be derived by considering the number of ways to choose p elements from a set of $d+1$ elements with replacement, which is $\binom{(d+1)+p-1}{p} = \binom{d+p}{p}$. We can double check that this equation is correct by considering the examples we have seen so far. For $d = 1, p = 2$, we have $\bar{p} = \binom{1+2}{2} = 3$, and if $p = 3$ then $\bar{p} = \binom{1+3}{3} = 4$, which agrees Example 6.20. If $d = 2, p = 2$, we have $\bar{p} = \binom{2+2}{2} = 6$, which agrees with Example 6.21.

The function class containing all polynomial functions of degree p or less is

$$\mathcal{F}_p = \{f | f : \mathbb{R}^{d+1} \to \mathbb{R}, \text{ and } f(\mathbf{x}) = \phi_p(\mathbf{x})^\top \mathbf{w}, \text{ for some } \mathbf{w} \in \mathbb{R}^{\bar{p}}\}.$$

As we increase the degree p, the function class $\mathcal{F}_p$ becomes more expressive, since it can capture more complex relationships between the input features. In particular, $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \subset \ldots$. Thus, as p increases the function class $\mathcal{F}_p$ contains a predictor $\hat{f}_p$ that can fit the

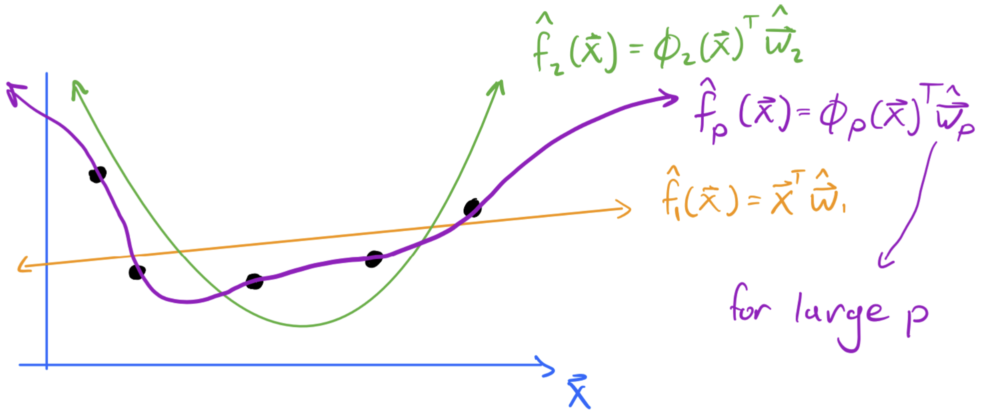

$\hat{f}_2(\vec{x}) = \phi_2(\vec{x})^T \hat{\vec{w}}_2$
$\hat{f}_p(\vec{x}) = \phi_p(\vec{x})^T \hat{\vec{w}}_p$
$\hat{f}_1(\vec{x}) = \vec{x}^T \hat{\vec{w}}_1$
for large p
$\vec{x}$

Figure 6.22: Predictors of various polynomial degrees fit to a dataset.

dataset at least as well as any predictor $\hat{f}_{p-1}$ in $\mathcal{F}_{p-1}$, since $\hat{f}_p$ could always be chosen to be $\hat{f}_{p-1}$. This can be seen in Fig. 6.22, where we fit a dataset with predictors of various polynomial degrees.

Exercise 6.6: If $f_3 \in \mathcal{F}_3$, can we be certain that $f_3 \in \mathcal{F}_2$ or that $f_3 \in \mathcal{F}_4$? $\square$

Exercise 6.7: Let $d = 3$, and $p = 2$. What is $\bar{p}$? Write the feature map $\phi_p(\mathbf{x})$. $\square$

The linear function class is actually a special case of the polynomial function class, where $p = 1$. Then $\phi_1(\mathbf{x}) = \mathbf{x}$ and $\bar{p} = \binom{d+1}{1} = d + 1$, giving back the linear function class $\mathcal{F}$, which we have seen in the previous sections. Also, $\mathcal{F}_p$ looks similar to the linear function class, except the feature vector $\mathbf{x}$ is now replaced by the feature map $\phi_p(\mathbf{x})$, and the weight vector $\mathbf{w}$ is now in $\mathbb{R}^{\bar{p}}$ instead of $\mathbb{R}^{d+1}$. For this reason, $\mathcal{F}_p$ can be thought of as the function class containing all linear functions of the transformed input $\phi_p(\mathbf{x})$. This turns out to be very useful, since the closed form solution and gradient descent algorithms we have seen so far for linear regression, can be applied to polynomial regression by simply replacing $\mathbf{x}$ with $\phi_p(\mathbf{x})$ and using $\mathbf{w} \in \mathbb{R}^{\bar{p}}$ everywhere.

An important consequence of the property that $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \subset \dots$ is that

$$\min_{f \in \mathcal{F}_1} \hat{L}(f) \geq \min_{f \in \mathcal{F}_2} \hat{L}(f) \geq \min_{f \in \mathcal{F}_3} \hat{L}(f) \geq \dots.$$

To see why this holds suppose that $\hat{f}_{p-1} = \min_{f \in \mathcal{F}_{p-1}} \hat{L}(f)$. Then we know that $\hat{f}_{p-1} \in \mathcal{F}_p$, since $\mathcal{F}_{p-1} \subset \mathcal{F}_p$. Let $\hat{f}_p = \min_{f \in \mathcal{F}_p} \hat{L}(f)$. Notice how $\hat{f}_p$ could be equal to $\hat{f}_{p-1}$, but it could also be a different function. If it is a different function, then it must make $\hat{L}(f)$ smaller, since otherwise it wouldn't be the minimizer of $\hat{L}(f)$ over $\mathcal{F}_p$. Thus, $\hat{L}(\hat{f}_p) \leq \hat{L}(\hat{f}_{p-1})$ which is the same as $\min_{f \in \mathcal{F}_p} \hat{L}(f) \leq \min_{f \in \mathcal{F}_{p-1}} \hat{L}(f)$ by definition of $\hat{f}_p$ and $\hat{f}_{p-1}$. If we plot the minimum loss $\min_{f \in \mathcal{F}_p} \hat{L}(f)$ as a function of p, we would see a decreasing curve (shown in Fig. 6.23). At this point you might be wondering why we don't just use the function class $\mathcal{F}_p$ with the highest degree p. Of course, the computational cost will increase as p increases, since the dimension of the weight vector $\mathbf{w}$ is increasing; however, there turns out to be an even more important reason. We will find that as p increases, the estimated loss of the predictor our learner outputs $\hat{L}(\hat{f}_p)$ will become a worse estimate of the true loss $L(\hat{f}_p)$.

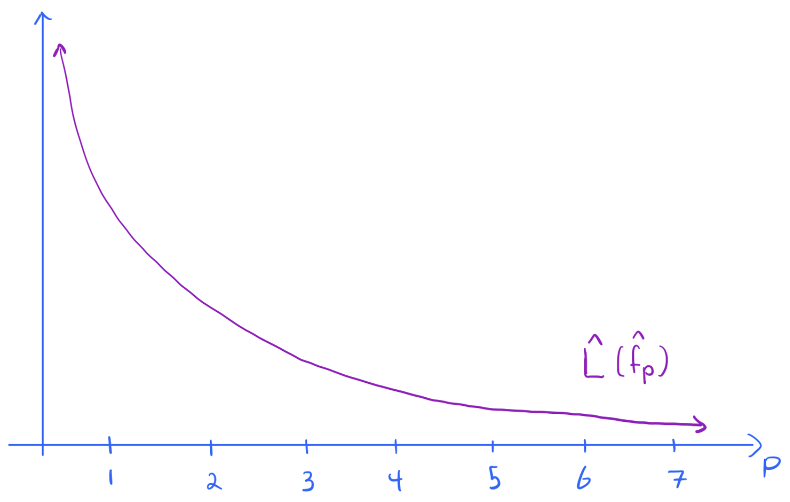


Figure 6.23: $\min_{f\in\mathcal{F}_p}\hat{L}(f)$ *as a function of the degree* p *of the polynomial function.*

The next chapter is dedicated to understanding this phenomenon, and how to evaluate if the predictor our learner outputs is good at predicting the label based on new data points.

Exercise 6.8: Suppose that

$$\bar{\mathcal{F}}_p = \{f | f : \mathbb{R}^{d+1} \to \mathbb{R},\ \text{and}\ f(\mathbf{x}) = \exp(\phi_p(\mathbf{x})^\top \mathbf{w}),\ \text{for some}\ \mathbf{w} \in \mathbb{R}^{\bar{p}}\}.$$

Is it true that $\bar{\mathcal{F}}_1 \subset \bar{\mathcal{F}}_2$? Is it true that $\mathcal{F}_1 \subset \bar{\mathcal{F}}_1$? Is it true that $\bar{\mathcal{F}}_1 \subset \mathcal{F}_1$? Is it true that $\min_{f\in\bar{\mathcal{F}}_1}\hat{L}(f) \leq \min_{f\in\bar{\mathcal{F}}_2}\hat{L}(f)$? □

Exercise 6.9: Suppose you want to find $\hat{f}_1 = \min_{f\in\mathcal{F}_1}\hat{L}(f)$, and $\hat{f}_2 = \min_{f\in\mathcal{F}_2}\hat{L}(f)$. To do this you decide to use batch gradient descent. Let $\hat{L}_p(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(\phi_p(\mathbf{x}_i)^\top\mathbf{w} - y_i)^2$. You run BGD on the function $\hat{L}_1(\mathbf{w})$ for $T = 100$ epochs, and get $\mathbf{w}_1^{(100)}$. You run BGD on the function $\hat{L}_2(\mathbf{w})$ for $T = 1000$ epochs, and get $\mathbf{w}_2^{(1000)}$. You are told that for both cases the number of iterations T that BGD is run for is likely enough for it to reach a good approximation of the minimizer.

Can you be certain that $\hat{L}_1(w_1^{(100)}) \geq \hat{L}_2(w_2^{(1000)})$? Do you think it is likely that $\hat{L}_1(w_1^{(100)}) \geq \hat{L}_2(w_2^{(1000)})$? □

To conclude we provide the pseudocode for the degree 2 polynomial feature map $\phi_2(\mathbf{x})$ in algorithm 4. The idea behind the implementation is to construct all possible pairs of the input features $x_0, x_1, \ldots, x_d$ without including any duplicates. This can be done by having an outer loop over all features, and an inner loop that starts at the index of the outer loop and goes to the end of the features. This way we avoid duplicates, since we only consider pairs (x_k, x_l) where $k \leq l$. If we wanted a degree 3 polynomial feature map, we would need to construct all possible triples of the input features. This would require a triple nested loop. In general if you wanted a degree p polynomial feature map, you would need to construct all possible p-tuples of the input features, which would require a p nested loop. There are more efficient ways to construct the polynomial feature map, but the nested loop method is the most intuitive and easiest to understand for the purposes of these course notes.

Algorithm 4: Degree 2 Polynomial Feature Map

```
1: **input:** feature vector $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_d)^\top \in \mathbb{R}^{d+1}$
2: $\bar{p} \leftarrow (d+1)(d+2)/2$
3: $\varphi \leftarrow (\varphi_0 = 0, \varphi_1 = 0, \ldots, \varphi_{\bar{p}-1} = 0)^\top \in \mathbb{R}^{\bar{p}}$
4: $j \leftarrow 0$
5: **for** $k = 0, \ldots, d$ **do**
6:    **for** $l = k, \ldots, d$ **do**
7:       $\varphi_j = x_k \cdot x_l$
8:       $j = j + 1$
9: **return** $\varphi$
```

Exercise 6.10: Write the pseudocode for the degree 3 polynomial feature map $\phi_3(\mathbf{x})$. $\square$

Chapter 7

Evaluating Predictors

In the last chapter we concluded our discussion with a question: what is the expected loss of the predictor output by an ERM learner? In particular, if $\mathcal{A}(\mathcal{D}) = \hat{f}$ is the predictor output by an ERM learner (Definition 1), then what is $L(\hat{f})$? Answering this question gives an *evaluation* of the predictor output by the ERM learner.

When we studied how to optimize the estimated loss $\hat{L}$ in Chapter 6, we assumed the dataset $\mathcal{D}$ was fixed. However, in Chapter 4 we discussed that in general the dataset is a random variable D. Our objective was then to define a learner $\mathcal{A}$ that that has a small expected loss in expectation over datasets D. More precisely, the learner should make $\mathbb{E}[L(\mathcal{A}(D))]$ small. Evaluating how well the ERM learner does at achieving this objective is the goal of this chapter.

7.1 Estimated vs. Expected Loss of a Predictor

Suppose that the learner has found the predictor $\hat{f}_D$ that minimizes the estimated loss over some function class $\mathcal{F}$. Formally, let

$$\mathcal{A}(D) = \hat{f}_D \quad \text{where} \quad \hat{f}_D = \operatorname*{argmin}_{f \in \mathcal{F}} \hat{L}(f).$$

We have added the subscript D to $\hat{f}$ to emphasize that the predictor output by the learner depends on the random dataset D, thus, is itself a random variable (which will be important to remember in the following discussion).

Since $\hat{f}_D$ is the minimizer of the estimated loss $\hat{L}$, but a good predictor would be one that minimizes the expected loss L, it is useful to compare these two quantities. In particular, we will would like to know how large the difference between the estimated and expected loss is in expectation over datasets D.

$$\mathbb{E}[L(\hat{f}_D)] - \mathbb{E}[\hat{L}(\hat{f}_D)]. \tag{7.1}$$

The expectation over datasets is needed since the predictor $\hat{f}_D$ is a random variable. It can be shown that the above quantity satisfies the following two properties:

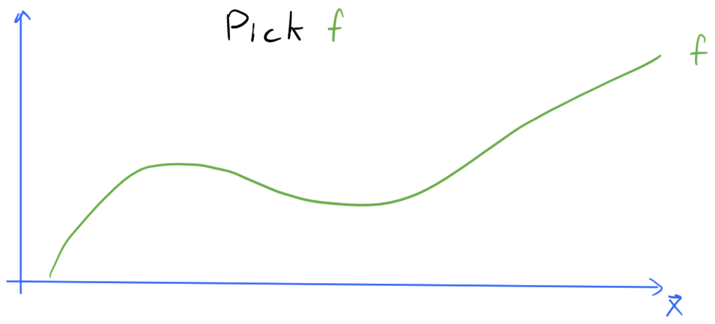
1. is always non-negative,
2. increases if the the function class $\mathcal{F}$ gets larger,
3. and decreases if the number of samples n in the dataset D increases.

We will not formally prove the properties here, since it is requires some technical machinery we have not discussed; however, we will try to give some intuition for why these statements are true.

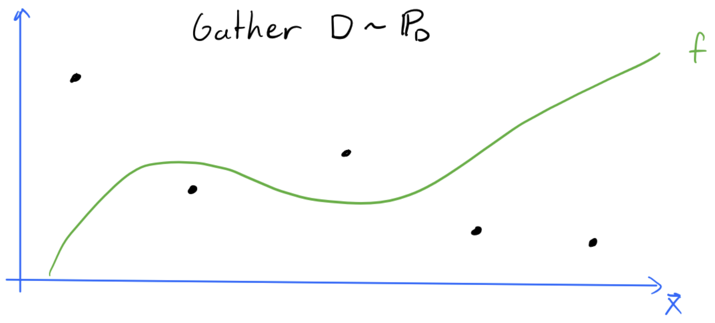
Recall that for a fixed predictor $f \in \mathcal{F}$, the expected value of the sample mean estimate $\mathbb{E}[\hat{L}(f)]$ is equal to the true loss $L(f)$, and its variance decreases as the number of samples n increases. More precisely, in Chapter 5 we showed that for a fixed predictor $f \in \mathcal{F}$,

$$\mathbb{E}[\hat{L}(f)] = L(f) \quad \text{and} \quad \mathrm{Var}[\hat{L}(f)] = \frac{1}{n}\mathrm{Var}[\ell(f(\boldsymbol{X}_1), Y_1)]. \tag{7.2}$$

This result can be understood as: if you first choose a predictor $f \in \mathcal{F}$ (shown in Fig. 7.1a), and then gather your dataset $D \sim \mathbb{P}_D$ (shown in Fig. 7.1b),



(a) Picking a predictor $f \in \mathcal{F}$ first.

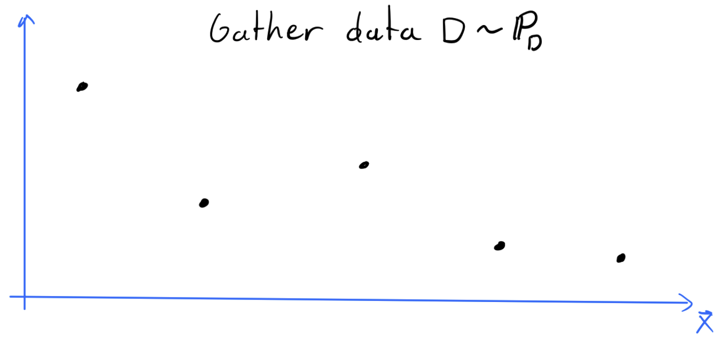


(b) Gathering a dataset D after picking a predictor $f \in \mathcal{F}$.

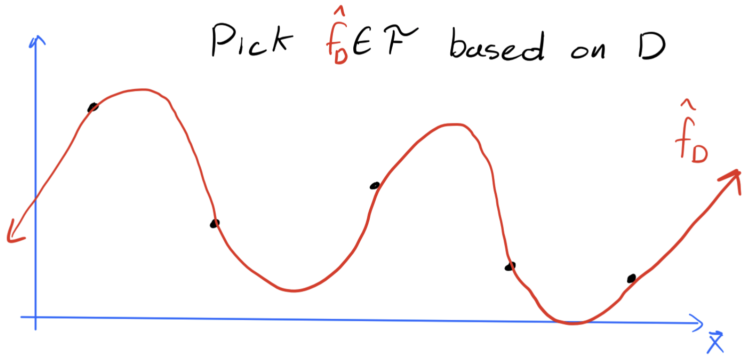
Figure 7.1: Picking a predictor f independent of the dataset D.

then Eq. (7.2) holds (i.e. how well the predictor f fits the dataset D is representitive of how it will fit new data). Formally, if $f \in \mathcal{F}$ is chosen idependently of the dataset D, then Eq. (7.2) holds.

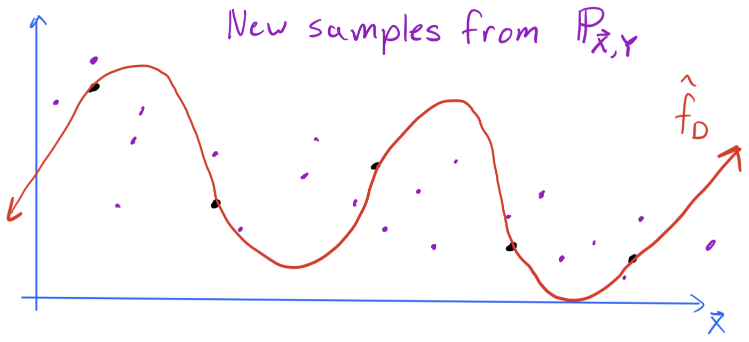
However, in our case the predictor $\hat{f}_D$ depends on the random dataset D. We can think about the learner as: first gathering the dataset $D \sim \mathbb{P}_D$ (shown in Fig. 7.2a), and then choosing the predictor $\hat{f}_D \in \mathcal{F}$ that best fits the data (shown in Fig. 7.2b). If we then gather new data sampled from the same distribution $\mathbb{P}_{\boldsymbol{X},Y}$ (shown in Fig. 7.2c), then we see that the predictor $\hat{f}_D$ does not fit the new data very well, which is often called *overfitting*.



(a) Gathering a dataset D first.



(b) Pick $\hat{f}_D \in \mathcal{F}$ after gathering a dataset D.



(c) Gathering new data according to $\mathbb{P}_{\boldsymbol{X},Y}$.

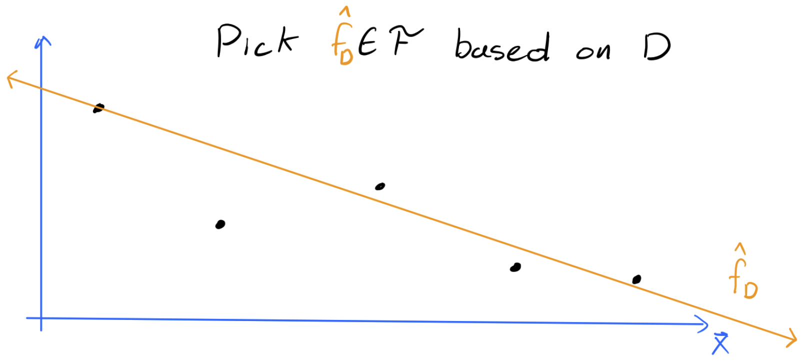
Figure 7.2: Picking a predictor $\hat{f}_D$ that best fits the dataset D, and then gathering new data.

Since $\hat{f}_D$ has overfit to the dataset we would find that $\hat{L}(\hat{f}_D)$ is smaller than $L(\hat{f}_D)$. If the original dataset we gathered was different, but we again chose the predictor $\hat{f}_D$ that best fits the data, we would probably again find that $\hat{L}(\hat{f}_D)$ is smaller than $L(\hat{f}_D)$. Thus, in expectation over datasets we would find that $\hat{L}(\hat{f}_D) \leq L(\hat{f}_D)$ holds, giving us the following inequality:

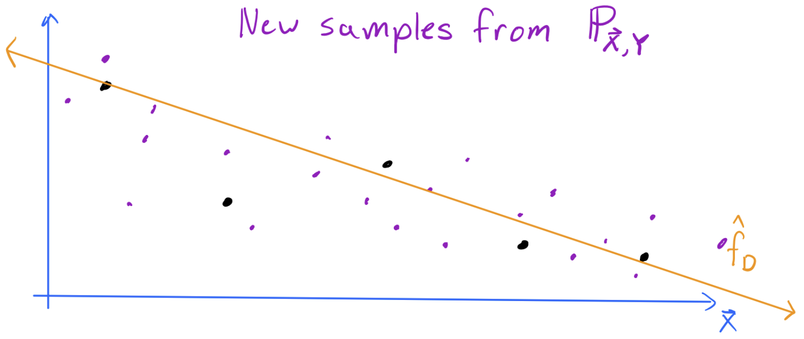
$$\mathbb{E}[\hat{L}(\hat{f}_D)] \leq \mathbb{E}[L(\hat{f}_D)].$$

Notice that this is exactly Property 1.

You might have noticed that the predictor $\hat{f}_D$ must have come from a large function class $\mathcal{F}$ (say degree 10 polynomials $\mathcal{F}_{10}$) in order to fit the dataset so perfectly. Suppose that instead the predictor was chosen from a smaller function class. We would find that the predictor no longer fits the dataset as well as in the previous example (shown in Fig. 7.3a), which means the estimated loss $\hat{L}(\hat{f}_D)$ would be larger. However, if we then gathered new data, we would find that the predictor fits the new data better than in the previous example (shown in Fig. 7.3b), and means the expected loss $L(\hat{f}_D)$ would be smaller.



(a) Pick $\hat{f}_D \in \mathcal{F}$ after gathering a dataset D.



(b) Gathering new data according to $\mathbb{P}_{\boldsymbol{X},Y}$.

Figure 7.3: Picking a predictor $\hat{f}_D$ that best fits the dataset D, and then gathering new data.

If we then take the expectation over datasets we would find that if the function class gets smaller, then $\mathbb{E}[L(\hat{f}_D)]$ decreases, while $\mathbb{E}[\hat{L}(\hat{f}_D)]$ increases. The opposite is of course true if we make $\mathcal{F}$ larger, which gives us Property 2. To visualize this property we can fix a dataset size (say $n = 100$) and consider the polynomial degree p function classes $\mathcal{F}_p$ and let $\hat{f}_{D,p} = \operatorname{argmin}_{f \in \mathcal{F}_p} \hat{L}(f)$, then we can plot $\mathbb{E}[\hat{L}(\hat{f}_{D,p})]$ and $\mathbb{E}[L(\hat{f}_{D,p})]$ as a function of the polynomial degree p (shown in Fig. 7.4).

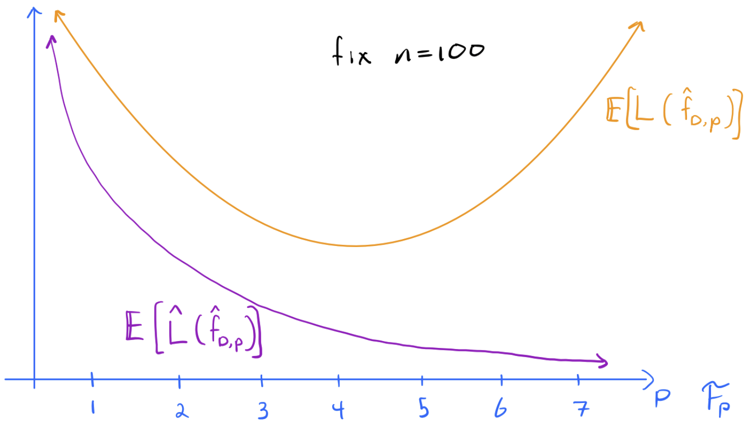


Figure 7.4: $\mathbb{E}[\hat{L}(\hat{f}_{D,p})]$ *and* $\mathbb{E}[L(\hat{f}_{D,p})]$ *as a function of the polynomial degree* p *for a fixed dataset size* $n = 100$.

Finally, if we have a large dataset (i.e. large n), then we have many samples that are likely to be representative of the true distribution $\mathbb{P}_{\boldsymbol{X},Y}$. This means that even if the predictor $\hat{f}_D$ is chosen to best fit the data D, it is likely to also fit new data well. Thus, if n increases we would expect $\mathbb{E}[\hat{L}(\hat{f}_D)]$ to get closer to $\mathbb{E}[L(\hat{f}_D)]$, which gives us Property 3. To visualize this property we can fix a function class (say degree 4 polynomials $\mathcal{F}_4$) and plot $\mathbb{E}[\hat{L}(\hat{f}_{D,4})]$ and $\mathbb{E}[L(\hat{f}_{D,4})]$ as a function of the dataset size n (shown in Fig. 7.5).

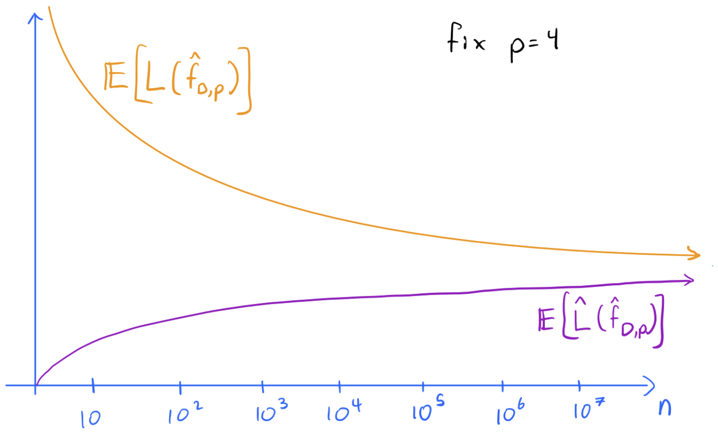


Figure 7.5: $\mathbb{E}[\hat{L}(\hat{f}_{D,4})]$ *and* $\mathbb{E}[L(\hat{f}_{D,4})]$ *as a function of the dataset size* n *for a fixed function class* $\mathcal{F}_4$.

The takeaway from this section is that if you want the estimated loss $\hat{L}$ of the predictor $\hat{f}_D$ to be close to its expected loss, then you should choose a small function class $\mathcal{F}$ and have a large dataset D. However, we will see in the next section that picking a small function class $\mathcal{F}$ limits how small the expected loss L can be, even for the best predictor $f \in \mathcal{F}$. Thus, picking the right function class $\mathcal{F}$ to use for the ERM learner will become a balancing act between these two objectives.

7.2 Estimation, Approximation, and Irreducible Error

Our goal in this section will be to understand how to select the function class $\mathcal{F}$ for the ERM learner such that $\mathbb{E}[L(\hat{f}_D)]$ is small. To do this we will decompose $\mathbb{E}[L(\hat{f}_D)]$ into three terms: the estimation error, the approximation error, and the irreducible error, and study how the function class affects each of these terms.

Before formally stating the decomposition of $\mathbb{E}[L(\hat{f}_D)]$, it will be helpful to first introduce some special predictors that will be used in the decomposition. Let us take a step back from the supervised learning setting we have been considering so far, and consider a simpler setting. Suppose the learner knew the true feature-label distribution $\mathbb{P}_{\boldsymbol{X},Y}$, and thus could compute the expected loss $L(f)$ for any predictor $f \in \{f | f : \mathcal{X} \to \mathcal{Y}\}$. Then, the best a learner can do is output

$$f_{\text{Bayes}} = \underset{f \in \{f | f : \mathcal{X} \to \mathcal{Y}\}}{\operatorname{argmin}} L(f).$$

This predictor is called the *Bayes optimal predictor*, hence the subscript in f_{Bayes}. This predictor is the best possible predictor that can be output by any learner, since it is the minimizer of the expected loss $L(f)$, and was chosen from an unrestricted set of functions $\{f | f : \mathcal{X} \to \mathcal{Y}\}$. To help with understanding the following predictors, let us suppose that f_{Bayes} is at least a degree 3 polynomial, as shown in Fig. 7.6.

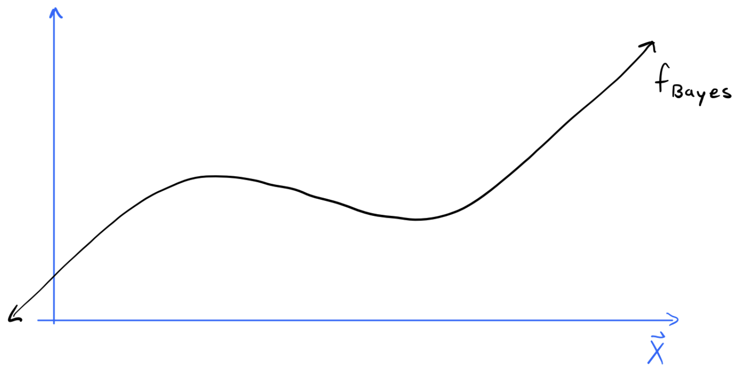


Figure 7.6: f_{Bayes}.

Now, suppose the learner still knew the true feature-label distribution $\mathbb{P}_{\boldsymbol{X},Y}$, but had to output a predictor from a restricted function class $\mathcal{F}$. Then, the best a learner can do is output the predictor that minimizes the expected loss $L(f)$ over the function class $\mathcal{F}$.

$$f^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} L(f).$$

If $f_{\text{Bayes}} \notin \mathcal{F}$, then f^* is not the same as f_{Bayes}, and $L(f_{\text{Bayes}}) < L(f^*)$. Continuing with the previous example, suppose that we use the degree 1 polynomials $\mathcal{F}_1$ as our function class, then f_{Bayes} is not in $\mathcal{F}_1$, and thus $f^* \neq f_{\text{Bayes}}$, as shown in Fig. 7.7.

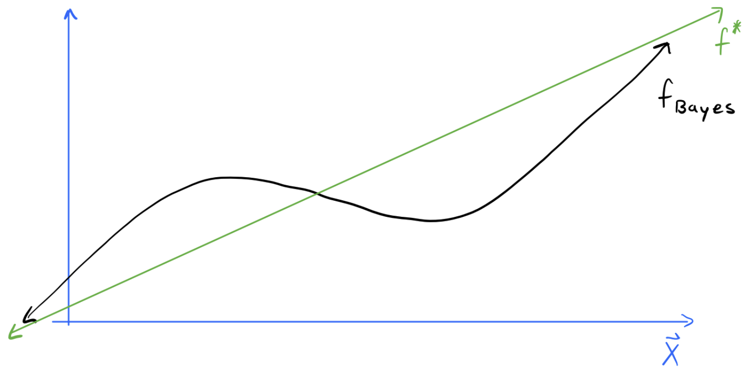


Figure 7.7: $f^* \in \mathcal{F}_1$ and f_{Bayes}.

Finally, suppose we are in the supervised learning setting (as defined in Chapter 4), where the learner does not know the true feature-label distribution $\mathbb{P}_{\boldsymbol{X},Y}$, but instead has access to a dataset D sampled from $\mathbb{P}_D$, and an estimated loss $\hat{L}$ based on the dataset. If the learner has to output a predictor from a restricted function class $\mathcal{F}$, and wants to minimize the estimated loss $\hat{L}$, then the best predictor the learner can output is

$$\hat{f}_D = \operatorname*{argmin}_{f \in \mathcal{F}} \hat{L}(f).$$

This is the same predictor as the one output by the ERM learner, and is the predictor we have been studying so far. It holds that $L(f^*) \leq L(\hat{f}_{\mathcal{D}})$ for all datasets $\mathcal{D}$ since $L(f^*) \leq L(f)$ for all $f \in \mathcal{F}$ and $\hat{f}_{\mathcal{D}}$ is an element of $\mathcal{F}$. Taking the expectation over datasets we have that $\mathbb{E}[L(f^*)] = L(f^*) \leq \mathbb{E}[L(\hat{f}_{\mathcal{D}})]$, where the first equality holds since $L(f^*)$ is a constant. The important thing to notice is that $\mathbb{E}[L(\hat{f}_D)] \geq L(f^*) \geq L(f_{\text{Bayes}})$. To see how the predictors might look we return to previous example. In Fig. 7.8 we have added $\hat{f}_{\mathcal{D}}$, which is based on a specfic dataset $\mathcal{D}$ of size $n = 5$ and the linear function class $\mathcal{F}_1$.

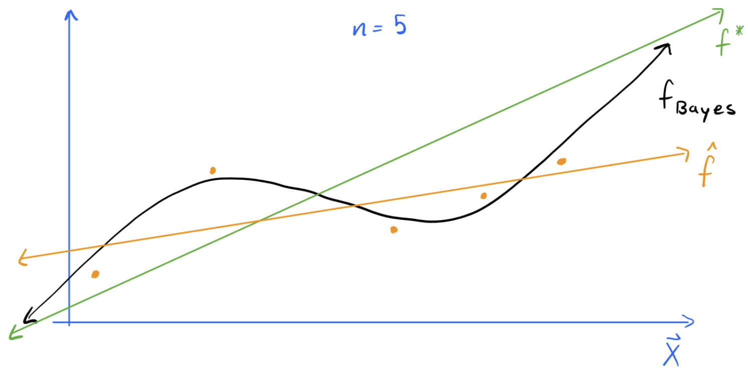


Figure 7.8: $\hat{f}, f^* \in \mathcal{F}_1$ and f_{Bayes}.

Since f_{Bayes} is the best possible predictor a learner can output, we would like to understand how to make $\hat{f}_D$ as close to f_{Bayes} as possible (in expectation over datasets), or equivalently how to make $\mathbb{E}[L(\hat{f}_D)]$ as close to $L(f_{\text{Bayes}})$ as possible. Understanding how to do this naturally leads us to decomposing $\mathbb{E}[L(\hat{f}_D)]$ as follows:

$$\mathbb{E}[L(\hat{f}_D)] = \underbrace{\mathbb{E}[L(\hat{f}_D)] - L(f^*)}_{\text{Estimation Error (EE)}} + \underbrace{L(f^*) - L(f_{\text{Bayes}})}_{\text{Approximation Error (AE)}} + \underbrace{L(f_{\text{Bayes}})}_{\text{Irreducible Error (IE)}} .$$

The error terms can be understood as follows:
Irreducible Error (IE): The is the lowest possible expected loss that can be achieved by any predictor, and is due to the inherent randomness in the feature-label distribution $\mathbb{P}_{\boldsymbol{X},Y}$. That is, if for some feature vector $\mathbf{x}$ there is a non zero probability that the label is y_1 and a non zero probability the label is y_2, then the predictor must suffer some loss for this feature vector. The only way to reduce the irreducible error is gather more informative features such that the randomness in the feature-label distribution is reduced. However, that is usually no possible, hence the name irreducible error.
Approximation Error (AE): This is the difference between the expected loss of the best predictor in the function class $\mathcal{F}$ and the expected loss of the best possible predictor. This error should be thought of as the price the learner pays for restricting the predictor it outputs to be an element of the function class $\mathcal{F}$. This error decreases as the function class $\mathcal{F}$ gets larger.
Estimation Error (EE): This is the difference between the expected loss of the predictor output by the learner averaged over datasets and the expected loss of the best predictor in the function class $\mathcal{F}$. This error should be thought of as the price learner pays for using the estimated loss $\hat{L}$ to choose the predictor. To see how to reduce this error it is helpful to decompose this error further:

$$\begin{aligned}\mathbb{E}[L(\hat{f}_D)] - L(f^*) &= \mathbb{E}[L(\hat{f}_D)] - \mathbb{E}[\hat{L}(\hat{f}_D)] + \mathbb{E}[\hat{L}(\hat{f}_D)] - \mathbb{E}[\hat{L}(f^*)] + \mathbb{E}[\hat{L}(f^*)] - L(f^*) \\ &= \mathbb{E}[L(\hat{f}_D)] - \mathbb{E}[\hat{L}(\hat{f}_D)] + \mathbb{E}[\hat{L}(\hat{f}_D) - \hat{L}(f^*)] + \mathbb{E}[\hat{L}(f^*)] - L(f^*) \\ &\leq \mathbb{E}[L(\hat{f}_D)] - \mathbb{E}[\hat{L}(\hat{f}_D)] + \mathbb{E}[\hat{L}(f^*)] - L(f^*). \end{aligned} \tag{7.3}$$

The inequality holds since $\hat{f}_D$ is the minizer of $\hat{L}$, so $\hat{L}(\hat{f}_D) \leq \hat{L}(f)$ for all $f \in \mathcal{F}$, and thus $\hat{L}(\hat{f}_D) \leq \hat{L}(f^*)$ since $f^* \in \mathcal{F}$. The reason this decomposition is useful is because the first term $\mathbb{E}[L(\hat{f}_D)] - \mathbb{E}[\hat{L}(\hat{f}_D)]$ is something we have already discussed in detail in Section 7.1. In particular we have seen that this term is non-negative, increases if the function class $\mathcal{F}$ gets larger, and decreases as the number of samples n increases. For the second term, since $f^* \in \mathcal{F}$ is chosen independently of the dataset D, we have that $\mathbb{E}[\hat{L}(f^*)] = L(f^*)$ and that $\text{Var}[\hat{L}(f^*)] = \frac{1}{n}\text{Var}[\ell(f^*(\boldsymbol{X}_1), Y_1)]$ (also discussed in Section 7.1). Roughly speaking, since the variance of $\hat{L}(f^*)$ decreases as n increases it implies that the second term also decreases as n increases. Thus, the estimation error decreases if n increases, but increases if the function class $\mathcal{F}$ gets larger.

The important thing to notice is that there is a trade-off between the approximation error and the estimation error in terms of how to select the function class $\mathcal{F}$. In particular if you make the function class $\mathcal{F}$ larger, then the approximation error decreases, but the estimation error increases. While, if you make the function class $\mathcal{F}$ smaller, then the approximation error increases, but the estimation error decreases. We can visualize this trade-off by plotting $L(f_{\text{Bayes}}), L(f^*), \mathbb{E}[L(\hat{f}_{D,p})]$, and $\mathbb{E}[\hat{L}(\hat{f}_{D,p})]$ as a function of the polynomial degree p for a fixed n (shown in Fig. 7.9), where $\hat{f}_{D,p} = \text{argmin}_{f \in \mathcal{F}_p} \hat{L}(f)$.

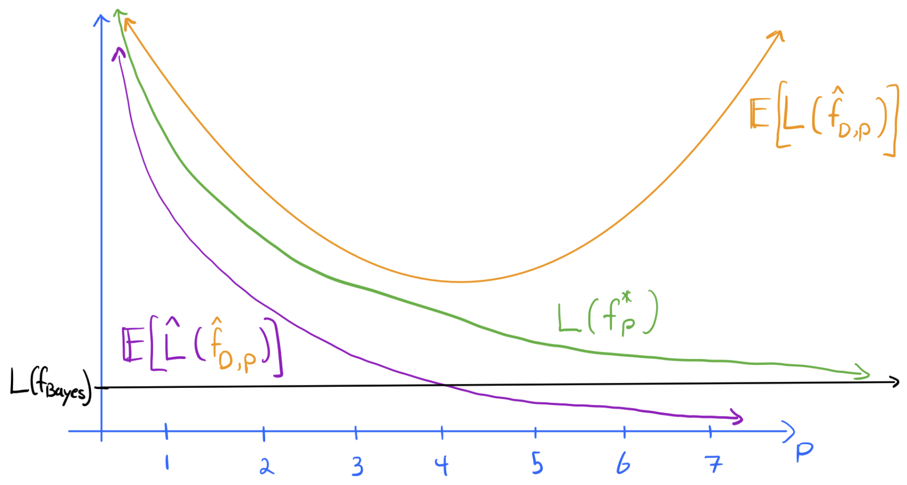


Figure 7.9: $L(f_{Bayes}), L(f^*), \mathbb{E}[L(\hat{f}_D)]$, and $\mathbb{E}[\hat{L}(\hat{f}_D)]$ as a function of the polynomial degree p for a fixed n.

We can see that $L(f_{\text{Bayes}})$ is the smallest, and does not change with p, while $L(f^*)$ approaches $L(f_{\text{Bayes}})$ as p increases. We can also see that when the function class is small (p is small) the approximation error (difference between $L(f_{\text{Bayes}})$ and $L(f^*)$) is large, but the estimation error (difference between $\mathbb{E}[L(\hat{f}_{D,p})]$ and $\mathbb{E}[\hat{L}(\hat{f}_{D,p})]$) is small. When this is the case, the predictor $\hat{f}_{D,p}$ is said to be *underfitting* the data. On the other hand, when the function class is large (p is large) the approximation error is small, but the estimation error is large. When this is the case, the predictor $\hat{f}_{D,p}$ is said to be *overfitting* the data. Also, notice how $\mathbb{E}[L(\hat{f}_{D,p})] - \mathbb{E}[\hat{L}(\hat{f}_{D,p})]$ has the same behaviour as the estimation error, as expected, due to the expression we derived in Eq. (7.3). The best choice of function class to use for a learner would be the one that minimizes $\mathbb{E}[L(\hat{f}_{D,p})]$, over polynomial degree p. In the plot this would correspond to the function class $\mathcal{F}_4$, resulting in a learner that outputs $\hat{f}_{D,4} = \text{argmin}_{f \in \mathcal{F}_4} \hat{L}(f)$. The same plot, but annotated with the observations made in this paragraph is shown in Fig. 7.10.

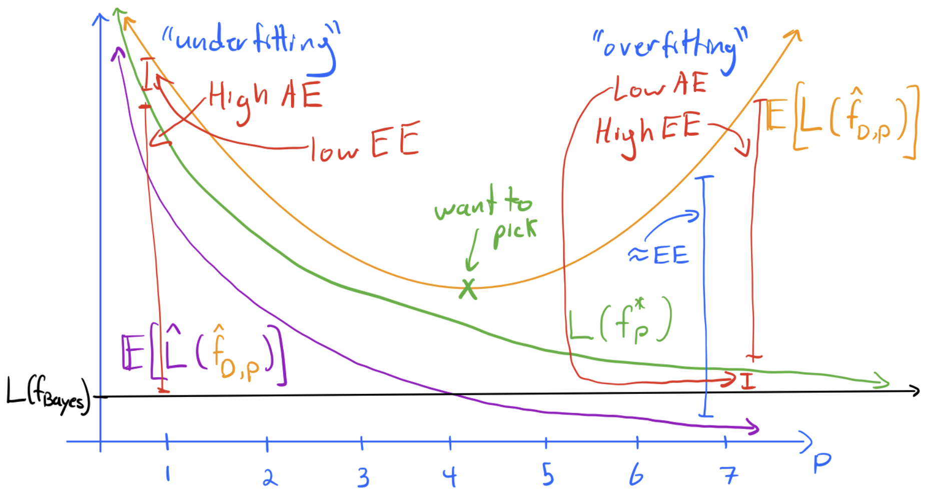


Figure 7.10: Annotated version of $L(f_{Bayes}), L(f^*), \mathbb{E}[L(\hat{f}_D)]$, and $\mathbb{E}[\hat{L}(\hat{f}_D)]$ as a function of the polynomial degree p for a fixed n.

A final observation to make from the decomposition of $\mathbb{E}[L(\hat{f}_D)]$ is that increasing the number of samples n in the dataset D will decrease the estimation error, with no affect on the other error terms, and is thus always benefecial to do if possible. This is shown in Fig. 7.11 where we again plot $L(f_{\text{Bayes}}), L(f^*), \mathbb{E}[L(\hat{f}_{D,p})]$, and $\mathbb{E}[\hat{L}(\hat{f}_{D,p})]$ as a function of the dataset size n for a fixed p.

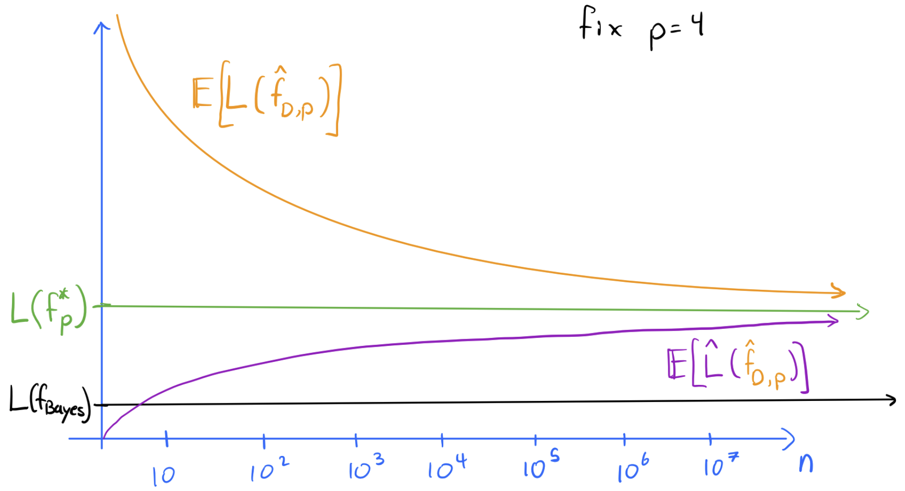


Figure 7.11: $L(f_{Bayes}), L(f^*), \mathbb{E}[L(\hat{f}_D)]$, and $\mathbb{E}[\hat{L}(\hat{f}_D)]$ as a function of the dataset size n for a fixed p.

We again see that indeed, the behaviour of $\mathbb{E}[L(\hat{f}_{D,p})] - \mathbb{E}[\hat{L}(\hat{f}_{D,p})]$ is the same as the estimation error.

Exercise 7.1: Suppose that you have a small dataset, but a large function class. Would the estimation error be large or small? Would you expect the approximation error to be large or small? Would expect the predictor $\hat{f}_D$ to be underfitting or overfitting the data or neither? $\square$

Exercise 7.2: Suppose that you have a large dataset, but a small function class, and f_{Bayes} is much more complex than any function in the function class. Would the estimation error be large or small? Would you expect the approximation error to be large or small? Would expect the predictor $\hat{f}_D$ to be underfitting or overfitting the data or neither? $\square$

Exercise 7.3: Suppose that you have a large dataset, but a small function class $\mathcal{F}$, and $f_{\text{Bayes}} \in \mathcal{F}$. Would the estimation error be large or small? Would you expect the approximation error to be large or small? Would expect the predictor $\hat{f}_D$ to be underfitting or overfitting the data or neither? $\square$

7.3 Selecting a Function Class

The careful reader might have noticed that it is not possible to do the procedure we described in the previous section for selecting a function class to use for the ERM learner. In particular, we said that the best choice of function class to use for a learner would be the one that minimizes $\mathbb{E}[L(\hat{f}_{D,p})]$, over polynomial degree p. However, we do not have access to the true feature-label distribution $\mathbb{P}_{\boldsymbol{X},Y}$, and thus cannot compute the expected loss $L(\hat{f}_{D,p})$. We also only have access to a single dataset $\mathcal{D}$ and cannot compute an expectation over datasets. Is our previous discussion of no use then? It turns out that estimation comes to save the day! If we had a good estimate of the expected loss $L(\hat{f}_{D,p})$ then we could just use that instead. Unfortunately, as we have seen in Section 7.1, the estimated loss $\hat{L}(\hat{f}_{D,p})$ is not very good, especially for large function classes, since the predictor $\hat{f}_{D,p}$ was chosen based on the dataset D. One option would then be to gather a new dataset D_{test}, and use that to estimate the expected loss $L(\hat{f}_{D,p})$. The reason this is a good idea is because the predictor $\hat{f}_{D,p}$ is chosen independently of D_{test}, thus an estimate of the expected loss $L(\hat{f}_{D,p})$ based on D_{test} will be good (as discussed in Section 7.1).

The final issue to address, is that in the supervised learning setting we only have access to one dataset D, and cannot gather a new dataset D_{test}. The way to resolve this issue is to split the dataset D into two parts: a training and test set:

$$D_{\text{train}} = ((\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_{n-m}, Y_{n-m})) \quad \text{and} \quad D_{\text{test}} = ((\boldsymbol{X}_{n-m+1}, Y_{n-m+1}), \ldots, (\boldsymbol{X}_n, Y_n)).$$

The learner should only use the training set D_{train} to choose the predictor $\hat{f}_{D_{\text{train}},p}$. In particular, the ERM learner should now use

$$\hat{L}_{\text{train}}(f) = \frac{1}{n-m} \sum_{i=1}^{n-m} \ell(f(\boldsymbol{X}_i), Y_i)$$

as its loss estimate. This way the test set D_{test} can be used to estimate the expected loss $L(\hat{f}_{D_{\text{train}},p})$, since the predictor $\hat{f}_{D_{\text{train}},p}$ was chosen independently of D_{test}. We will denote the estimated loss of a predictor $f \in \mathcal{F}$ based on the test set D_{test} as

$$\hat{L}_{\text{test}}(f) = \frac{1}{m} \sum_{i=n-m+1}^{n} \ell(f(\boldsymbol{X}_i), Y_i).$$

We can then plot $\hat{L}_{\text{train}}(\hat{f}_{\mathcal{D}_{\text{train}},p})$ and $\hat{L}_{\text{test}}(\hat{f}_{\mathcal{D}_{\text{train}},p})$ as a function of the polynomial degree p for a specific dataset $\mathcal{D}$ (shown in Fig. 7.12).

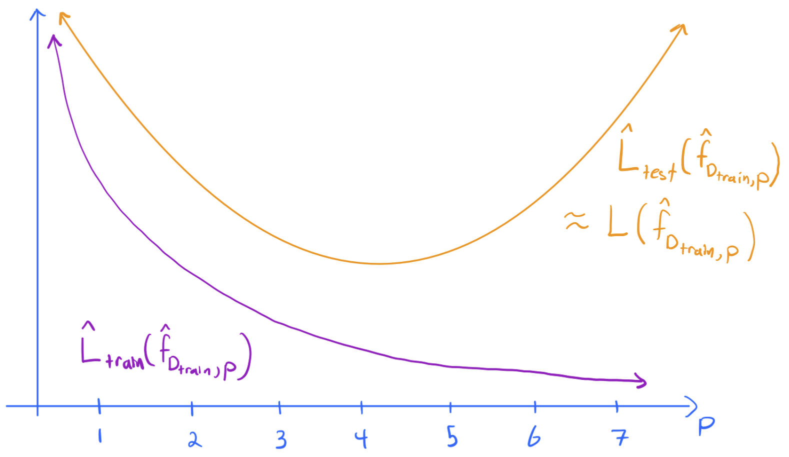


Figure 7.12: $\hat{L}_{train}(\hat{f}_{\mathcal{D}_{train},p})$ and $\hat{L}_{test}(\hat{f}_{\mathcal{D}_{train},p})$ as a function of the polynomial degree p for a specific dataset $\mathcal{D}$.

In this plot, and for most instances of the dataset $\mathcal{D}$ we expect to see a similar behaviour as we saw for $\mathbb{E}[L(\hat{f}_{D,p})]$ and $\mathbb{E}[\hat{L}(\hat{f}_{D,p})]$ in Fig. 7.9.

What is important, is that we actually have access to $\hat{L}_{\text{test}}(\hat{f}_{\mathcal{D}_{\text{train}},p})$ and can use that to select the function class $\mathcal{F}_p$. Putting things together, the procedure for selecting a predictor is as follows:

1. Split the fixed dataset $\mathcal{D}$ into a training set $\mathcal{D}_{\text{train}}$ and a test set $\mathcal{D}_{\text{test}}$.
2. For each polynomial degree $p \in \{1, \ldots, p_{\max}\}$, get the predictor $\hat{f}_{\mathcal{D}_{\text{train}},p}$ output by the ERM learner $\mathcal{A}(\mathcal{D}_{\text{train}})$ that uses the function class $\mathcal{F}_p$. The value of $p_{\max}$ should be chosen based on the complexity of the problem, and the computational resources available.
3. Calculate the estimated loss $\hat{L}_{\text{test}}(\hat{f}_{\mathcal{D}_{\text{train}},p})$ for each $p \in \{1, \ldots, p_{\max}\}$.
4. Select the predictor $\hat{f}_{\mathcal{D}_{\text{train}},p}$ that minimizes $\hat{L}_{\text{test}}(\hat{f}_{\mathcal{D}_{\text{train}},p})$ over values of $p \in \{1, \ldots, p_{\max}\}$.

7.4 Bias and Variance Tradeoff

The procedure described in the previous section can work well; however, the choice of function class $\mathcal{F}_p$ had do be done in discrete steps of polynomial degree p. Consider the case where the best function class is something between $\mathcal{F}_4$ and $\mathcal{F}_5$, that contains some some degree 5 polynomial features, but not all. We do not have such fine grained control with the parameter p. In the next section we will see that a technique called regularization can be thought of as a way to get this fine grained control. But in order to understand the effects of regularization, it we will be useful to first introduce the bias and variance tradeoff. In Section 7.2 we saw one way to decompose the $\mathbb{E}[L(\hat{f}_D)]$ into three terms: the estimation error, the approximation error, and the irreducible error. In this section we will see another way to decompose $\mathbb{E}[L(\hat{f}_D)]$ into three terms: the variance, the bias, and the irreducible error.

For the decomposition to work it is necessary to assume that the loss function ℓ is the squared loss. Then, after some slightly technical steps it is possible to show that the following holds:

$$\begin{aligned}\mathbb{E}[L(\hat{f}_D)] &= \mathbb{E}\Big[\underbrace{\mathbb{E}[(\hat{f}_D(\boldsymbol{X}) - \bar{f}(\boldsymbol{X}))^2|\boldsymbol{X}]}_{\text{Variance}}\Big] + \mathbb{E}\Big[\underbrace{(\bar{f}(\boldsymbol{X}) - f_{\text{Bayes}}(\boldsymbol{X}))}_{\text{Bias}}{}^2\Big] + \underbrace{L(f_{\text{Bayes}})}_{\text{Irreducible Error}} \\ &= \mathbb{E}\Big[\mathrm{Var}[\hat{f}_D(\boldsymbol{X})|\boldsymbol{X}]\Big] + \mathbb{E}\Big[\mathrm{Bias}[\bar{f}(\boldsymbol{X})]^2\Big] + L(f_{\text{Bayes}}).\end{aligned}$$

The function $\bar{f}(\boldsymbol{X})$ is called the *expected predictor* and is defined as $\bar{f}(\boldsymbol{X}) = \mathbb{E}[\hat{f}_D(\boldsymbol{X})|\boldsymbol{X}]$. You should think of $\bar{f}$ as the predictor you would get if you averaged over all possible datasets D, hence the expected predictor. The irreducible error term is the same as before, but the bias and variance terms are new. The variance describes the average squared deviation of the predictor $\hat{f}_D$ from the expected predictor $\bar{f}$. While the bias describes how different the expected predictor $\bar{f}$ is from the best possible predictor f_{Bayes}.

A nice property to know is that if the predictor $\hat{f}_D = \operatorname{argmin}_{f\in\mathcal{F}} \hat{L}(f)$ (and some other minor conditions), then the expected predictor $\bar{f} = f^* = \operatorname{argmin}_{f\in\mathcal{F}} \mathbb{E}[L(f)]$. If $\bar{f} = f^*$, then it can be shown that the variance term is equal to the estimation error and the bias term is equal to the approximation error. More precisely, if $\bar{f} = f^*$, then the following holds:

$$\mathbb{E}\Big[\mathrm{Var}[\hat{f}_D(\boldsymbol{X})|\boldsymbol{X}]\Big] = \mathbb{E}[L(\hat{f}_D)] - L(f^*) \quad \text{and} \quad \mathbb{E}\Big[\mathrm{Bias}[\bar{f}(\boldsymbol{X})]^2\Big] = L(f^*) - L(f_{\text{Bayes}}).$$

Thus, under the condition that $\bar{f} = f^*$ the effect of changing the function class $\mathcal{F}$ or the number of samples n has the same effect on the variance and bias terms as it does on the estimation and approximation error terms. In particular, the following properties holds:
Bias: Decreases as the function class $\mathcal{F}$ gets larger.
Variance: Increases as the function class $\mathcal{F}$ gets larger, and decreases as the number of samples n increases.

It turns out that the above properties hold in general, even if $\bar{f} \neq f^*$. The term underfitting is then used to describe the case where the bias is large, but the variance is small. While, the term overfitting is used to describe the case where the bias is small, but the variance is large.

7.5 Regularization

Suppose that $f_p(\mathbf{x}) = \phi_p(\mathbf{x})^\top \mathbf{w}$ is a polynomial of degree p and $\mathbf{w} = (w_0, w_1, \dots, w_{\bar{p}-1}) \in \mathbb{R}^{\bar{p}}$ is the weight vector. One motivation behind introducing regularization is based on the observation that if the values of the weights w_i are large, then the predictor f_p will be more complex [?]. Roughly speaking the complexity of a predictor can be thought of as the degree of the lowest degree polynomial that can be used to approximate the predictor well. For instance, in Fig. 7.13 the predictor g is less complex since it can be well approximated by a line (a degree 1 polynomial), while the predictor f is more complex since it cannot be well approximated by a line, and likely needs at least a degree 4 polynomial.

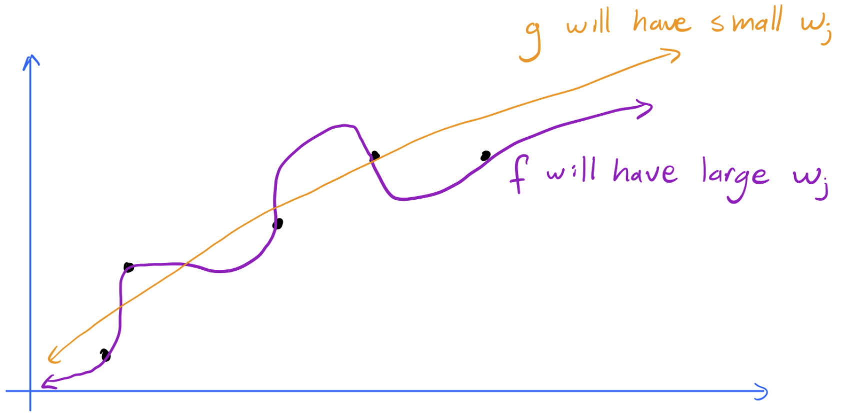


Figure 7.13: A complex and a simple predictor.

Since we can think of the value of the weights as a measure of the complexity of the predictor, we can introduce a new term that penalizes the weights for being large in the estimated loss that the ERM learner is trying to minimize. In particular, the regularized estimated loss for a predictor $f_p \in \mathcal{F}_p$ becomes:

$$\hat{L}_\lambda(f_p) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_p(\mathbf{x}_i), y_i) + \frac{\lambda}{n}\sum_{j=1}^{\bar{p}-1} w_j^2,$$

The term $\frac{\lambda}{n}\sum_{j=1}^{\bar{p}-1} w_j^2$ is called the *regularizer*, and the parameter λ is called the *regularization parameter*. Notice that regularization is only applied to the weights w_j for $j \in \{1, \ldots, \bar{p}-1\}$ and not to the bias w_0. This is because the bias does not affect the slope or curvature of the predictor, and thus does not affect the complexity of the predictor. Minimizing this new estimated loss is sometimes called *ridge regression*.

If $\lambda = 0$ then regularized estimated loss is the same as the unregularized one $\hat{L}_0 = \hat{L}$. But if $\lambda > 0$ then the regularized estimated loss will penalize the weights being large. In this way the parameter λ gives us a smooth way to scale the complexity of the predictor. The function class $\mathcal{F}$ should be selected to be quite large (for instance $\mathcal{F}_{10}$), so that small values of λ will result in a complex predictor, and large values of λ will result in a simple predictor. Notice how the function class $\mathcal{F}$ does not change here, thus the approximation error is the same for all values of λ. Since large λ will result in a simple predictor, the expected predictor $\bar{f}$ will also be simple. In general, if $\lambda \neq 0$, then $\bar{f} \neq f^*$, and the bias will not be equal to the approximation error. Similarly, the variance will not be equal to the estimation error, when $\lambda \neq 0$. The behaviour of the bias and variance as a function of λ is as follows. If λ is large then the predictors $\hat{f}_D$ and $\bar{f}$ will both be simple, and since two simple predictors are likely to be similar, the variance will be small. However, since $\bar{f}$ is simple, it will likely be far from the best possible predictor f_{Bayes}, and thus the bias will be large. On the other hand if λ is small then the predictors $\hat{f}_D$ and $\bar{f}$ will both be complex, and since two complex predictors can be very different from each other, the variance will be large. However, since $\bar{f}$ is complex it will likely be close to the best possible predictor f_{Bayes}, and thus the bias will be small. We can see that the decreasing λ has the same effect on the bias and variance terms as increasing the function class $\mathcal{F}$; however, for slightly different reasons. Thus, the regularization parameter λ can be used to control the bias and variance of the predictor, and is a way to get a fine grained control over the complexity of the predictor.

We can follow a similar procedure as described in Section 7.3, but now use the regularized estimated loss $\hat{L}_{\text{train},\lambda}$ (based on the train set) instead of $\hat{L}_{\text{train}}$. Let $\hat{f}_{\mathcal{D}_{\text{train}},\lambda}$ be the predictor output by the learner that uses the regularized estimated loss $\hat{L}_{\text{train},\lambda}$. Since our goal is to minimize the expected loss $L(\hat{f}_{\mathcal{D}_{\text{train}},\lambda})$, we should select the predictor $\hat{f}_{\mathcal{D}_{\text{train}},\lambda}$ that minimizes the estimated loss based on the test set without regularization $\hat{L}_{\text{test}}$. We can plot this in the same way as before, but now as a function of the regularization parameter λ (shown in Fig. 7.14).

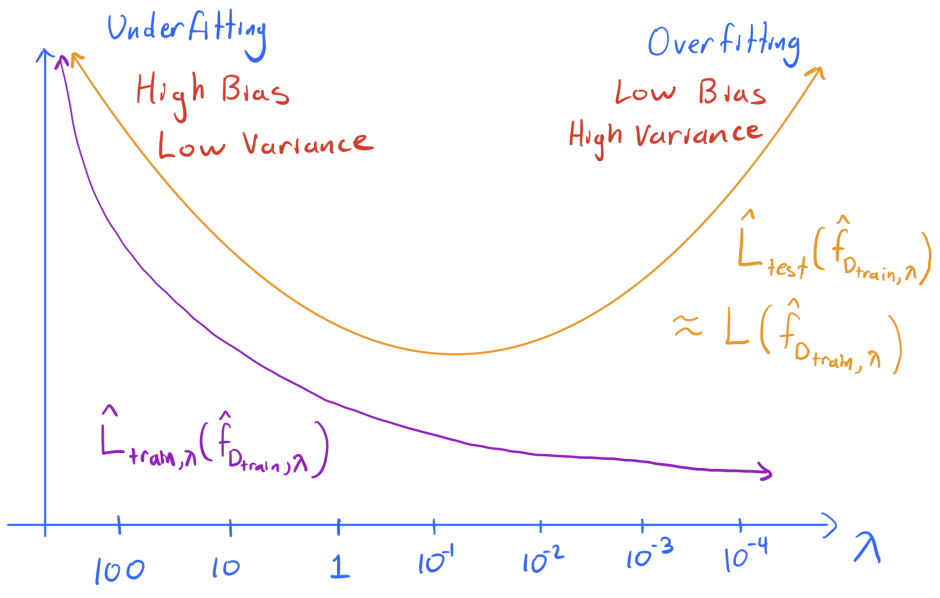


Figure 7.14: $\hat{L}_{train,\lambda}(\hat{f}_{\mathcal{D}_{train},\lambda})$ and $\hat{L}_{test}(\hat{f}_{\mathcal{D}_{train},\lambda})$ as a function of the regularization parameter λ for a specific dataset $\mathcal{D}$.

For the case shown in the plot we would select the predictor $\hat{f}_{\mathcal{D}_{\text{train}},0.1}$, since $\hat{L}_{\text{test}}(\hat{f}_{\mathcal{D}_{\text{train}},0.1})$ is the smallest.

What is left to show is how to minimize the regularized estimated loss $\hat{L}_{\lambda}$. To keep the notation simpler we minimize $\hat{L}_{\lambda}$, which uses the whole dataset $\mathcal{D}$, instead of just the training set $\mathcal{D}_{\text{train}}$, but can easily be applied to $\hat{L}_{\text{train},\lambda}$ by simply changing n to $n-m$ in the formulas.

It is possible to derive a closed form solution for the weights that minimize $\hat{L}_{\lambda}$; however, we will use batch gradient descent in this section. As we have done in Section 6.5.1, we will derive the gradient for the case when the function class is the set of all linear functions $\mathcal{F}_1$. Then the same exact result holds for the case when the function class is the set of all degree p polynomials $\mathcal{F}_p$, by simply replacing $\mathbf{x}$ with $\phi_p(\mathbf{x})$. Assume that the loss function is the squared loss, and let

$$\hat{L}_\lambda(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)^2 + \frac{\lambda}{n}\sum_{j=1}^{d} w_j^2$$

be the regularized estimated loss as a function of the weight vector $\mathbf{w}$. Recall that the batch gradient descent update rule is given by:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)}\nabla\hat{L}_\lambda(\mathbf{w}^{(t)}),$$

Thus, we just need to compute the gradient of the regularized estimated loss

$$\nabla\hat{L}_\lambda(\mathbf{w}) = \left(\frac{\partial\hat{L}_\lambda}{\partial w_0}(\mathbf{w}), \frac{\partial\hat{L}_\lambda}{\partial w_1}(\mathbf{w}), \ldots, \frac{\partial\hat{L}_\lambda}{\partial w_d}(\mathbf{w})\right)^\top.$$

Let $\hat{L}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)^2$, and $g(\mathbf{w}) = \frac{\lambda}{n}\sum_{j=1}^{d} w_j^2$. Then, we have that $\hat{L}_\lambda(\mathbf{w}) = \hat{L}(\mathbf{w}) + g(\mathbf{w})$. Then, for any $k \in \{0, 1, \ldots, d\}$ we have that:

$$\frac{\partial\hat{L}_\lambda}{\partial w_k}(\mathbf{w}) = \frac{\partial\hat{L}}{\partial w_k}(\mathbf{w}) + \frac{\partial g}{\partial w_k}(\mathbf{w}).$$

But, we have already calculated $\frac{\partial\hat{L}}{\partial w_k}(\mathbf{w})$ in Section 6.5.1, and it is given by:

$$\frac{\partial\hat{L}}{\partial w_k}(\mathbf{w}) = \frac{2}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)x_{ik}.$$

To calculate $\frac{\partial g}{\partial w_k}(\mathbf{w})$, notice that g does not depend on w_0, thus $\frac{\partial g}{\partial w_0}(\mathbf{w}) = 0$. For the case when $k \in \{1, \ldots, d\}$ we have that:

$$\frac{\partial g}{\partial w_k}(\mathbf{w}) = \frac{\lambda}{n}\frac{\partial}{\partial w_k}\sum_{j=1}^{d} w_j^2 = \frac{\lambda}{n}\sum_{j=1}^{d}\frac{\partial}{\partial w_k} w_j^2 = \frac{2\lambda}{n} w_k.$$

Putting things together we have that:

$$\frac{\partial\hat{L}_\lambda}{\partial w_k}(\mathbf{w}) = \begin{cases} \frac{2}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)x_{ik} & \text{if } k = 0, \\ \frac{2}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)x_{ik} + \frac{2\lambda}{n} w_k & \text{if } k \in \{1, \ldots, d\}. \end{cases}$$

Plugging this into gradient we get:

$$\begin{aligned}
\nabla\hat{L}_\lambda(\mathbf{w}) &= \left(\frac{2}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)x_{i0}, \frac{2}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)x_{i1} + \frac{2\lambda}{n} w_1, \ldots, \frac{2}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)x_{id} + \frac{2\lambda}{n} w_d\right)^\top \\
&= \frac{2}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)(x_i0, \ldots, x_id)^\top + \left(0, \frac{2\lambda}{n} w_1, \ldots, \frac{2\lambda}{n} w_d\right)^\top \\
&= \frac{2}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)\mathbf{x}_i + \left(0, \frac{2\lambda}{n} w_1, \ldots, \frac{2\lambda}{n} w_d\right)^\top.
\end{aligned}$$

The batch gradient descent update rule is then obtained by plugging the above gradient into the update rule.

7.6 Summary of the Different Errors

In this chapter we learned two common ways to decompose $\mathbb{E}[L(\hat{f}_D)]$. The first way was into the estimation error, approximation error, and irreducible error:

$$\mathbb{E}[L(\hat{f}_D)] = \underbrace{\mathbb{E}[L(\hat{f}_D)] - L(f^*)}_{\text{Estimation Error (EE)}} + \underbrace{L(f^*) - L(f_{\text{Bayes}})}_{\text{Approximation Error (AE)}} + \underbrace{L(f_{\text{Bayes}})}_{\text{Irreducible Error}} .$$

The second way was into the variance, bias, and irreducible error:

$$\mathbb{E}[L(\hat{f}_D)] = \mathbb{E}\Big[\underbrace{\mathbb{E}[(\hat{f}_D(\boldsymbol{X}) - \bar{f}(\boldsymbol{X}))^2|\boldsymbol{X}]}_{\text{Variance}}\Big] + \mathbb{E}\Big[\underbrace{(\bar{f}(\boldsymbol{X}) - f_{\text{Bayes}}(\boldsymbol{X}))^2}_{\text{Bias}}\Big] + \underbrace{L(f_{\text{Bayes}})}_{\text{Irreducible Error}} .$$

We learned that the only way to reduce the irreducible error is gather more informative features, which is often not possible.

We also studied how changing the size of the dataset n, the complexity of the polynomial function class p, and the regularization parameter λ affects the AE, EE, variance, and bias. In Table 7.1 we provide a summary of how changing n, p, and λ affects the different errors.

	Increasing n	**Increasing p with $\lambda = 0$**	**Decreasing λ with $p = 10$**
EE	Decreases	Increases	Unclear
AE	Unchanged	Decreases until $f_{\text{Bayes}} \in \mathcal{F}_p$	Unchanged
Variance	Decreases	Increases	Increases
Bias	Unchanged	Decreases until $f_{\text{Bayes}} \in \mathcal{F}_p$	Decreases

Table 7.1: How changing n, p, and λ affects the EE, AE, variance, and bias. The terms "Increasing" and "Decreasing" also encompass cases where values remain constant.

Some important takeaways from this table are:

- Increasing the size of the dataset n is always beneficial, since it decreases the EE and variance without affecting the AE and bias.
- If no regularization is used $\lambda = 0$, then EE and variance will increase, while the AE and bias will decrease as the complexity of the function class p increases.
- If the function class $\mathcal{F}_p$ contains f_{Bayes} then the AE is zero.
- Changing the regularization parameter λ has no effect on the AE since the function class $\mathcal{F}_p$ does not change.
- Decreasing λ has a complicated effect on the EE, since the predictor $\hat{f}_D$ will become more complex which means it can more closely approximate f^*, however, due to this complexity it will vary more from dataset to dataset. As such, the behaviour is complicated and we will not study it in these notes.

As a final remark, we will often claim that two functions get closer to each other as the error between the two of them gets smaller. For instance we might claim that the predictor f^* gets closer to f_{Bayes} if the AE decreases. Although the AE only compares the two functions through the expected loss L, it is often a good indicator that the two functions are close (i.e. $f^*(\mathbf{x}) \approx f_{\text{Bayes}}(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$).

Chapter 8

Maximium Likelihood and Maximium A Posteriori Estimation

So far we have only studied the ERM learner. In this chapter we study two new learners. The first is based on maximum likelihood estimation (MLE) and the second is based on maximum a posteriori estimation (MAP). We begin with MLE.

8.1 Maximum Likelihood Estimation (MLE)

Roughly speaking, our goal in supervised learning is to predict the label y correctly based on a new input feature vector $\mathbf{x}$. In Chapter 4 we argued that a predictor f that minimizes the expected loss $L(f)$ is a good choice. Since we do not know the true distibution over feature-label pairs $\mathbb{P}_{\boldsymbol{X},Y}$, we cannot compute the expected loss directly. ERM addressed this issue by estimating the expected loss L using the estimated loss $\hat{L}$.

Next, we present at different learner that does not estimate the expected loss L. Instead, recall that in Chapter 7 we defined the best possible predictor as

$$f_{\text{Bayes}} = \underset{f \in \{f | f : \mathcal{X} \to \mathcal{Y}\}}{\operatorname{argmin}} L(f).$$

We said this was the best possible predictor because it minimizes the expected loss $L(f)$ over the set of all possible predictors. If we assume we are in the regression setting and using the squared loss, then it is possible to show that the following holds:

$$f_{\text{Bayes}}(\mathbf{x}) = \mathbb{E}[Y | \boldsymbol{X} = \mathbf{x}] = \int_{\mathcal{Y}} y \cdot p_{Y|\boldsymbol{X}}(y|\mathbf{x})\, dy\,,$$

where the last equality is simply the definition of the conditional expectation. This means that the best possible predictor is the conditional expectation of the label given the feature vector. We can also see that in order to calculate f_{Bayes} we need to know the conditional pdf $p_{Y|\boldsymbol{X}}$. Thus, if a learner could estimate the conditional pdf $p_{Y|\boldsymbol{X}}$, then the learner could plug it into the above equation and obtain an estimate of f_{Bayes}. This will be the idea behind the MLE learner.

Before discussing how to estimate the conditional pdf $p_{Y|\boldsymbol{X}}$, we will first need to go over some MLE basics.

8.1.1 MLE Basics

Assume that their is a random dataset of n samples:

$$D = (Z_1, \dots Z_n) \in \mathcal{Z}^n,$$

with distribution $\mathbb{P}_D$ and pmf or pdf p_D. Each random sample Z_i is i.i.d. with distribution $\mathbb{P}_Z$ and pmf or pdf p_Z. If we are only given access to a fixed dataset $\mathcal{D} = (z_1, \ldots, z_n)$, then MLE is a method to estimate p_Z. To see how this is done, notice that since the samples are i.i.d., we can write the following:

$$p_D(\mathcal{D}) = p_D(z_1, \ldots, z_n) = p_Z(z_1) \cdots p_Z(z_n) = \prod_{i=1}^{n} p_Z(z_i),$$

where the second equality is due to the i.i.d. assumption. The last equality uses the product symbol $\prod$ which is defined as $\prod_{i=1}^{n} a_i = a_1 \cdots a_n$. Notice how $p_D(\mathcal{D})$ only depends on p_Z. We can think of $p_D(\mathcal{D})$ as representing how likely we are to get the dataset $\mathcal{D}$. MLE estimates p_Z by finding a pdf or pmf, p_{MLE}, that maximizes $p_D(\mathcal{D})$. That is, it chooses the pmf or pdf that maximizes the likelihood of the dataset $\mathcal{D}$, hence the name maximum likelihood estimation. Formally,

$$p_{\text{MLE}} = \underset{p \in \mathcal{H}}{\operatorname{argmax}} \prod_{i=1}^{n} p(z_i),$$

where $\mathcal{H}$ is the set of pdfs or pmfs that we are considering. Usually, MLE is considered for distributions parametrized by some parameter(s), for example: Bernoulli distribution, Gaussian distribution, etc.

Example 8.1: [MLE for Bernoulli.] Let $Z_1, \ldots, Z_n$ be i.i.d. random variables representing n flips of the same unfair coin. The distribution of each Z_i is given by Bernoulli(α^*), and the pmf is $p_Z(z) = (\alpha^*)^z \cdot (1 - \alpha^*)^{(1-z)}$. However, α^* is unknown, and thus p_Z is unknown. We would like to estimate p_Z using MLE. To do this, we must select the set of functions $\mathcal{H}$ that we are considering. For this example it is natural to consider the set of all Bernoulli distributions:

$$\mathcal{H} = \{p_\alpha \,|\, p_\alpha : \{0, 1\} \to [0, 1] \text{ and } p_\alpha(z) = \alpha^z \cdot (1 - \alpha)^{(1-z)} \text{ for some } \alpha \in [0, 1]\}.$$

Notice that any $p_\alpha \in \mathcal{H}$ can uniquely be identified by the parameter α, and has the form $p_\alpha(z) = \alpha^z \cdot (1 - \alpha)^{(1-z)}$. Thus we can rewrite the MLE solution as follows:

$$p_{\text{MLE}} = p_{\alpha_{\text{MLE}}} \quad \text{where} \quad \alpha_{\text{MLE}} = \underset{\alpha \in [0,1]}{\operatorname{argmax}} \prod_{i=1}^{n} p_\alpha(z_i).$$

The function $p_\alpha(z_i)$ is often written as $p(z_i|\alpha)$ to emphasize that it depends on the parameter α; however, it is important to note this is not a conditional distribution, since α is not a random variable. Then, the term $\prod_{i=1}^{n} p(z_i|\alpha)$ is called the *likelihood* of the dataset $\mathcal{D}$ given the parameter α, and is often written as $p(\mathcal{D}|\alpha)$. For a fixed dataset $\mathcal{D}$, the likelihood is seen as a function of the parameter(s), which is α in this case.

At this point we simply need to solve an optimization problem to find α_{MLE}. We have discussed optimization in detail in Chapter 6, and the techniques we learned there can be applied here.

To end this example we give a visual representation of what MLE is doing. Suppose that you had $n = 5$ flips of the coin with dataset $\mathcal{D} = (1, 0, 1, 0, 0)$. This is shown as the red Xs in Fig. 8.1.

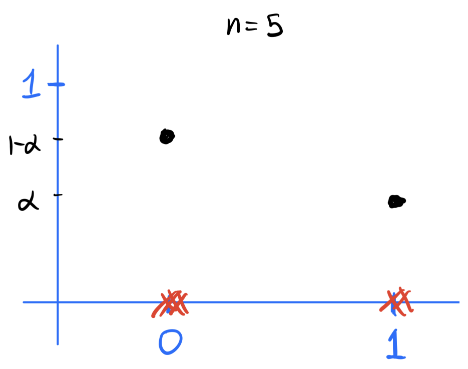


Figure 8.1: MLE for a unfair coin.

The MLE solution selects the pmf that maximizes the likelihood of seeing the 5 flips. Since there are 3 tails(0), but only 2 heads(1) in the dataset, the MLE solution will end up selecting a pmf that gives a higher probability to tails than heads as shown by the two black points in the plot. □

Exercise 8.1: Solve for α_{MLE} in Example 8.1. □

Example 8.2: [MLE for Gaussian.] Assume that the data $Z_1, \ldots, Z_n$ (e.g. height of a person) is i.i.d. with distribution $\mathcal{N}(\mu^*, 1)$ and pdf

$$p_Z(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(z-\mu^*)^2}{2}\right).$$

Given the dataset $\mathcal{D} = (z_1, \ldots, z_n)$, we would like to estimate μ^* using MLE, which will give us an estimate of the true mean μ^* so that $p_{\mu_{\text{MLE}}}(z) \approx p_Z$. The MLE for μ^* can be seen as sweeping through all possible values of μ and selecting the one that maximizes the likelihood of the dataset $\mathcal{D}$. Now we procede to find the MLE solution, where we will take the log of the likelihood to simplify the optimization problem.

$$\begin{aligned}
\mu_{\text{MLE}} &= \operatorname*{argmax}_{\mu\in\mathbb{R}} \prod_{i=1}^{n} p(z_i|\mu) \\
&= \operatorname*{argmax}_{\mu\in\mathbb{R}} \log\left(\prod_{i=1}^{n} p(z_i|\mu)\right) && \text{(since log is monotonic)} \\
&= \operatorname*{argmax}_{\mu\in\mathbb{R}} \sum_{i=1}^{n} \log(p(z_i|\mu)) && \text{(since } \log(ab) = \log(a) + \log(b)\text{)} \\
&= \operatorname*{argmin}_{\mu\in\mathbb{R}} -\sum_{i=1}^{n} \log(p(z_i|\mu)) && \text{(changing max to min)} \\
&= \operatorname*{argmin}_{\mu\in\mathbb{R}} \sum_{i=1}^{n} \frac{(z_i-\mu)^2}{2} && \text{(since } \log\tfrac{1}{\sqrt{2\pi}} \text{ does not depent on } \mu\text{)}
\end{aligned}$$

In the third equality we switched from maximizing to minimizing simply due to convention in optimization. The term $-\sum_{i=1}^{n} \log(p(z_i|\mu))$ is called the *negative log-likelihood*, and is often written as $-\log(p(\mathcal{D}|\mu))$. For most MLE problems, you will find that it is helpful to take the log of the likelihood to simplify the optimization problem. To solve the minimization problem we can take the derivative of the last expression with respect to μ and set it to zero.

$$\frac{d}{d\mu}\sum_{i=1}^{n}\frac{(z_i-\mu)^2}{2}=0 \quad \Rightarrow \quad \sum_{i=1}^{n}(z_i-\mu)=0 \quad \Rightarrow \quad \sum_{i=1}^{n} z_i = n\mu \quad \Rightarrow \quad \mu_{\text{MLE}}=\frac{1}{n}\sum_{i=1}^{n} z_i.$$

□

Exercise 8.2: Suppose that now the data $Z_1,\dots,Z_n$ is i.i.d. with distribution $\mathcal{N}(0,(\sigma^*)^2)$. Find the MLE solution for σ^*. □

Exercise 8.3: Suppose that the data $Z_1,\dots,Z_n$ is i.i.d. with a Poisson with parameter $\lambda^* \in [0,\infty)$. Although we did not learn about the Poisson distribution, all you will need to know about it for this exercise is that a Poisson random variables takes values $0,1,2,\dots$ and its the pmf is

$$p_Z(z)=\frac{(\lambda^*)^z}{z!}\exp(-\lambda^*).$$

Find the MLE solution for λ^*. □

8.1.2 MLE Learner

Now we come back to the question of estimating f_{Bayes} for which we need to estimate the conditional pdf $p_{Y|\boldsymbol{X}}$. The regression setting considers a random dataset $D=(\boldsymbol{X}_1,Y_1),\dots,(\boldsymbol{X}_n,Y_n)$, where $(\boldsymbol{X}_i,Y_i)$ are i.i.d. samples from $\mathbb{P}_{\boldsymbol{X},Y}$ with density $p_{\boldsymbol{X},Y}$, our goal is to estimate the conditional density $p_{Y|\boldsymbol{X}}$ using MLE. Recall from the previous section that to be able to use MLE, we need to assume that the data comes from a specific family of distributions over which we can optimize. Note that we do not have this assumption for the ERM learner. For MLE in the linear regression setting, we assume that

$$Y_i|\boldsymbol{X}_i=\mathbf{x}_i \sim \mathcal{N}(\mathbf{x}_i^\top\mathbf{w},1)\,,$$

where we want to find the parameter $\mathbf{w}$ using MLE. We start by optimizing over the joint density and relate it to the conditional density:

$$\begin{aligned}
\mathbf{w}_{\text{MLE}} &= \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^{d+1}} \prod_{i=1}^{n} p(\mathbf{x}_i, y_i|\mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^{d+1}} \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) \cdot p(\mathbf{x}_i) && \text{(product rule)} \\
&= \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^{d+1}} \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) && \text{(since } p(\mathbf{x}_i) \text{ does not depend on } \mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^{d+1}} \sum_{i=1}^{n} \log(p(y_i|\mathbf{x}_i, \mathbf{w})) && \text{(since log is monotonic)} \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} -\sum_{i=1}^{n} \log(p(y_i|\mathbf{x}_i, \mathbf{w})) \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} \sum_{i=1}^{n} \frac{(y_i - \mathbf{x}_i^\top \mathbf{w})^2}{2} && \text{(since } \log \tfrac{1}{\sqrt{2\pi}} \text{ does not depend on } \mathbf{w}) \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} \frac{n}{2} \hat{L}(\mathbf{w}) && \text{(by the defn of } \hat{L}) \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} \hat{L}(\mathbf{w}) \,.
\end{aligned}$$

The last line shows that the MLE solution $\mathbf{w}_{\text{MLE}}$ is the same as the closed form ERM solution $\hat{\mathbf{w}}$ in the regression setting with the squared loss (see Section 6.3). Finally, we can plug in $\mathbf{w}_{\text{MLE}}$ into the conditional pdf $p_{Y|\boldsymbol{X}}$ to get an estimate of f_{Bayes}:

$$\begin{aligned}
f_{\text{Bayes}}(\mathbf{x}) &= \mathbb{E}[Y|\boldsymbol{X} = \mathbf{x}] \\
&= \int_{\mathbb{R}} y \cdot p_{Y|\boldsymbol{X}}(y|\mathbf{x})\, dy \\
&\approx \int_{\mathbb{R}} y \cdot p_{Y|\boldsymbol{X}}(y|\mathbf{x}, \mathbf{w}_{\text{MLE}})\, dy \\
&= \mathbb{E}[Y'|\boldsymbol{X} = \mathbf{x}] && \text{(where } Y'|\boldsymbol{X} = \mathbf{x} \sim \mathcal{N}(\mathbf{x}^\top \mathbf{w}_{\text{MLE}}, 1)) \\
&= \mathbf{x}^\top \mathbf{w}_{\text{MLE}} \\
&= \mathbf{x}^\top \hat{\mathbf{w}} \\
&= \hat{f} \,.
\end{aligned}$$

This shows that the MLE learner is the same as the ERM learner in the regression setting with the squared loss. However, note that here we assumed that the data comes from a Gaussian distribution, which was not the case for the ERM learner.

8.2 Maximum A Posteriori Estimation (MAP)

In MLE, a key step to make it possible to estimate a pdf or pmf is to assume that the data comes from a specific distribution, which means the pdf or pmf can be written in terms of some parameter(s). The reason this was important was that we can then optimize over the parameter(s) to find the best pdf or pmf that makes the data the most likely. Maximum A Posteriori Estimation (MAP) is a different approach to estimating the parameter(s) of a pdf or pmf. In MAP, we assume that the parameter(s) of the pdf or pmf are random variables themselves. We estimate the parameters by finding the most likely values of the parameters given the data. More precisely, the MAP solution is

$$w_{\text{MAP}} = \operatorname*{argmax}_{w \in \mathcal{W}} p(w|\mathcal{D}).$$

The term $p(w|\mathcal{D})$ is called the *posterior* of w given the data $\mathcal{D}$. To help build some intuition and optimize the posterior it is helpful to expand $p(w|\mathcal{D})$:

$$\begin{aligned}
w_{\text{MAP}} &= \operatorname*{argmax}_{w \in \mathcal{W}} p(w|\mathcal{D}) \\
&= \operatorname*{argmax}_{w \in \mathcal{W}} \frac{p(\mathcal{D}|w) \cdot p(w)}{p(\mathcal{D})} && \text{(Bayes' rule)} \\
&= \operatorname*{argmax}_{w \in \mathcal{W}} p(\mathcal{D}|w) \cdot p(w). && \text{(since } p(\mathcal{D}) \text{ does not depend on } w)
\end{aligned}$$

The final expression shows that the MAP solution is the same as the MLE solution, but with an additional term $p(w)$. The term $p(w)$ is called the *prior* of w. The prior is a distribution over the parameter(s) that encodes our beliefs about the parameter(s) before seeing the data.

Before we use MAP to estimate the parameter(s) of $p_{Y|\boldsymbol{X}}$, we will first go over some basics MAP basics.

8.2.1 MAP Basics

Example 8.3: [MAP for Gaussian] Suppose that the data $Z_1, \ldots, Z_n$ is i.i.d. with distribution $\mathcal{N}(\mu^*, 1)$ and represents the height of a person. We would like to estimate μ^* using MAP. Suppose we have some prior knowledge that the mean height of a person is around 160 cm. We can then choose a gaussian prior with mean 160 and some variance σ^2 (which is also based on our prior knowledge and we will set later). This means that

$$p(\mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\mu - 160)^2}{2\sigma^2}\right).$$

Then the MAP solution is

$$\begin{aligned}
\mu_{\text{MAP}} &= \operatorname*{argmax}_{\mu \in \mathbb{R}} p(\mathcal{D}|\mu) \cdot p(\mu) \\
&= \operatorname*{argmin}_{\mu \in \mathbb{R}} -\log(p(\mathcal{D}|\mu)p(\mu)) \\
&= \operatorname*{argmin}_{\mu \in \mathbb{R}} \left[-\log(p(\mathcal{D}|\mu)) - \log(p(\mu))\right] \\
&= \operatorname*{argmin}_{\mu \in \mathbb{R}} \left[\sum_{i=1}^{n} \frac{(z_i - \mu)^2}{2} - \log(p(\mu))\right].
\end{aligned}$$

The last equality used the results from Example 8.2. If we just focus on the term $\log(p(\mu))$ and plug in the definition of $p(\mu)$, we get

$$\begin{aligned}\log(p(\mu)) &= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(\mu-160)^2}{2\sigma^2}\right)\right)\\ &= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(\mu-160)^2}{2\sigma^2}.\end{aligned}$$

Plugging this back into the MAP solution, we get

$$\begin{aligned}\mu_{\text{MAP}} &= \operatorname*{argmin}_{\mu\in\mathbb{R}}\left[\sum_{i=1}^{n}\frac{(z_i-\mu)^2}{2} - \log(p(\mu))\right]\\ &= \operatorname*{argmin}_{\mu\in\mathbb{R}}\left[\sum_{i=1}^{n}\frac{(z_i-\mu)^2}{2} - \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \frac{(\mu-160)^2}{2\sigma^2}\right]\\ &= \operatorname*{argmin}_{\mu\in\mathbb{R}}\left[\sum_{i=1}^{n}\frac{(z_i-\mu)^2}{2} + \frac{(\mu-160)^2}{2\sigma^2}\right].\end{aligned}$$

The last equality holds since the term $\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)$ does not depend on μ. To solve the optimization problem we can take the derivative of the last expression with respect to μ and set it to zero.

$$\begin{aligned}&\frac{d}{d\mu}\left[\sum_{i=1}^{n}\frac{(z_i-\mu)^2}{2} + \frac{(\mu-160)^2}{2\sigma^2}\right] = 0\\ \implies\ & -\sum_{i=1}^{n}(z_i-\mu) + \frac{(\mu-160)}{\sigma^2} = 0\\ \implies\ & n\mu - \sum_{i=1}^{n} z_i + \frac{\mu}{\sigma^2} - \frac{160}{\sigma^2} = 0\\ \implies\ & \mu_{\text{MAP}} = \frac{\sum_{i=1}^{n} z_i + \frac{160}{\sigma^2}}{n + \frac{1}{\sigma^2}}.\end{aligned}$$

It is insightful to study μ_{MAP} for different cases of σ^2 and n. In particular, we make the following observations:

1. If n is large, then μ_{MAP} will be approximately $\frac{1}{n}\sum_{i=1}^{n} z_i$, which is μ_{MLE} (from Example 8.2). This can be thought of as the prior having less influence on the MAP solution as the amount of data increases.

2. If σ^2 is small, then μ_{MAP} will be approximately 160. A small σ^2 means that we are confident in our prior knowledge, and thus the MAP solution will be close to the prior mean.

3. If σ^2 is large then μ_{MAP} will again be approximately μ_{MLE}. A large σ^2 means that we are not confident in our prior knowledge, and thus the MAP solution will be close to the MLE solution.

In the extreme case, if the prior is the pdf of the uniform distribution, then the MAP solution will be the same as the MLE solution, since the prior has no influence. □

Exercise 8.4: Suppose that the data $Z_1, \ldots, Z_n$ is i.i.d. with a Poisson with parameter $\lambda^* \in [0, \infty)$. Its pmf is

$$p_Z(z) = \frac{(\lambda^*)^z}{z!} \exp(-\lambda^*).$$

Suppose that we have some prior knowledge that the parameter λ^* follows a gamma distribution with parameters $k = 3$ and $\theta = 1$. A gamma random variable has a continuous distribution and it takes values in $(0, \infty)$[1]. Again, we will not go into the details of the gamma distribution, but the pdf is

$$p(\lambda) = \frac{\lambda^{k-1}}{(k-1)!\theta^k} \exp\left(-\frac{\lambda}{\theta}\right) = \frac{\lambda^2}{2} \exp(-\lambda).$$

Find the MAP solution for λ^*. □

8.2.2 MAP Learner

Now we come back to the question of estimating f_{Bayes} for which we need to estimate the conditional pdf $p_{Y|\boldsymbol{X}}$. Similar to MLE, we assume that the data comes from a Gaussian distribution with mean $\mathbf{x}^\top \mathbf{w}^*$ and variance 1. But, now we will use MAP to estimate the parameters $\mathbf{w}^* = (w_0^*, w_1^*, \ldots, w_d^*) \in \mathbb{R}^{d+1}$ of the distribution. To use MAP we also need to assume a prior distribution over the parameters. One common choice is to assume that all the parameters (except the bias term) $w_1^*, \ldots, w_d^*$ are i.i.d. with a Gaussian distribution with mean 0 and variance $1/\lambda$. This means that

$$p(w_j) = \frac{1}{\sqrt{2\pi/\lambda}} \exp\left(-\frac{w_j^2}{2/\lambda}\right).$$

for all $j \in \{1, \ldots, d\}$. This means that our prior belief is that the parameters are likely to be close to 0, which as we have seen in Section 7.5 implies the function $\mathbf{x}^\top \mathbf{w}^*$ is simple. The variable λ then allows us to control how much we believe in this prior. If λ is large, then we believe in the prior a lot, and if λ is small, then we do not believe in the prior much. Since the bias term w_0^* does not affect the complexity of the function $\mathbf{x}^\top \mathbf{w}^*$, we often do not consider having prior knowledge about it. One way to encode this is to assume that w_0^* is uniformly distributed on the interval $[-a, a]$ for a large a. This means that

$$p(w_0) = \frac{1}{2a}.$$

It is also assumed that w_0 is independent of the other parameters $w_1, \ldots, w_d$, so that the following math is simplified.

[1]Gamma distributin: https://en.wikipedia.org/wiki/Gamma_distribution

Now we can calculate the MAP solution:

$$\begin{aligned}
\mathbf{w}_{\text{MAP}} &= \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^{d+1}} p(\mathcal{D}|\mathbf{w}) \cdot p(\mathbf{w}) \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} -\log(p(\mathcal{D}|\mathbf{w}) \cdot p(\mathbf{w})) \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} [-\log(p(\mathcal{D}|\mathbf{w})) - \log(p(\mathbf{w}))] \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} \left[\sum_{i=1}^{n} \frac{(y_i - \mathbf{x}_i^\top \mathbf{w})^2}{2} - \log(p(\mathbf{w}))\right].
\end{aligned}$$

We can then expand and simplify the term $\log(p(\mathbf{w}))$ as follows:

$$\begin{aligned}
\log(p(\mathbf{w})) &= \log(p(w_0, w_1, \ldots, w_d)) \\
&= \log(p(w_0) \cdot p(w_1) \cdots p(w_d)) && \text{(independence assumption)} \\
&= \log(p(w_0)) + \log(p(w_1)) + \cdots + \log(p(w_d)) \\
&= \log(p(w_0)) + \sum_{j=1}^{d} \log(p(w_j)) \\
&= \log(p(w_0)) + \sum_{j=1}^{d} \log\left(\frac{1}{\sqrt{2\pi/\lambda}} \exp\left(-\frac{w_j^2}{2/\lambda}\right)\right) \\
&= \log(p(w_0)) + \sum_{j=1}^{d} \left[\log\left(\frac{1}{\sqrt{2\pi/\lambda}}\right) - \frac{\lambda w_j^2}{2}\right] \\
&= \log(p(w_0)) + d\log\left(\frac{1}{\sqrt{2\pi/\lambda}}\right) - \sum_{j=1}^{d} \frac{\lambda w_j^2}{2}.
\end{aligned}$$

We can then plug this back into the MAP solution to get

$$\begin{aligned}
\mathbf{w}_{\text{MAP}} &= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} \left[\sum_{i=1}^{n} \frac{(y_i - \mathbf{x}_i^\top \mathbf{w})^2}{2} - \log(p(w_0)) - d\log\left(\frac{1}{\sqrt{2\pi/\lambda}}\right) + \sum_{j=1}^{d} \frac{\lambda w_j^2}{2}\right] \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} \left[\sum_{i=1}^{n} \frac{(y_i - \mathbf{x}_i^\top \mathbf{w})^2}{2} + \sum_{j=1}^{d} \frac{\lambda w_j^2}{2}\right].
\end{aligned}$$

The last equality holds since the term $-\log(p(w_0)) - d\log\left(\frac{1}{\sqrt{2\pi/\lambda}}\right)$ does not depend on $\mathbf{w}$. Finally, recall the definition of the regularized estimated loss

$$\hat{L}_{\text{reg}}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^\top \mathbf{w}\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2.$$

Notice how this is almost the same function we are trying to minimize in the MAP solution. In particular, it holds that:

$$\begin{aligned}
\mathbf{w}_{\text{MAP}} &= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} \left[\frac{n}{2}\hat{L}_{\text{reg}}(\mathbf{w})\right] \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} \hat{L}_{\text{reg}}(\mathbf{w}).
\end{aligned}$$

Thus the MAP parameters $\mathbf{w}_{\text{MAP}}$ is the same as the regularized ERM parameters. If then want an estimate of f_{Bayes} we can plug in $\mathbf{w}_{\text{MAP}}$ into the conditional pdf $p_{Y|\boldsymbol{X}}$ to get:

$$\begin{aligned} f_{\text{Bayes}}(\mathbf{x}) &= \mathbb{E}[Y|\boldsymbol{X} = \mathbf{x}] \\ &= \int_{\mathbb{R}} y \cdot p_{Y|\boldsymbol{X}}(y|\mathbf{x})\, dy \\ &\approx \int_{\mathbb{R}} y \cdot p_{Y|\boldsymbol{X}}(y|\mathbf{x}, \mathbf{w}_{\text{MAP}})\, dy \\ &= \mathbb{E}[Y'|\boldsymbol{X} = \mathbf{x}] && (\text{where } Y'|\boldsymbol{X} = \mathbf{x} \sim \mathcal{N}(\mathbf{x}^\top \mathbf{w}_{\text{MAP}}, 1)) \\ &= \mathbf{x}^\top \mathbf{w}_{\text{MAP}}. \end{aligned}$$

Which is the same as the regularized ERM predictor we saw in Section 7.5. This shows that another interpertation of what regularization term λ is doing is that it is encoding our prior beliefs about the parameters $\mathbf{w}^*$.

Lasso Regression

In the previous section we assume that the parameters $w_1, \ldots, w_d$ are i.i.d. with a Gaussian distribution. Another common choice is to assume that the parameters $w_1, \ldots, w_d$ are i.i.d. with a Laplace distribution with mean 0 and scale parameter $1/\lambda$. It can be shown that the MAP solution is:

$$\mathbf{w}_{\text{MAP}} = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^{d+1}} \left[\sum_{i=1}^{n} \frac{(y_i - \mathbf{x}_i^\top \mathbf{w})^2}{2} + \sum_{j=1}^{d} \lambda |w_j| \right] = \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^{d+1}} \left[\frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^\top \mathbf{w})^2 + \frac{2}{n} \sum_{j=1}^{d} \lambda |w_j| \right].$$

Optimizing this objective is known as *Lasso regression*.

Chapter 9

Classification

So far we have only considered regression problems, where the labels have a notion of order. In classification problems, the labels are unordered and usually belong to a finite set.

Example 9.1: [Types on Wine] Consider the problem of classifying wines into three types: Barolo, Grignolino, and Barbera. The set of labels is $\mathcal{Y} = \{\text{Barolo}, \text{Grignolino}, \text{Barbera}\}$. For mathematical operations we will find it is often convenient to encode the labels as integers, for example $\{0, 1, 2\}$. The number 0 would correspond to Barolo, 1 to Grignolino, and 2 to Barbera. □

In Chapter 4 we discussed how for classification problems the loss function is often the zero-one loss, defined as

$$\ell(\hat{y}, y) = \begin{cases} 0 & \text{if } \hat{y} = y, \\ 1 & \text{otherwise.} \end{cases}$$

The reason for this is that the labels are unordered, so there is no notion of distance between them. Thus, if the prediction is correct, the loss is zero, and if it is incorrect, the loss is one. If we use the set of labels $\mathcal{Y} = \{0, 1, 2\}$ (as in Example 9.1), then for a fixed true label y we can plot the loss as a function of the predicted label $\hat{y}$, as shown in Fig. 9.1. An important

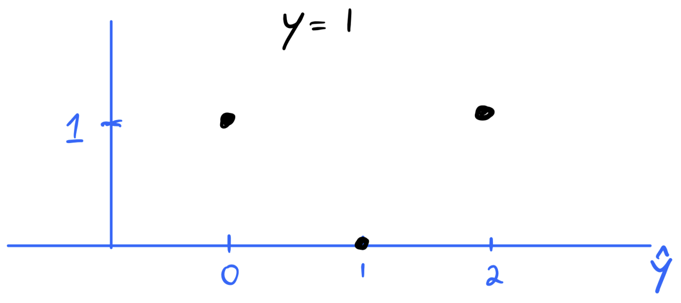


Figure 9.1: Zero-one loss for a fixed true label $y = 1$.

observation is that the zero-one loss is not a continuous function of the predicted label $\hat{y}$.

Suppose we try to use an ERM learner (Definition 1) to solve a classification problem. We would first estimate the expected loss

$$\hat{L}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i).$$

Then, we would need to minimize this estimated loss over the set of functions $\mathcal{F}$. Even if we define $\mathcal{F}$ to only contain functions that are continuous and described by some parameter $\mathbf{w}$

(as in linear regression), we will find that still the estimated loss $\hat{L}(f)$ is not continuous in $\mathbf{w}$, since the zero-one loss is not continuous. This means we can no longer use the optimization techniques we learned in Chapter 6 to minimize the estimated loss. More generally, if a function is not continuous, then it is often difficult to optimize it. To avoid this issue, we can try a different learner from ERM. In particular, we will try a similar approach as in Chapter 8.

Recall that the best possible predictor is the Bayes predictor

$$f_{\text{Bayes}} = \operatorname*{argmin}_{f \in \{f | f : \mathcal{X} \to \mathcal{Y}\}} L(f) \quad \text{where} \quad L(f) = \mathbb{E}[\ell(f(\boldsymbol{X}), Y)].$$

It can be shown that the following holds if the loss function is the zero-one loss:

$$f_{\text{Bayes}}(\mathbf{x}) = \operatorname*{argmax}_{y \in \mathcal{Y}} p(y|\mathbf{x}).$$

Importantly, if we know the pmf $p(y|\mathbf{x})$, then we can compute the Bayes predictor. We have already studied two methods (MLE and MAP) to do this in Chapter 8. In this chapter we will focus on using the MLE method, however the MAP method can be used in the same way.

Our goal will be to consider the case when the set of labels $\mathcal{Y}$ is finite and contains K elements. However, to build intuition, we will first consider the case when there are only two labels $\mathcal{Y} = \{0, 1\}$.

9.1 Binary Classification

In binary classification, the set of labels is $\mathcal{Y} = \{0, 1\}$. To get an estimate of f_{Bayes}, we need to estimate the pmf $p(y|\mathbf{x})$. We will do this using MLE. We are working in the supervised learning setting, so we have a dataset $D = ((\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n))$. The feature-label pairs $(\boldsymbol{X}_i, Y_i)$ are i.i.d. samples from the joint distribution $\mathbb{P}_{\boldsymbol{X},Y}$. Since the labels can only take two values, the distribution of Y given $\boldsymbol{X} = \mathbf{x}$ can be described by a Bernoulli distribution with some true parameter $\alpha^*(\mathbf{x}) \in [0, 1]$. This implies that the pmf $p(y|\mathbf{x})$ is given by

$$p(y|\mathbf{x}) = \alpha^*(\mathbf{x})^y (1 - \alpha^*(\mathbf{x}))^{1-y}.$$

Importantly, the pmf $p(y|\mathbf{x})$ is entirely determined by the parameter $\alpha^*(\mathbf{x})$. Thus, estimating $p(y|\mathbf{x})$ is equivalent to estimating $\alpha^*(\mathbf{x})$. We need to make an assumption of what the function $\alpha^*(\mathbf{x})$ is. So far in these notes we have used linear or polynomial functions, which can take any value in $\mathbb{R}$. However, in this case $\alpha^*(\mathbf{x})$ is a probability, so it must be in the range $[0, 1]$. We will use the *logistic* (or *sigmoid*) function to ensure this. The logistic function is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

The logistic function maps any real number to the range $[0, 1]$, as shown in Fig. 9.2. Thus, we will assume that

$$p(y|\mathbf{x}) = p(y|\mathbf{x}, \mathbf{w}^*) = \sigma(\mathbf{x}^\top \mathbf{w}^*)^y (1 - \sigma(\mathbf{x}^\top \mathbf{w}^*))^{1-y},$$

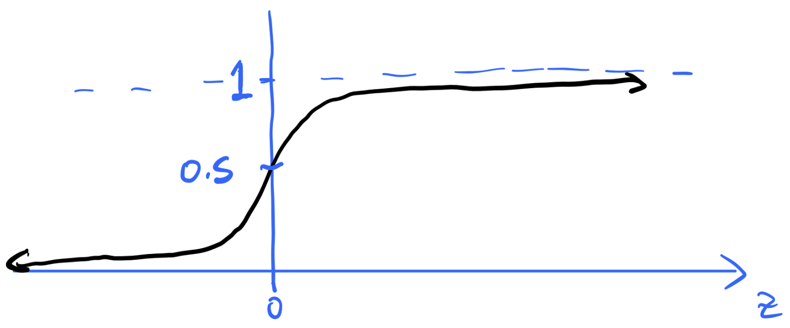


Figure 9.2: The logistic function.

where $\mathbf{w}^* \in \mathbb{R}^{d+1}$ is the true weight vector.

Next, we procede to estimate the parameter $\mathbf{w}^*$ using MLE. Let $\mathcal{D} = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n))$ be a fixed dataset. Then

$$\begin{aligned}
\mathbf{w}_{\text{MLE}} &= \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^{d+1}} p(\mathcal{D}|\mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^{d+1}} \prod_{i=1}^{n} p(\mathbf{x}_i, y_i|\mathbf{w}) \\
&= \operatorname*{argmax}_{\mathbf{w}\in\mathbb{R}^{d+1}} \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) p(\mathbf{x}_i) \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} -\sum_{i=1}^{n} [\log(p(y_i|\mathbf{x}_i, \mathbf{w})) + \log(p(\mathbf{x}_i))] \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} -\sum_{i=1}^{n} \log(p(y_i|\mathbf{x}_i, \mathbf{w})).
\end{aligned}$$

The above steps should all be familiar from Chapter 8. We can now plug in the definition of $p(y_i|\mathbf{x}_i, \mathbf{w})$ and use the logarithm properties that $\log(ab) = \log(a) + \log(b)$ and $\log(a^b) = b\log(a)$ to get

$$\begin{aligned}
\mathbf{w}_{\text{MLE}} &= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} -\sum_{i=1}^{n} \log\left((\sigma(\mathbf{x}_i^\top \mathbf{w}))^{y_i} (1 - \sigma(\mathbf{x}_i^\top \mathbf{w}))^{1-y_i} \right) \\
&= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^{d+1}} \underbrace{-\sum_{i=1}^{n} \left[y_i \log(\sigma(\mathbf{x}_i^\top \mathbf{w})) + (1 - y_i) \log(1 - \sigma(\mathbf{x}_i^\top \mathbf{w})) \right]}_{g(\mathbf{w})}.
\end{aligned}$$

It can be shown that the function $g(\mathbf{w})$ is convex, so we can use the optimization techniques from Chapter 6 to minimize it. Unfortunately, there is no closed form solution for the minimizer of $g(\mathbf{w})$. However, we can use gradient descent to find an approximate solution. We need to compute the gradient of $g(\mathbf{w})$. To do this, we will work out the partial derivative of $g(\mathbf{w})$ with respect to the j-th component of $\mathbf{w}$. First we define some terms:

$$u_i = \mathbf{x}_i^\top \mathbf{w}, \quad v_i = \sigma(u_i), \quad h_i = y_i \log(v_i), \quad r_i = (1 - y_i) \log(1 - v_i).$$

Then we have

$$\frac{\partial g(\mathbf{w})}{\partial w_j} = -\sum_{i=1}^{n}\left[\frac{dh_i}{dv_i}\frac{dv_i}{du_i}\frac{\partial u_i}{\partial w_j} + \frac{dr_i}{dv_i}\frac{dv_i}{du_i}\frac{\partial u_i}{\partial w_j}\right]$$

We can work out each of the derivatives separately:

$$\frac{dh_i}{dv_i} = \frac{y_i}{v_i} = \frac{y_i}{\sigma(u_i)}, \quad \frac{dr_i}{dv_i} = -\frac{1-y_i}{1-v_i} = -\frac{1-y_i}{1-\sigma(u_i)}, \quad \frac{dv_i}{du_i} = \sigma(u_i)(1-\sigma(u_i)), \quad \frac{\partial u_i}{\partial w_j} = x_{ij}.$$

The derivative $\frac{dv_i}{du_i}$ takes a bit of work to compute and is left as an exercise for the reader. Plugging these into the expression for $\frac{\partial g(\mathbf{w})}{\partial w_j}$ we get

$$\begin{aligned}
\frac{\partial g(\mathbf{w})}{\partial w_j} &= -\sum_{i=1}^{n}\left[\frac{y_i}{\sigma(u_i)}\sigma(u_i)(1-\sigma(u_i))x_{ij} - \frac{1-y_i}{1-\sigma(u_i)}\sigma(u_i)(1-\sigma(u_i))x_{ij}\right] \\
&= -\sum_{i=1}^{n}\left[y_i(1-\sigma(u_i))x_{ij} - (1-y_i)\sigma(u_i)x_{ij}\right] \\
&= -\sum_{i=1}^{n}\left[y_i x_{ij} - y_i\sigma(u_i)x_{ij} - \sigma(u_i)x_{ij} + y_i\sigma(u_i)x_{ij}\right] \\
&= \sum_{i=1}^{n}\left[\sigma(u_i) - y_i\right]x_{ij}.
\end{aligned}$$

One interesting observation is that this gradient is very similar to the gradient of the loss function in linear regression, if the predictor is given by $\sigma(\mathbf{x}^\top\mathbf{w})$[1]. The gradient of $g(\mathbf{w})$ is then

$$\nabla g(\mathbf{w}) = \sum_{i=1}^{n}\left[\sigma(\mathbf{x}_i^\top\mathbf{w}) - y_i\right]\mathbf{x}_i,$$

and the update rule for gradient descent is

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t+1)}\nabla g(\mathbf{w}^{(t)}).$$

If we run gradient descent with a good step size $\eta^{(t)}$, for enough epochs T, then we should expect $\mathbf{w}^{(T)}$ to be approximately equal to $\mathbf{w}_{\text{MLE}}$. If we have $\mathbf{w}_{\text{MLE}}$ (or an approximation of it), then we have an estimate of $\mathbf{w}^*$. This implies that $f_{\text{MLE}}(\mathbf{x}) = \sigma(\mathbf{x}^\top\mathbf{w}_{\text{MLE}})$ is approximately equal to $\sigma(\mathbf{x}^\top\mathbf{w}^*) = \alpha^*(\mathbf{x})$. Thus, $p(y|\mathbf{x}, \mathbf{w}_{\text{MLE}}) = \sigma(\mathbf{x}^\top\mathbf{w}_{\text{MLE}})^y(1-\sigma(\mathbf{x}^\top\mathbf{w}_{\text{MLE}}))^{1-y}$ is an estimate of $p(y|\mathbf{x})$.

We can now use $p(y|\mathbf{x}, \mathbf{w}_{\text{MLE}})$ to get an estimate of f_{Bayes}.

$$\begin{aligned}
f_{\text{Bayes}}(\mathbf{x}) &= \operatorname*{argmax}_{y\in\mathcal{Y}} p(y|\mathbf{x}) \\
&\approx \operatorname*{argmax}_{y\in\{0,1\}} p(y|\mathbf{x}, \mathbf{w}_{\text{MLE}}) \\
&= \hat{f}_{\text{Bin}}(\mathbf{x}).
\end{aligned}$$

[1]This turns out to not be a coincidence. This example and the one covered in Section 8.1.2 are both examples of generalized linear models.

In particular, we can use the above function $\hat{f}_{\text{Bin}}$ to define a learner for the binary classification problem as

$$\mathcal{A}(\mathcal{D}) = \hat{f}_{\text{Bin}}(\mathbf{x}).$$

To build some intuition for what $\hat{f}$ is doing we can write out its defintion in conditional form:

$$\hat{f}_{\text{Bin}}(\mathbf{x}) = \begin{cases} 1 & \text{if } p(y=1|\mathbf{x}, \mathbf{w}_{\text{MLE}}) \geq p(y=0|\mathbf{x}, \mathbf{w}_{\text{MLE}}), \\ 0 & \text{if} \end{cases}$$

But since $p(y=1|\mathbf{x}, \mathbf{w}_{\text{MLE}}) = \sigma(\mathbf{x}^\top \mathbf{w}_{\text{MLE}})$ and $p(y=0|\mathbf{x}, \mathbf{w}_{\text{MLE}}) = 1 - \sigma(\mathbf{x}^\top \mathbf{w}_{\text{MLE}})$, we can simplify the above to

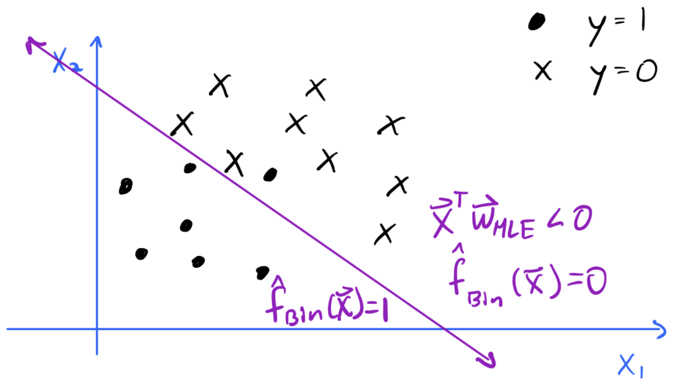
$$\begin{aligned} \hat{f}_{\text{Bin}}(\mathbf{x}) &= \begin{cases} 1 & \text{if } \sigma(\mathbf{x}^\top \mathbf{w}_{\text{MLE}}) \geq 0.5, \\ 0 & \text{if } \sigma(\mathbf{x}^\top \mathbf{w}_{\text{MLE}}) < 0.5 \end{cases} \\ &= \begin{cases} 1 & \text{if } \mathbf{x}^\top \mathbf{w}_{\text{MLE}} \geq 0, \\ 0 & \text{if } \mathbf{x}^\top \mathbf{w}_{\text{MLE}} < 0. \end{cases} \end{aligned}$$

The last equality is true because the logistic function is increasing and crosses 0.5 at 0. The reason the last expression is useful is that it shows that the output of f_{Bin} changes from 0 to 1 at the point $\mathbf{x}^\top \mathbf{w}_{\text{MLE}} = 0$. The equation $\mathbf{x}^\top \mathbf{w}_{\text{MLE}} = 0$ defines a hyperplane in $\mathbb{R}^{d-1}$ and is called a *decision boundary.*

Example 9.2: Let $d = 2$. Then the decision boundary $\mathbf{x}^\top \mathbf{w}_{\text{MLE}} = 0$ is a line in $\mathbb{R}^2$. To see this notice that

$$\mathbf{x}^\top \mathbf{w}_{\text{MLE}} = w_{\text{MLE},0} + w_{\text{MLE},1} x_1 + w_{\text{MLE},2} x_2 = 0 \implies x_2 = -\frac{w_{\text{MLE},0}}{w_{\text{MLE},2}} - \frac{w_{\text{MLE},1}}{w_{\text{MLE},2}} x_1.$$

We can plot this decision boundary in the feature space, as shown in Fig. 9.3. The symbol



$y=1$
$y=0$
X_2
X_1
$\vec{X}^T \vec{w}_{HLE} < 0$
$\hat{f}_{Bin}(\vec{X})=0$
$\hat{f}_{Bin}(\vec{X})=1$

Figure 9.3: The decision boundary in the feature space.

"X" represents the data points that have a label of 0, and the small black circles represent the data points that have a label of 1. The decision boundary is the line that best separates the two classes. □

9.1.1 Logistic Regression

You might have noticed that the binary classification learner we have defined above can be thought of as a two step process. First, estimate the probability $p(y|\mathbf{x})$, then use this probability to predict the label. An important observation is that in the binary classification case, since $\mathcal{Y} = \{0, 1\}$, to estimate $p(y|\mathbf{x})$ we just need to estimate $p(y = 1|\mathbf{x})$ and $p(y = 0|\mathbf{x})$. Further, since $p(y = 0|\mathbf{x}) = 1 - p(y = 1|\mathbf{x})$, we actually only need to estimate $p(y = 1|\mathbf{x}) = \alpha^*(\mathbf{x})$. It turns out that we can get the same estimate of $p(y = 1|\mathbf{x})$ as we saw above (i.e. f_{MLE}) by thinking of the problem as a regression problem and using a function class containing logistic functions. This is called *logistic regression*.

Suppose we have the same setup as in the binary classification problem. That is, we have a dataset $D = ((\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n))$ where the labels are binary. Although we know that $Y_i \in \{0, 1\}$, we can define a new (larger) set of labels as $\mathcal{Y}_{\text{Log}} = [0, 1]$, which is the closed interval from 0 to 1. We will think of the labels Y_i as elements of $\mathcal{Y}_{\text{Log}}$, and representing the probability of the label being 1. For the loss function we will use the *binary cross-entropy loss*, defined as

$$\ell(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}).$$

The function class will contain the logistic functions, defined as

$$\mathcal{F} = \left\{ f | f : \mathbb{R}^{d+1} \to [0, 1], \text{ where } f(\mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w}), \text{ and } \mathbf{w} \in \mathbb{R}^{d+1} \right\}.$$

The ERM learner for this problem is

$$\mathcal{A}(\mathcal{D}) = \hat{f}_{\text{ERM}} \quad \text{where} \quad \hat{f}_{\text{ERM}} = \operatorname*{argmin}_{f \in \mathcal{F}} \hat{L}(f).$$

If we solve for $\hat{f}_{\text{ERM}}$ we get that

$$\begin{aligned}
\hat{f}_{\text{ERM}} &= \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i) \\
&= \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} [-y_i \log(f(\mathbf{x}_i)) - (1 - y_i) \log(1 - f(\mathbf{x}_i))] \\
&= \operatorname*{argmin}_{f \in \mathcal{F}} - \sum_{i=1}^{n} [y_i \log(f(\mathbf{x}_i)) + (1 - y_i) \log(1 - f(\mathbf{x}_i))] \\
&= f_{\text{MLE}}.
\end{aligned}$$

The second last equality holds since scaling a function by a constant $(1/n)$ does not change the minimizer. The last equality holds by noticing that the function being minimized is exactly the function $g(\mathbf{w})$ defined above. This result shows that the MLE function f_{MLE} can be thought of as the solution to the logistic regression problem, where the loss function is the cross-entropy loss. In the other direction, $\hat{f}_{\text{ERM}}$ can be understood as predicting $p(y = 1|\mathbf{x}) = \alpha^*(\mathbf{x})$, since that is what f_{MLE} does.

Exercise 9.1: Suppose that you try to solve for $\hat{f}_{\text{ERM}}$ using gradient descent with 200 epochs. You save the predictors at the end of each epoch and call them $\hat{f}^{(1)}, \ldots, \hat{f}^{(200)}$. You plot the estimated loss $\hat{L}(\hat{f}^{(T)})$ (with the binary cross-entropy loss function) as a

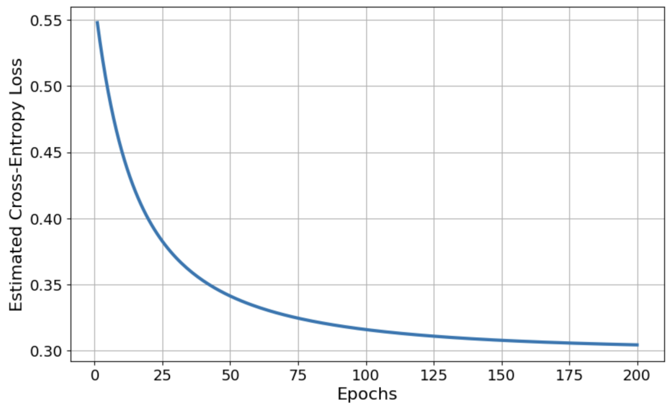


Figure 9.4: Estimated loss as a function of the epoch number.

function of the epoch number T and get the plot shown in Fig. 9.4. You can get a binary classification predictor $f_{\text{bin}}^{(T)}$ by thresholding the output of $\hat{f}^{(T)}$ at 0.5. That is, $f_{\text{bin}}^{(T)}(\mathbf{x}) = 1$ if $\hat{f}^{(T)}(\mathbf{x}) \geq 0.5$ and 0 otherwise. Let $f^* = \operatorname{argmin}_{f \in \mathcal{F}} L(f)$, and $f_{\text{Bayes}} = \operatorname{argmin}_{f \in \{f : f : \mathcal{X} \to \mathcal{Y}\}} L(f)$, where the loss function is the zero-one loss. What is the relationship between $\hat{L}(f_{\text{bin}}^{(T)}), L(f_{\text{bin}}^{(T)}), L(f^*), L(f_{\text{Bayes}})$, where the loss function is the zero-one loss?

□

Polynomial Features and Regularization

In the above discussion we assumed that a linear function of the features passed as input to the logistic function was sufficient to model the data. However, f_{Bayes} could be more complex than this. As we have seen before, one way to address this is to transform the features using the polynomial function ϕ_p. If we do this then the function class will become more complex, and we will be able to model more complex relationships between the features and the labels (lower approximation error). However, increasing the complexity too much can lead to overfitting (higher estimation error). The ideas discussed in Chapter 7 can then be used to select the best polynomial degree p.

An alternative to selecting the polynomial degree is to use *regularization*. In this case we would select a large polynomial degree and then use then try different values of the regularization parameter λ to find the best predictor. Again the same ideas from Chapter 7 can be used to select the best λ. Similarly, the MAP interpretation of regularization can be worked out for the binary classification problem.

Exercise 9.2: Let $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$ be a dataset and $d = 2$. Suppose that for $p = 1$ and $p = 5$ you are given $f_{\text{MLE, p}}(\mathbf{x}) = \sigma(\phi_p(\mathbf{x})^\top \mathbf{w}_{\text{MLE, p}})$, which is the MLE solution based on $\mathcal{D}$ when assuming that $\alpha^*(\mathbf{x}) = \sigma(\phi_p(\mathbf{x})^\top \mathbf{w}^*)$. You define the binary classification predictor $f_{\text{Bin, p}}(\mathbf{x})$ as 1 if $f_{\text{MLE, p}}(\mathbf{x}) \geq 0.5$ and 0 otherwise. The decision boundary of $f_{\text{Bin, p}}$ is the set of points $\mathbf{x}$ such that $f_{\text{MLE, p}}(\mathbf{x}) = 0.5$. You plot the dataset and the decision boundary for $p = 1$ and $p = 5$ and get the plots shown in Fig. 9.5. What are the answers to the following questions?

1. Is the decision boundary for $p = 1$ the line $\mathbf{x}^\top \mathbf{w}_{\text{MLE, 1}} = 0$, and is the decision

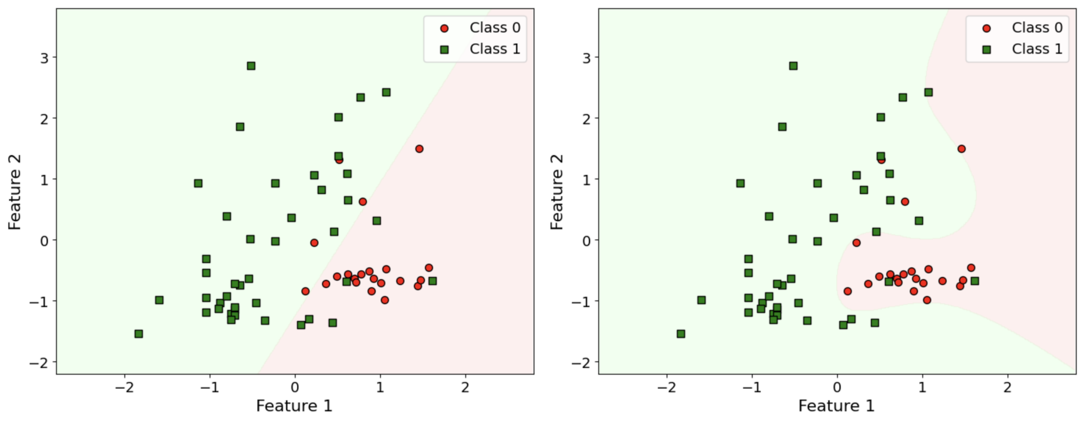


Figure 9.5: Decision boundary for $p = 1$ and $p = 5$.

boundary for $p = 5$ the curve $phi(\mathbf{x})^\top \mathbf{w}_{\text{MLE, }5} = 0$?

2. Is the left or right plot for $p = 1$?
3. What does the green and red region represent in each plot?
4. If you changed the threshold for the binary classification predictor from 0.5 to 0.1, how would the decision boundary change in each plot?
5. Which of the two predictors $f_{\text{Bin, }1}$ and $f_{\text{Bin, }5}$ has a lower estimated loss on the dataset $\mathcal{D}$ with the zero-one loss function?
6. Is $p = 5$ or $p = 1$ more likely to overfit the dataset $\mathcal{D}$?

$\square$

Logistic Regression vs. Linear Regression for Predicting Probabilities

When we think of binary classification as the two step process discussed above, it may not be entirely clear that using the logistic function class and the cross-entropy loss is the best way to estimate $p(y = 1|\mathbf{x})$. For instance, we could use the linear function class and the squared loss to estimate $p(y = 1|\mathbf{x})$. It turns out that this idea runs into some issues, when the value of $\mathbf{x}$ varies much more for one class than the other. We will not go into the mathematical details here, but we give a visual example to build some intuition.

Example 9.3: Let $d = 1$. We can plot the dataset, where the x-axis represents the single feature x_1 and the y-axis represents the label, shown in Fig. 9.6. The logistic regression learner would output a predictor defined by the logistic function, shown as the purple curve. Then, the binary classification learner would output a predictor which outputs 1 if the logistic function is greater than 0.5 and 0 otherwise, shown as the purple dotted line. The linear regression learner would output a predictor defined by a linear function, shown as the green line. Then, the binary classification learner would output a predictor which outputs 1 if the linear function is greater than 0.5 and 0 otherwise, also shown as the purple dotted line. In this plot we have made the dataset such that features have similar variance

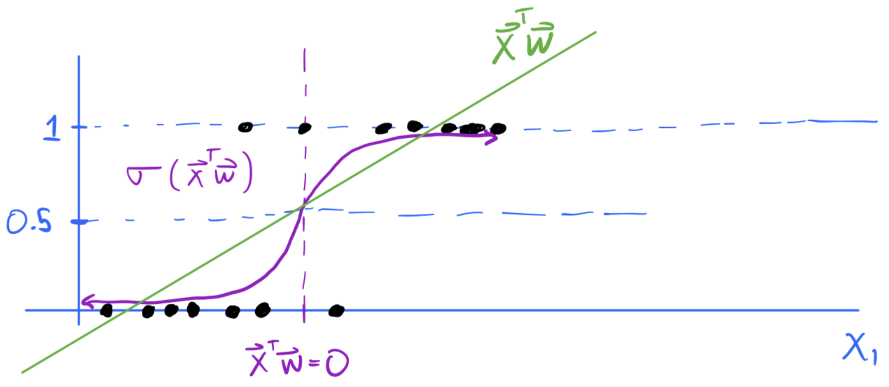


Figure 9.6: Linear vs. logistic regression for features with similar variance for both classes.

for the two classes. In this case, the logistic regression learner and linear regression learner would perform similarly.

Suppose that we have the same dataset, but with one extra datapoint that has a label of 1 but a very large value of x_1. This is shown as the red point in Fig. 9.7. This new datapoint

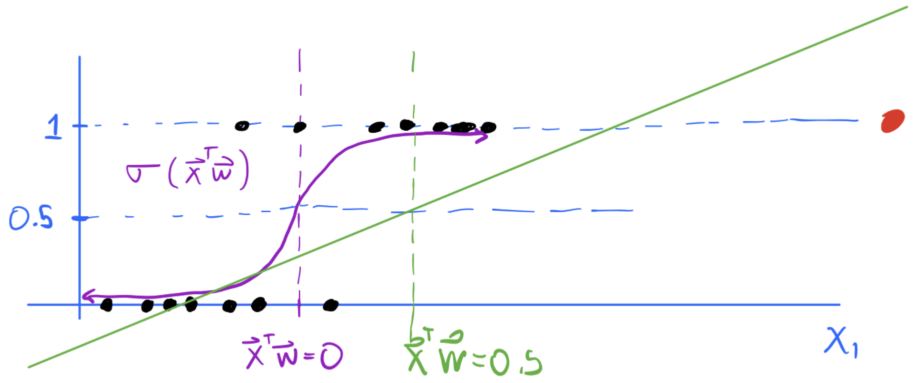


Figure 9.7: Linear vs. logistic regression for features with differing variance for each class.

would barely change the logistic regression predictor, since the logistic function (in purple) is already close to 1 for large values of x_1. However, the linear regression predictor (in green) would be significantly affected by this new datapoint. Notice how if you were to extrapolate the green line in Fig. 9.6 to the right you would get a value much greater than 1 when you reach the red point. Since the linear regression learner is using the squared loss, it would experience a large loss for this new datapoint. As such, the linear regression learner would try to move the green line to the right to reduce the loss. The new linear regression predictor is shown as the green line in Fig. 9.7. Notice how the decision boundary, when the linear function is equal to 0.5, has moved to the right, shown as the green dotted line. Importantly, what you should notice is that since the decision boundary has moved to the right, the linear regression learner would now predict 0 for more datapoints, many of which have a label of 1. □

9.2 Multiclass Classification

In the binary classification problem we had two labels, so the set of labels was $\mathcal{Y} = \{0, 1\}$. In the mutliclass classification problem we have K labels, so the set of labels is $\mathcal{Y} = \{0, 1, \dots, K-1\}$. Clearly the binary classification problem is a special case of the mutliclass classification problem, where $K = 2$. Our objective will again be to estimate the Bayes predictor f_{Bayes}, by estimating the pmf $p(y|\mathbf{x})$ using MLE.

The setting starts as before. We have a dataset $D = ((\boldsymbol{X}_1, Y_1), \dots, (\boldsymbol{X}_n, Y_n))$ where the feature-label pairs are i.i.d. samples from the joint distribution $\mathbb{P}_{\boldsymbol{X},Y}$. However, since the labels can take K values, the distribution of Y given $\boldsymbol{X} = \mathbf{x}$ can no longer be described by a Bernoulli distribution. Instead, we will use a *categorical* distribution, which is a generalization of the Bernoulli distribution to K values. The categorical distribution is defined by a vector of parameters $\boldsymbol{\alpha}^*(\mathbf{x}) = (\alpha_0^*(\mathbf{x}), \dots, \alpha_{K-1}^*(\mathbf{x})) \in [0,1]^K$. Each of the $\alpha_y^*(\mathbf{x})$ is the probability that the label is y given the feature $\mathbf{x}$. In particular, the pmf $p(y|\mathbf{x})$ is given by

$$p(y|\mathbf{x}) = \alpha_y^*(\mathbf{x}) \quad \text{for } y \in \mathcal{Y}.$$

Of course, since the parameters $\boldsymbol{\alpha}^*(\mathbf{x})$ are probabilities, they must be in the range $[0,1]$ and sum to 1. That is $\sum_{q=0}^{K-1} \alpha_q^*(\mathbf{x}) = 1$.

Now we need to make an assumption of what the function $\alpha_y^*(\mathbf{x})$ is for each $y \in \mathcal{Y}$. In the binary classification case we only needed to estimate a single parameter $\alpha^*(\mathbf{x}) \in [0,1]$, so it suficed to have one weight vector $\mathbf{w}^*$ that linearly mapped the features to a real number, and then to pass this through the logistic function to ensure it was in the range $[0,1]$. In the mutliclass classification case we have K parameters $\boldsymbol{\alpha}^*(\mathbf{x})$. One option is to assume that there are K weight vectors $\mathbf{w}_0^*, \dots, \mathbf{w}_{K-1}^* \in \mathbb{R}^{d+1}$, such that $\alpha_y^*(\mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w}_y^*)$, for each $y \in \mathcal{Y}$. However, this is not often done because once we estimate the K weight vectors, it is unlikely that the K estimates will sum to 1. Instead we will want to use a function other than the logistic function that takes as input all K linear functions of the features and outputs K probabilities that sum to 1. The function we will use is the *softmax* function, defined as

$$\sigma(\mathbf{z}) = (\sigma_0(\mathbf{z}), \dots, \sigma_{K-1}(\mathbf{z})) \quad \text{where} \quad \sigma_y(\mathbf{z}) = \frac{\exp(z_y)}{\sum_{q=0}^{K-1} \exp(z_q)} \in [0,1], \quad \text{for } y \in \mathcal{Y}.$$

We have overloaded the σ function here, since σ was previously used to denote the logistic function. However, it should be clear which function is being referred to based on the input, since for the softmax function takes input a vector $\mathbf{z} \in \mathbb{R}^K$ and the logistic function takes input a scalar $z \in \mathbb{R}$. Notice how the softmax function ensures that the K probabilities sum to 1 since

$$\sum_{y=0}^{K-1} \sigma_y(\mathbf{z}) = \sum_{y=0}^{K-1} \frac{\exp(z_y)}{\sum_{q=0}^{K-1} \exp(z_q)} = \frac{\sum_{y=0}^{K-1} \exp(z_y)}{\sum_{q=0}^{K-1} \exp(z_q)} = 1.$$

In conclusion, we will assume that

$$p(y|\mathbf{x}) = p(y|\mathbf{x}, \mathbf{w}_0^*, \dots, \mathbf{w}_{K-1}^*) = \sigma_y(\mathbf{x}^\top \mathbf{w}_0^*, \dots, \mathbf{x}^\top \mathbf{w}_{K-1}^*) \quad \text{where } \mathbf{w}_y^* \in \mathbb{R}^{d+1} \text{ for } y \in \mathcal{Y}.$$

We can now estimate the parameters $\mathbf{w}_0^*, \ldots, \mathbf{w}_{K-1}^*$ using MLE. Let $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$ be a fixed dataset. Then

$$\begin{aligned}
\mathbf{w}_{\mathrm{MLE},0}, \ldots, \mathbf{w}_{\mathrm{MLE},K-1} &= \underset{\mathbf{w}_0,\ldots,\mathbf{w}_{K-1} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} -\sum_{i=1}^{n} \log(p(y_i|\mathbf{x}_i, \mathbf{w}_0, \ldots, \mathbf{w}_{K-1})) \qquad (9.1) \\
&= \underset{\mathbf{w}_0,\ldots,\mathbf{w}_{K-1} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} -\sum_{i=1}^{n} \log\left(\frac{\exp(\mathbf{x}_i^\top \mathbf{w}_{y_i})}{\sum_{q=0}^{K-1} \exp(\mathbf{x}_i^\top \mathbf{w}_q)}\right) \\
&= \underset{\mathbf{w}_0,\ldots,\mathbf{w}_{K-1} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \underbrace{-\sum_{i=1}^{n} \left[\mathbf{x}_i^\top \mathbf{w}_{y_i} - \log\left(\sum_{q=0}^{K-1} \exp(\mathbf{x}_i^\top \mathbf{w}_q)\right)\right]}_{g(\mathbf{w}_0,\ldots,\mathbf{w}_{K-1})}.
\end{aligned}$$

The last equality used the logarithm property $\log(a/b) = \log(a) - \log(b)$. The function $g(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1})$ is convex, so we can use the optimization techniques from Chapter 6 to minimize it. Similar to the binary classification case, there is no closed form solution for the minimizer of $g(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1})$. As such, we wil need to use gradient descent to find an approximate solution. To calculate the gradient of $g(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1})$ we will need to work out the partial derivative of $g(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1})$ with respect to the j-th component of $\mathbf{w}_y$ (written as w_{yj}). Before doing so we will define some terms:

$$u_{iq} = \mathbf{x}_i^\top \mathbf{w}_q, \quad v_{iq} = \exp(u_{iq}), \quad r_i = \sum_{q=0}^{K-1} v_{iq}, \quad h_i = \log(r_i).$$

Then, we have

$$\frac{\partial g}{\partial w_{yj}}(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1}) = -\sum_{i=1}^{n} \left[\frac{\partial u_{iy_i}}{\partial w_{yj}} - \frac{dh_i}{dr_i} \sum_{q=0}^{K-1} \frac{dv_{iq}}{du_{iq}} \frac{\partial u_{iq}}{\partial w_{yj}}\right]. \qquad (9.2)$$

To work out each of the derivatives it will be helpful to introduce the indicator function $\mathbb{I}_{\mathcal{Z}}(z)$, which is defined as

$$\mathbb{I}_{\mathcal{Z}}(z) = \begin{cases} 1 & \text{if } z \in \mathcal{Z}, \\ 0 & \text{otherwise.} \end{cases}$$

Now we can work out the derivatives in Eq. (9.2).

$$\frac{dh_i}{dr_i} = \frac{1}{r_i}, \quad \frac{dv_{iq}}{du_{iq}} = \exp(u_{iq}), \quad \frac{\partial u_{iq}}{\partial w_{yj}} = \mathbb{I}_{\{q\}}(y) x_{ij} \quad \frac{\partial u_{iy_i}}{\partial w_{yj}} = \mathbb{I}_{\{y_i\}}(y) x_{ij}.$$

Plugging these into Eq. (9.2) we get

$$\begin{aligned}
\frac{\partial g}{\partial w_{yj}}(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1}) &= -\sum_{i=1}^{n} \left[\mathbb{I}_{\{y_i\}}(y) x_{ij} - \frac{1}{r_i} \sum_{q=0}^{K-1} \exp(u_{iq}) \mathbb{I}_{\{q\}}(y) x_{ij}\right] \\
&= -\sum_{i=1}^{n} \left[\mathbb{I}_{\{y_i\}}(y) x_{ij} - \frac{\exp(u_{iy})}{r_i} x_{ij}\right] \\
&= -\sum_{i=1}^{n} \left[\mathbb{I}_{\{y_i\}}(y) x_{ij} - \frac{\exp(\mathbf{x}_i^\top \mathbf{w}_y)}{\sum_{q=0}^{K-1} \exp(\mathbf{x}_i^\top \mathbf{w}_q)} x_{ij}\right] \\
&= \sum_{i=1}^{n} \left[\sigma_y(\mathbf{x}_i^\top \mathbf{w}_0, \ldots, \mathbf{x}_i^\top \mathbf{w}_{K-1}) - \mathbb{I}_{\{y_i\}}(y)\right] x_{ij}.
\end{aligned}$$

For $y \in \mathcal{Y}$, the gradient of $g(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1})$ with respect to $\mathbf{w}_y$ is

$$\begin{aligned}\nabla_{\mathbf{w}_y} g(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1}) &= \left(\frac{\partial g}{\partial w_{y0}}(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1}), \ldots, \frac{\partial g}{\partial w_{yd}}(\mathbf{w}_0, \ldots, \mathbf{w}_{K-1})\right)^\top \\ &= \sum_{i=1}^n \left[\sigma_y(\mathbf{x}_i^\top \mathbf{w}_0, \ldots, \mathbf{x}_i^\top \mathbf{w}_{K-1}) - \mathbb{I}_{\{y_i\}}(y)\right] \mathbf{x}_i.\end{aligned}$$

The gradient descent update rule for class y is then

$$\mathbf{w}_y^{(t+1)} = \mathbf{w}_y^{(t)} - \eta^{(t)} \nabla_{\mathbf{w}_y} g(\mathbf{w}_0^{(t)}, \ldots, \mathbf{w}_{K-1}^{(t)}).$$

An important note is that for each iteration t we will need to update all K weight vectors. It would be incorrect to only perform gradient descent for T epochs on one weight vector and then move on to the next weight vector and repeat this process for all K weight vectors.

If we select a good step size, and run for enough epochs T, then we should expect $\mathbf{w}_0^{(T)}, \ldots, \mathbf{w}_{K-1}^{(T)}$ to be approximately equal to $\mathbf{w}_{\text{MLE},0}, \ldots, \mathbf{w}_{\text{MLE},K-1}$. Then, if we have $\mathbf{w}_{\text{MLE},0}, \ldots, \mathbf{w}_{\text{MLE},K-1}$ (or an estimate), we have an estimate of $\mathbf{w}_0^*, \ldots, \mathbf{w}_{K-1}^*$, and can use this to estimate f_{Bayes}.

$$\begin{aligned}f_{\text{Bayes}}(\mathbf{x}) &= \operatorname*{argmax}_{y \in \mathcal{Y}} p(y|\mathbf{x}) \\ &\approx \operatorname*{argmax}_{y \in \{0,1,\ldots,K-1\}} p(y|\mathbf{x}, \mathbf{w}_{\text{MLE},0}, \ldots, \mathbf{w}_{\text{MLE},K-1}) \\ &= \hat{f}_{\text{Mul}}(\mathbf{x}).\end{aligned}$$

This gives us a learner for the mutliclass classification problem as

$$\mathcal{A}(\mathcal{D}) = \hat{f}_{\text{Mul}}(\mathbf{x}).$$

Exercise 9.3: Let $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$ be a dataset with $d = 2$ and $K = 3$. To simplify notation let $\sigma_y = \sigma_y(\mathbf{x}^\top \mathbf{w}_{\text{MLE},0}, \mathbf{x}^\top \mathbf{w}_{\text{MLE},1}, \mathbf{x}^\top \mathbf{w}_{\text{MLE},2})$ The decision boundary of f_{Mul} for classes $y \neq k$ is the set of points $\mathbf{x}$ such that

$$\sigma_y = \sigma_k \quad \text{and} \quad \sigma_q \leq \sigma_k \quad \text{for } y \neq q \neq k.$$

You plot the dataset and all the decision boundaries and get the plot shown in Fig. 9.8. What are the answers to the following questions?

1. What is the condition that holds when all three decision boundaries intersect?
2. What is equation of the line representing the decision boundary for classes 0 and 1?
3. How might the decision boundaries change if we got the MLE solution by assuming that

   $$p(y|\mathbf{x}) = \sigma_y(\phi_p(\mathbf{x})^\top \mathbf{w}_0^*, \ldots, \phi_p(\mathbf{x})^\top \mathbf{w}_{K-1}^*) \quad \text{where } \mathbf{w}_y^* \in \mathbb{R}^{\bar{p}} \text{ for } y \in \mathcal{Y},$$

   for some $p > 2$?

□

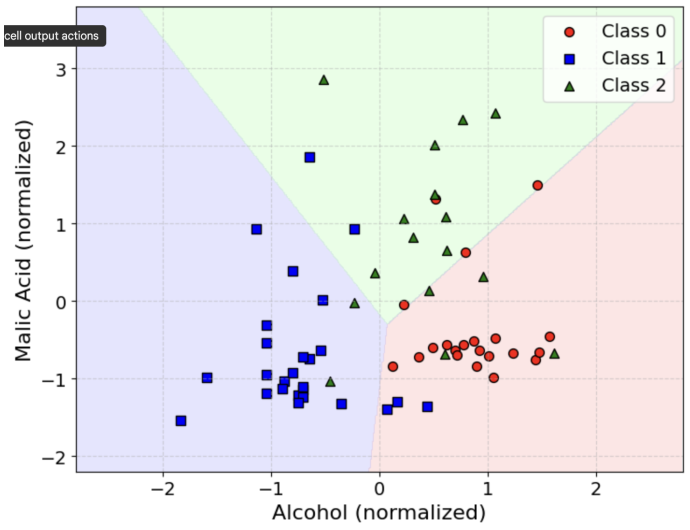


Figure 9.8: Decision boundaries for $K = 3$.

9.2.1 Softmax Regression

Similar to the binary classification case, we can think of the mutliclass classification learner as a two step process. First, estimate the probabilities $p(y|\mathbf{x})$, then use these probabilities to predict the label. Now, however, we have K probabilities to estimate. It again turns out that we can get the same estimate of $p(y|\mathbf{x}) = \alpha_y^*(\mathbf{x})$ as we saw above by thinking of the problem as a regression problem and using a function class containing softmax functions. This is called *softmax regression*.

In softmax regression we will have a new set of labels $\mathcal{Y}_{\text{Soft}}$ representing the probabilities of the classes $0, 1, \ldots, K-1$. Since there are K probabilities each label in $\mathcal{Y}_{\text{Soft}}$ will be a vector of length K, and thus $\mathcal{Y}_{\text{Soft}} = [0,1]^K$. For example if $K = 3$, then the label $(0.2, 0.8, 0)^\top$ would be interpreted as the probability of the class being 0 is 0.2, the probability of the class being 1 is 0.8, and the probability of the class being 2 is 0. Thus, it is necessary to convert the original labels $0, 1, \ldots, K-1$ to new labels in $\mathcal{Y}_{\text{Soft}}$ using a *one-hot encoding*. The one-hot encoding of a label $y \in \mathcal{Y}$ is a vector of length K where the y-th element is 1 and all other elements are 0. For example, if $K = 3$ then the one-hot encoding of an original label $y \in \mathcal{Y}$ would be

$$\begin{aligned} y = 0 &\to (1,0,0)^\top \in \mathcal{Y}_{\text{Soft}}, \\ y = 1 &\to (0,1,0)^\top \in \mathcal{Y}_{\text{Soft}}, \\ y = 2 &\to (0,0,1)^\top \in \mathcal{Y}_{\text{Soft}}. \end{aligned}$$

Thus, for softmax regression we will use a modified dataset $\mathcal{D}_{\text{Soft}} = ((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))$ where each of the labels $\mathbf{y}_i$ are one-hot encoded. The loss function will be the *multiclass cross-entropy loss*, defined as

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{q=0}^{K-1} y_q \log(\hat{y}_q).$$

The function class will contain the softmax functions, defined as

$$\mathcal{F} = \left\{ f | f : \mathbb{R}^{d+1} \to [0,1]^K, \text{ where } f(\mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w}_0, \ldots, \mathbf{x}^\top \mathbf{w}_{K-1}), \text{ and } \mathbf{w}_y \in \mathbb{R}^{d+1} \text{ for } y \in \mathcal{Y} \right\}.$$

The ERM learner for this problem is

$$\mathcal{A}(\mathcal{D}) = \hat{f}_{\text{ERM}} \quad \text{where} \quad \hat{f}_{\text{ERM}} = \operatorname*{argmin}_{f \in \mathcal{F}} \hat{L}(f).$$

If we solve for $\hat{f}_{\text{ERM}}$ we get that

$$\begin{aligned}
\hat{f}_{\text{ERM}} &= \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i), \mathbf{y}_i) \\
&= \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} - \sum_{q=0}^{K-1} y_{iq} \log(f_q(\mathbf{x}_i)) \\
&= \operatorname*{argmin}_{f \in \mathcal{F}} - \sum_{i=1}^{n} \sum_{q=0}^{K-1} y_{iq} \log(f_q(\mathbf{x}_i)) \\
&= f_{\text{MLE}}.
\end{aligned}$$

The function f_{MLE} is defined as $f_{\text{MLE}}(\mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w}_{\text{MLE},0}, \ldots, \mathbf{x}^\top \mathbf{w}_{\text{MLE},K-1})$. The notation $f_q(\mathbf{x})$ is used to denote the q-th element of the vector $f(\mathbf{x})$. The second last equality holds since scaling a function by a constant $(1/n)$ does not change the minimizer. The last equality can be shown by carefully comparing to Eq. (9.1).

Similar to the logistic regression case, the above result shows an interesting connection between the MLE solution and the softmax regression problem. In particular, it shows that the ERM predictor $\hat{f}_{\text{ERM}}$ can be understood as predicting the vector of probabilities $(p(y = 0|\mathbf{x}), \ldots, p(y = K-1|\mathbf{x})) = \boldsymbol{\alpha}^*(\mathbf{x})$, since that is what f_{MLE} does.

9.2.2 Logistic Regression as a Special Case of Softmax Regression

One might suspect that logistic regression is a special case of softmax regression, since it considers the case where $K = 2$. However, the logistic function and the softmax function are different, so it is not immediately clear if this is the case. Next, we will show that logistic regression solution is the same as the softmax regression solution when $K = 2$.

To show this, it will be easier to work with the MLE setting for binary and mutliclass classification. This is without loss of generality, since in Sections 9.1.1 and 9.2.1 we showed the equivilence between the ERM and MLE solutions in the binary and multiclass classification respectively. Suppose we follow the same MLE steps as in the multiclass classification case, but with $K = 2$. The labels are $\mathcal{Y} = \{0, 1\}$. We assume that the distribution of Y given $\boldsymbol{X} = \mathbf{x}$ is a categorical distribution with parameters $\boldsymbol{\alpha}^*(\mathbf{x}) = (\alpha_0^*(\mathbf{x}), \alpha_1^*(\mathbf{x}))$. The pmf $p(y|\mathbf{x})$ is then given by

$$p(y = 1|\mathbf{x}) = \alpha_1^*(\mathbf{x}) \quad \text{and} \quad p(y = 0|\mathbf{x}) = \alpha_0^*(\mathbf{x}).$$

Notice, however, that $\alpha_0^*(\mathbf{x}) = 1 - \alpha_1^*(\mathbf{x})$, since $p(y = 0|\mathbf{x}) + p(y = 1|\mathbf{x}) = 1$. Thus, the parameter $\alpha_0^*(\mathbf{x})$ is redundant, and we can just use $\alpha_1^*(\mathbf{x})$[2]. This means that the distribution

[2]This is true in general for the categorical distribution, not just when $K = 2$. In particular only $K - 1$ parameters are needed to define the categorical distribution, since the K-th parameter can be calculated from the other $K - 1$ parameters. However, it is still common to use K parameters, since it makes the math simpler.

of Y given $\boldsymbol{X} = \mathbf{x}$ is a Bernoulli distribution with the parameter $\alpha^*(\mathbf{x}) = \alpha_1^*(\mathbf{x})$. Thus, the distribution assumption for the binary classification and the mutliclass classification is the same. If we can show that the MLE estimate of $\alpha_1^*(\mathbf{x})$ in the multinomial classification case is the same as the estimate of $\alpha^*(\mathbf{x})$ in the binary classification case then we are done. This is what we will show next.

In the multiclass case we assume that the function $\alpha_1^*(\mathbf{x})$ is given by $\sigma_1(\mathbf{x}^\top \mathbf{w}_0^*, \mathbf{x}^\top \mathbf{w}_1^*)$. For any two weight vectors $\mathbf{w}_0, \mathbf{w}_1 \in \mathbb{R}^{d+1}$ we can rexpress $\sigma_1(\mathbf{x}^\top \mathbf{w}_0, \mathbf{x}^\top \mathbf{w}_1)$ by using the exponential property that $\exp(a)\exp(b) = \exp(a+b)$ as follows

$$\begin{aligned}
\sigma_1(\mathbf{x}^\top \mathbf{w}_0, \mathbf{x}^\top \mathbf{w}_1) &= \frac{\exp(\mathbf{x}^\top \mathbf{w}_1)}{\exp(\mathbf{x}^\top \mathbf{w}_0) + \exp(\mathbf{x}^\top \mathbf{w}_1)} \\
&= \frac{\exp(\mathbf{x}^\top \mathbf{w}_1)}{\exp(\mathbf{x}^\top \mathbf{w}_0) + \exp(\mathbf{x}^\top \mathbf{w}_1)} \frac{\exp(-\mathbf{x}^\top \mathbf{w}_1)}{\exp(-\mathbf{x}^\top \mathbf{w}_1)} \\
&= \frac{\exp(\mathbf{x}^\top (\mathbf{w}_1 - \mathbf{w}_1))}{\exp(\mathbf{x}^\top (\mathbf{w}_0 - \mathbf{w}_1)) + \exp(\mathbf{x}^\top (\mathbf{w}_1 - \mathbf{w}_1))} \\
&= \frac{1}{\exp(-\mathbf{x}^\top (\mathbf{w}_1 - \mathbf{w}_0)) + 1} \\
&= \sigma(\mathbf{x}^\top (\mathbf{w})). \qquad (9.3)
\end{aligned}$$

The last equality holds by the definition of the logistic function and letting $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_0$.

In the multiclass case the MLE solution is

$$\begin{aligned}
\mathbf{w}_{\text{MLE},0}, \mathbf{w}_{\text{MLE},1} &= \underset{\mathbf{w}_0, \mathbf{w}_1 \in \mathbb{R}^{d+1}}{\operatorname{argmin}} -\sum_{i=1}^n \log(p(y_i|\mathbf{x}_i, \mathbf{w}_0, \mathbf{w}_1)) \\
&= \underset{\mathbf{w}_0, \mathbf{w}_1 \in \mathbb{R}^{d+1}}{\operatorname{argmin}} -\sum_{i=1}^n \log\left(\sigma_{y_i}(\mathbf{x}_i^\top \mathbf{w}_0, \mathbf{x}_i^\top \mathbf{w}_1)\right) \\
&= \underset{\mathbf{w}_0, \mathbf{w}_1 \in \mathbb{R}^{d+1}}{\operatorname{argmin}} -\sum_{i=1}^n \Big[y_i \log(\sigma_1(\mathbf{x}_i^\top \mathbf{w}_0, \mathbf{x}_i^\top \mathbf{w}_1)) + (1-y_i)\log(\sigma_0(\mathbf{x}_i^\top \mathbf{w}_0, \mathbf{x}_i^\top \mathbf{w}_1))\Big] \\
&= \underset{\mathbf{w}_0, \mathbf{w}_1 \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \underbrace{-\sum_{i=1}^n \Big[y_i \log(\sigma(\mathbf{x}_i^\top (\mathbf{w}_1 - \mathbf{w}_0))) + (1-y_i)\log(1 - \sigma(\mathbf{x}_i^\top (\mathbf{w}_1 - \mathbf{w}_0)))\Big]}_{g(\mathbf{w}_0, \mathbf{w}_1)}.
\end{aligned}$$

The second last equality holds by observing that when $y_i = 1$ the second term in the summand is 0 and when $y_i = 0$ the first term in the summand is 0. The last equality holds by the result we derived in Eq. (9.3). Notice that although $g(\mathbf{w}_0, \mathbf{w}_1)$ is a function of two vectors, it is equivilent to a function $h(\mathbf{w})$ of only one vector, where $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_0$, since the two vectors only appear as the difference $\mathbf{w}_1 - \mathbf{w}_0$ in g. This is an important observation, since in the binary classification case the MLE solution is

$$\mathbf{w}_{\text{MLE}} = \underset{\mathbf{w} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} -\sum_{i=1}^n \Big[y_i \log(\sigma(\mathbf{x}_i^\top \mathbf{w})) + (1-y_i)\log(1 - \sigma(\mathbf{x}_i^\top \mathbf{w}))\Big] = \underset{\mathbf{w} \in \mathbb{R}^{d+1}}{\operatorname{argmin}}\, h(\mathbf{w}).$$

Thus, $\mathbf{w}_{\text{MLE}} = \mathbf{w}_{\text{MLE},1} - \mathbf{w}_{\text{MLE},0}$.

Finally, by Eq. (9.3) we have that

$$\sigma(\mathbf{x}^\top \mathbf{w}_{\text{MLE}}) = \sigma_1(\mathbf{x}^\top \mathbf{w}_{\text{MLE},0}, \mathbf{x}^\top \mathbf{w}_{\text{MLE},1}).$$

Confirming that the MLE estimate of $\alpha^*(\mathbf{x})$ in the binary classification case and the MLE estimate of $\alpha_1^*(\mathbf{x})$ in the multiclass classification case are indeed the same.

Chapter 10

Neural Networks

In the previous chapters we have learned that depending on the set of labels we should use different function classes. For instance, if the labels can be any real number, we should use linear functions, while if the labels are in the interval $[0, 1]$ we should apply the logistic function on top of the linear functions. However, sometimes the relationship between the features and the label is not linear. To address this problem we introduced the polynomial feature map ϕ_p (see Section 6.6), which transforms the original feature vector $\mathbf{x}$, into a polynomial feature vector $\phi_p(\mathbf{x})$ of degree p. As the degree p increases, the function class using the polynomial feature map is able to represent more complex relationships between the features and the label. Since the feature map ϕ_p is a fixed function, some of the features in ϕ_p might not be relevant for the supervised learning problem at hand. For example, if $d = 2$ and the labels have a quadratic relationship only depending of the first feature x_1, then the polynomial feature map $\phi_2(\mathbf{x})$ would have the features x_2^2 and x_1x_2, which are both irrelevant. In this chapter we will introduce a new function, the *neural network* (NN), which unlike the functions using the polynomial feature map, can learn which features are relevant for the supervised learning problem at hand. More precisely, we will study a specific type of neural network called a *multilayer perceptron* (MLP), which we will simply refer to as a NN in these notes.

10.1 Neural Networks: A Special Case

A neural network is a function $f : \mathcal{X} \to \mathcal{Y}$. In this section we will study a specific (NN) to introduce the concept. In the next section we will introduce the general structure of a NN. It is often helpful to visually represent a NN. The specific NN we will discuss in this section is shown in Fig. 10.1. The input $\mathbf{x}$ is shown on the left, and the output $f(\mathbf{x}) = a^{(3)}$ is shown on the right. You should think of information as flowing from left to right. Next, we introduce some terminology and notation to describe the NN, shown in Fig. 10.2. Each of the circles in the figure is called a *neuron*, and the lines connecting the neurons are called *weights*. A column of neurons is called a *layer*. This NN has three layers, since the input layer is not counted. The value of a neuron is called its *activation* and is written as $a_j^{(b)}$, where the superscript b is the layer number and the subscript j is the neuron number. It is convenient to group all the activations in a layer into a vector, which we will denote as $\mathbf{a}^{(b)}$. An activation depends on all of the activations in the previous layer, and the weights connecting to it. For instance, the activation $a_1^{(1)}$ depends on the input $\mathbf{x} = \mathbf{a}^{(0)}$ and the weights connecting to it (shown as blue lines), which we will call $\mathbf{w}_1^{(1)}$. The output of the NN function $f(\mathbf{x})$ is the activation of the last layer, $a^{(3)}$.

Now, we will define how the activations are computed, which will give us the formal

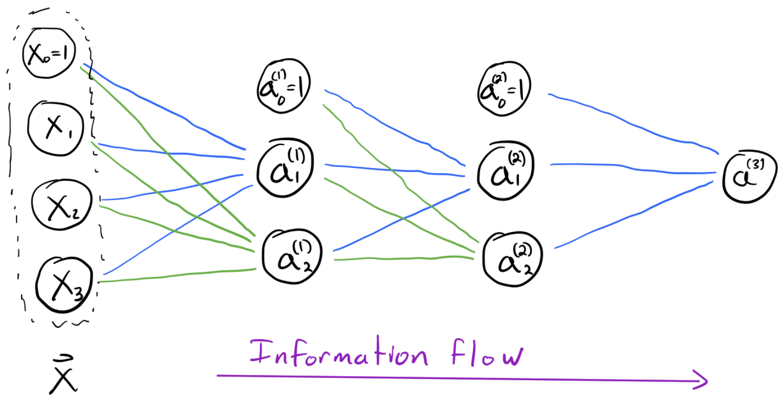


Figure 10.1: A specific neural network.

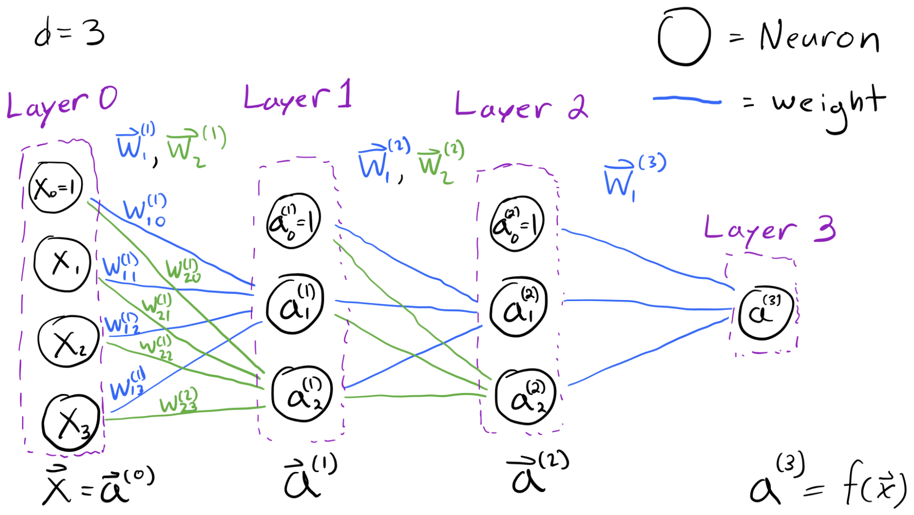


Figure 10.2: A specific neural network with annotations.

definition of the NN $f(\mathbf{x})$, since it is equal to the activation of the last layer. Lets begin with layer 1. The activation vector is

$$\mathbf{a}^{(1)} = (a_0^{(1)} = 1, a_1^{(1)}, a_2^{(1)})$$

Notice how the first activation $a_0^{(1)}$ is equal to 1, which represents the bias term. The reason for including the bias term is the same as in linear regression. Each activation in layer 1 is defined as

$$a_1^{(1)} = h^{(1)}(z_1^{(1)}),\ a_2^{(1)} = h^{(1)}(z_2^{(1)}) \quad \text{where} \quad z_1^{(1)} = \mathbf{x}^\top \mathbf{w}_1^{(1)},\ z_2^{(1)} = \mathbf{x}^\top \mathbf{w}_2^{(1)},$$

and $h^{(1)} : \mathbb{R} \to \mathbb{R}$ is an *activation function.* One example of an activation function which we have seen before is the logistic function, in which case you can think of the neuron as being *active* when the pre-activation z is positive (logistic function is approximately 1), and *inactive* when the pre-activation z is negative (logistic function is approximately 0).

The weight vectors $\mathbf{w}_1^{(1)}$ and $\mathbf{w}_2^{(1)}$ represent the blue and green lines respectively between the input layer and layer 1. The terms $z_1^{(1)}$ and $z_2^{(1)}$ are called *pre-activations*. Notice how indeed the activations only depend on the previous activation vector $\mathbf{a}^{(0)} = \mathbf{x}$ and the weights connecting to them. An activation cannot depend on the activations or weights of any other layer other than the previous one. The dimensionality of the terms we have defined so far are

$$\mathbf{x} \in \mathbb{R}^{d+1}, \quad \mathbf{w}_1^{(1)}, \mathbf{w}_2^{(1)} \in \mathbb{R}^{d+1}, \quad \mathbf{a}^{(1)} \in \mathbb{R}^{2+1}.$$

In the next layer, the activations are computed in the same way as in layer 1. The activation vector is

$$\mathbf{a}^{(2)} = (a_0^{(2)} = 1, a_1^{(2)}, a_2^{(2)}),$$

and each activation is defined as

$$a_1^{(2)} = h^{(2)}(z_1^{(2)}),\ a_2^{(2)} = h^{(2)}(z_2^{(2)}), \quad \text{where} \quad z_1^{(2)} = \left(\mathbf{a}^{(1)}\right)^\top \mathbf{w}_1^{(2)},\ z_2^{(2)} = \left(\mathbf{a}^{(1)}\right)^\top \mathbf{w}_2^{(2)}.$$

The weight vectors $\mathbf{w}_1^{(2)}$ and $\mathbf{w}_2^{(2)}$ represent the blue and green lines respectively between layer 1 and layer 2. The activation function $h^{(2)}$ can be different from the activation function $h^{(1)}$. The dimensionality of the new terms we have defined are

$$\mathbf{w}_1^{(2)}, \mathbf{w}_2^{(2)} \in \mathbb{R}^{2+1}, \quad \mathbf{a}^{(2)} \in \mathbb{R}^{2+1}.$$

Finally, the output of the NN is the activation of the last layer, $a^{(3)}$, defined as

$$a^{(3)} = h^{(3)}(z^{(3)}), \quad \text{where} \quad z^{(3)} = \left(\mathbf{a}^{(2)}\right)^\top \mathbf{w}^{(3)}.$$

The weight vector $\mathbf{w}^{(3)}$ represents the blue lines between layer 2 and the output. The function $h^{(3)}$ can again be different from the activation functions $h^{(1)}$ and $h^{(2)}$. The dimensionality of the new terms we have defined are

$$\mathbf{w}^{(3)} \in \mathbb{R}^{2+1}, \quad a^{(3)} \in \mathbb{R}.$$

Now that we have studied some specific neural networks, we will introduce the general definition of a neural network in the next section.

10.2 Neural Networks: The General Case

In general, a neural network f can have any number of layers $B \in \{1, \dots\}$. Each layer $b \in \{1, \dots, B\}$ can have any number of non-bias neurons $d^{(b)} \in \{1, \dots\}$. The number of non-bias neurons in the input layer will sometimes be denoted as $d^{(0)} = d$. Each layer except the output layer has a bias neuron always set to 1, thus the number of neurons in layer $b \in \{0, \dots, B-1\}$ is $d^{(b)} + 1$. The output layer does not have a bias neuron, since its activations are not used to compute any other activations, thus the number of neurons in the output layer is $d^{(B)}$. A general NN structure is shown in Fig. 10.3. For an input feature

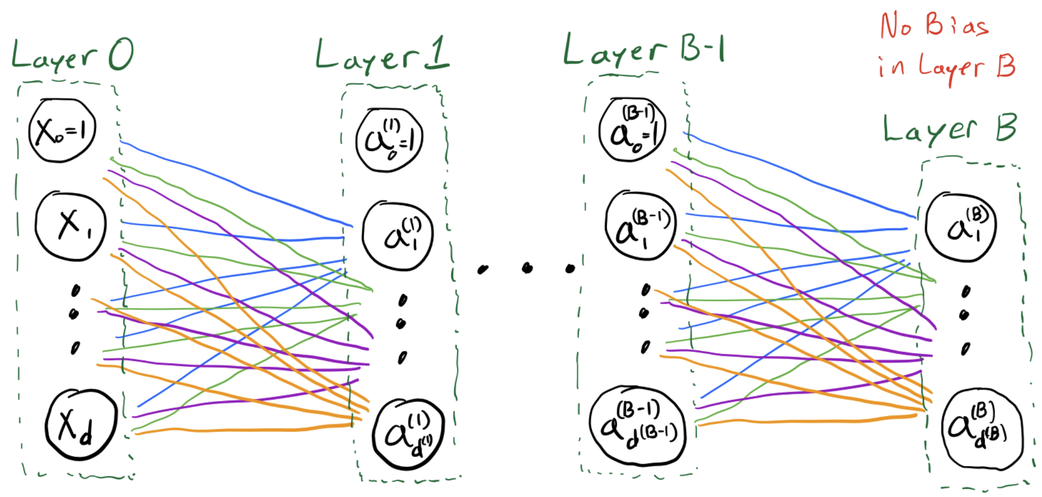


Figure 10.3: A general neural network structure.

vector $\mathbf{x}$, the output of the NN is $f(\mathbf{x}) = \mathbf{a}^{(B)}$. Next, we define the terms in a NN, which leads to the definition of $\mathbf{a}^{(B)}$. For any layer $b \in \{1, \ldots, B\}$, we have

$$\textbf{Weights: } \mathbf{w}_1^{(b)}, \ldots, \mathbf{w}_{d^{(b)}}^{(b)} \in \mathbb{R}^{d^{(b-1)}+1}$$

$$\textbf{Pre-activations: } \mathbf{z}^{(b)} = \left(z_1^{(b)}, \ldots, z_{d^{(b)}}^{(b)}\right) \in \mathbb{R}^{d^{(b)}},$$

$$\text{where} \quad z_j^{(b)} = \left(\mathbf{a}^{(b-1)}\right)^\top \mathbf{w}_j^{(b)} \quad \text{for } j \in \{1, \ldots, d^{(b)}\}.$$

$$\textbf{Activation function: } h^{(b)} : \mathbb{R} \to \mathbb{R}$$

$$\textbf{Activations: } \mathbf{a}^{(0)} = \mathbf{x} \in \mathbb{R}^{d+1} = \mathbb{R}^{d^{(0)}+1},$$

$$\mathbf{a}^{(b)} = \left(a_0^{(b)} = 1, a_1^{(b)}, \ldots, a_{d^{(b)}}^{(b)}\right) \in \mathbb{R}^{d^{(b)}+1} \quad \text{except } b = B,$$

$$\mathbf{a}^{(B)} = \left(a_1^{(B)}, \ldots, a_{d^{(B)}}^{(B)}\right) \in \mathbb{R}^{d^{(B)}},$$

$$\text{where} \quad a_j^{(b)} = h^{(b)}(z_j^{(b)}) \quad \text{for } j \in \{1, \ldots, d^{(b)}\}.$$

We can simplify the above equations by using matrix notation. We will store all the weight vectors $\mathbf{w}_1^{(b)}, \ldots, \mathbf{w}_{d^{(b)}}^{(b)}$ at a layer b as columns in a matrix

$$\mathbf{W}^{(b)} = \begin{bmatrix} | & | & & | \\ \mathbf{w}_1^{(b)} & \mathbf{w}_2^{(b)} & \cdots & \mathbf{w}_{d^{(b)}}^{(b)} \\ | & | & & | \end{bmatrix}.$$

The matrix $\mathbf{W}^{(b)}$ has $d^{(b-1)}+1$ rows and $d^{(b)}$ columns. The activation vector at layer b can now be written as

$$\mathbf{a}^{(b)} = h^{(b)}(\mathbf{z}^\top) \quad \text{where} \quad h^{(b)}(\mathbf{z}^\top) = (h^{(b)}(z_1), \ldots, h^{(b)}(z_{d^{(b)}})) \quad \text{and} \quad \mathbf{z}^\top = \left(\mathbf{a}^{(b-1)}\right)^\top \mathbf{W}^{(b)}.$$

This implies that the NN function can be written as follows

$$f(\mathbf{x}) = \mathbf{a}^{(B)} = h^{(B)}\left(h^{(B-1)}\left(\cdots h^{(2)}\left(h^{(1)}\left(\mathbf{x}^\top \mathbf{W}^{(1)}\right)\mathbf{W}^{(2)}\right)\cdots \mathbf{W}^{(B-1)}\right)\mathbf{W}^{(B)}\right)^\top.$$

The above equation clearly shows that $f(\mathbf{x})$ is a composition of the activation functions $h^{(b)}$ applied to the pre-activations $\mathbf{z}^{(b)}$ at each layer b. Using the above equation we can also easily motivate the use of activation functions. Suppose that we did not use any activation functions, which is equivilent to using the identity function $h^{(B)}(z) = \cdots = h^{(1)}(z) = z$. Then, the NN function would be

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{W}^{(1)}\mathbf{W}^{(2)}\cdots\mathbf{W}^{(B)} = \mathbf{x}^\top \mathbf{W},$$

where $\mathbf{W} = \mathbf{W}^{(1)}\mathbf{W}^{(2)}\cdots\mathbf{W}^{(B)}$. This is just a linear function. Thus, if we would like the NN function to be able to represent non-linear relationships between the features and the label, we should use activation functions that are non-linear.

Every NN satisfies the above equations. Thus, to define a NN we need to specify the number of layers B, the number of neurons in each layer $d^{(1)}, \ldots, d^{(B)}$, the activation functions $h^{(1)}, \ldots, h^{(B)}$, and the weights $\mathbf{w}_1^{(b)}, \ldots, \mathbf{w}_{d^{(b)}}^{(b)}$ for each layer $b \in \{1, \ldots, B\}$. If we only specify the number of layers, number of neurons in each layer, and the activation functions, we will have a *NN architecture*. For instance if the activation functions are all the logistic function, then in Section 10.1 we studied the NN architecture with $B = 3$ layers, and $d^{(1)} = 2, d^{(2)} = 2, d^{(3)} = 1$ neurons in each layer. Some of the functions we have seen before can also be represented as NNs.

Example 10.1: [Linear Function]
The linear function $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$ can be represented as a NN with $B = 1$ layer and $d^{(1)} = 1$ neuron. The activation function is the identity function $h^{(1)}(z) = z$. The weight vector is $\mathbf{w}_1^{(1)} = \mathbf{w}$. Visually, the NN is shown in Fig. 10.4. □

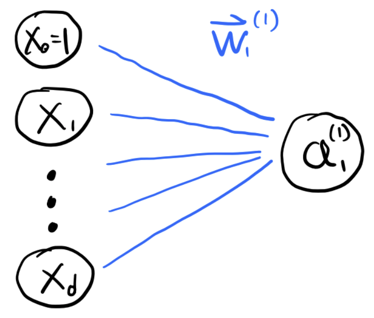


Figure 10.4: A NN representing the linear or logistic function.

Example 10.2: [Logistic Function] The logistic function applied to a linear function $f(\mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w})$ can be represented as a NN with $B = 1$ layer and $d^{(1)} = 1$ neuron. The activation function is the logistic function $h^{(1)}(z) = \sigma(z)$. The weight vector is $\mathbf{w}_1^{(1)} = \mathbf{w}$. Notice how this can again be visualized as in Fig. 10.4, since the only difference is the activation function. □

Notice how once a specifc NN architecture is chosen, the NN is uniquely defined by its weights. Importantly, this implies that optimizing over a set of NNs with a fixed architecture is equivalent to optimizing over the set of weights that uniquely define the NNs[1]. We will make use of this property in the next section, where we describe how ERM can be used with the NN function class.

[1]It may seem arbitrary as to why we choose to only optimize over the weights and not any other term that defines the NN architecture, such as the number of layers or the number of neurons in each layer. The reason for this is that the NN is a continuous function of the weights, that is, small changes in the weights result in small changes in the NN values. On the other hand, small changes in the number of layers or the number of neurons in each layer can result in large changes in the NN values. If we were to optimize over the number of layers or the number of neurons in each layer, we would be optimizing over a discrete space, which is usually more difficult and the techniques we learned in Chapter 6 would not apply. The same reasoning applies for why we did not optimize over the polynomial degree p in Section 6.6

10.3 ERM with Neural Networks

In this section we will discuss the ERM learner that uses a NN function class. The ERM learner is defined as

$$\mathcal{A}(\mathcal{D}) = \hat{f} = \operatorname*{argmin}_{f \in \mathcal{F}} \hat{L}(f) \quad \text{where} \quad \hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i).$$

The function class $\mathcal{F}$ is the set of all NNs with a fixed architecture, written as

$$\mathcal{F} = \{f | f : \mathcal{X} \to \mathcal{Y} \text{ where } f \text{ is a NN with a fixed architecture}\}.$$

As we discussed in the previous section, the important thing to notice is that since $\mathcal{F}$ contains NNs with a fixed architecture, each NN is uniquely defined by its weights. In particular, each $f \in \mathcal{F}$ can be written as $f(\mathbf{x}) = \mathbf{a}^{(B)}$ where $\mathbf{a}^{(B)}$ is computed using some weight matrices $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(B)}$. To make this relationship explicit, we will write a NN from the function class with a subscript $f_{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(B)}}$ to denote the weights that define the NN. Now, we can optimize over the weights to find the best NN in the function class $\mathcal{F}$. The ERM learner becomes

$$\mathcal{A}(\mathcal{D}) = \hat{f} \quad \text{where} \quad \hat{f}(\mathbf{x}) = \mathbf{a}^{(B)} \text{ is defined by the weight matrices}$$
$$\hat{\mathbf{W}}^{(1)}, \ldots, \hat{\mathbf{W}}^{(B)} = \operatorname*{argmin}_{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(B)}} \frac{1}{n}\sum_{i=1}^{n} \ell\left(f_{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(B)}}(\mathbf{x}_i), y_i\right).$$

Unless the activation functions are all the identity function, there is usually no closed form solution to the above optimization problem. Thus, we have to use gradient descent to find the optimal weights. The update rule for the weight vector at layer $b \in \{1, \ldots, B\}$ and neuron $j \in \{1, \ldots, d^{(b)}\}$ is

$$(\mathbf{w}_j^{(b)})^{(t+1)} = (\mathbf{w}_j^{(b)})^{(t)} - \eta^{(t)} \nabla_{\mathbf{w}_j^{(b)}} \hat{L}\left(f_{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(B)}}\right),$$
$$\text{where} \quad \hat{L}\left(f_{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(B)}}\right) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(f_{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(B)}}(\mathbf{x}_i), y_i\right).$$

The gradient can be worked out by using several applications of the chain rule, however, we will not work out the details in these notes. The process of computing the gradient in the above update rule is often referred to as *backpropagation*, since the gradients at layer b can be shown to depend on the gradients at layer $b + 1$, hence propogating backwards through the layers.

Unfortunately, unless the neural network is very simple (such as linear or a logistic function), and the loss function is chosen appropriately, the optimization problem is non-convex in general. This means that gradient descent might not approach the actual minimizer of the estimated loss function. Instead, gradient descent might get stuck in what is called a *local minimum.* The details of non-convex optimization and how NNs behave is beyond the scope of these notes.

10.4 Neural Networks Learning Better Features

In this section our goal is to show what the activations of the NN output by the ERM learner might represent.

Before we begin, let us return to the only method we have used so far to improve our feature representation, the polynomial feature map ϕ_p. Suppose that the label has a quadratic relationship with the first feature x_1, that is $y \approx x_1^2$. To model this relationship we would need to use at least $p = 2$ in the polynomial feature map ϕ_p. If $d = 3$ then $\phi_2(\mathbf{x}) = (1, x_1, x_2, x_3, x_1^2, x_1x_2, x_1x_3, x_2^2, x_2x_3, x_3^2) \in \mathbb{R}^{10}$, which has 6 new features compared to the original feature vector $\mathbf{x}$. We can visualize the new feature space and how it can be used in a linear function in Fig. 10.5. Notice, however, that only the new feature x_1^2

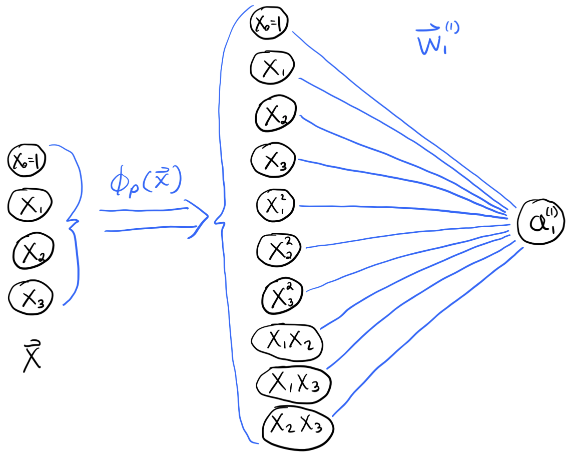


Figure 10.5: The polynomial feature map $\phi_2(\mathbf{x})$ and a linear function in the new features.

is relevant for the supervised learning problem at hand. As such, 5 of the new features in $\phi_2(\mathbf{x})$ are irrelevant, namely, $x_1x_2, x_1x_3, x_2^2, x_2x_3, x_3^2$. It would be much more desirable if we could have a function that could learn which features are relevant for the supervised learning. This is where the NN comes in. The claim is that the activations of the NN output by the ERM learner can be thought of as representing better features as we move through the layers of the NN. To illustrate this we will use a classic example of classifying wether an image contains a cat or not. This task is challenging because the features that define a "cat" are not directly apparent from the raw input data, but the NN learns to extract and combine these features effectively.

Example 10.3: [Image Classification] Suppose that we have an image classification problem where the input is a black and white 16×16 pixel image and the set of labels is $\mathcal{Y} = \{0, 1\}$, where 1 represents that the image contains a cat. Similar to logistic regression (see Section 9.1.1), we would like to predict the probability that the image contains a cat, but now with a NN. We need to convert the pixel image into a feature vector $\mathbf{x}$. The pixel image contains $16 \times 16 = 256$ pixels, each representing the intensity of the pixel. We can flatten the pixel image into a feature vector and add a bias term to get $\mathbf{x} \in \mathbb{R}^{256+1}$. This is shown in Fig. 10.6.

We decide to use a NN architecture with $B = 3$ layers, $d^{(1)} = 100, d^{(2)} = 50, d^{(3)} = 1$

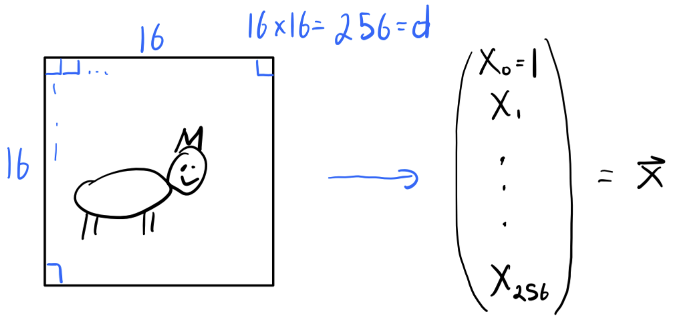


Figure 10.6: The flattened pixel image.

neurons in each layer, and the logistic function for all the activation functions. The output of the neural network is $f(\mathbf{x}) = \mathbf{a}^{(3)}$, representing the probability that the image contains a cat. This is shown in Fig. 10.7. The ERM learner will learn some weight matrices using

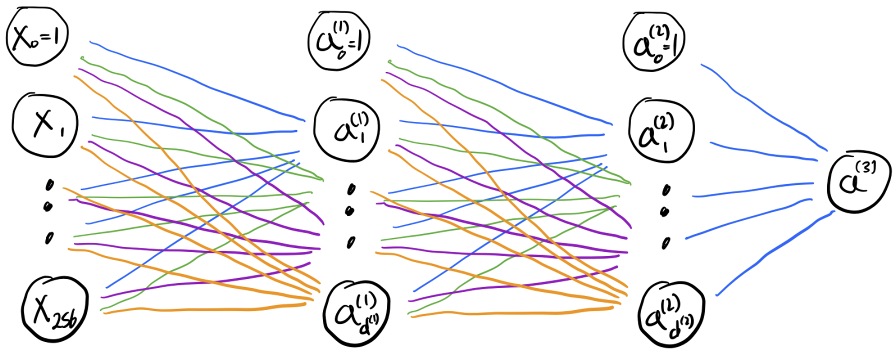


Figure 10.7: A NN for image classification.

gradient descent. On the zeroth epoch, the weights are initialized randomly, and thus the activations of the NN are random and do not represent anything meaningful. However, as the number of epochs increases, the activations of the NN will start to represent better features. The type of features we might expect the activations to learn after a large number of epochs is often described as follows. Lets first focus on the activation $a_1^{(1)}$ from the first layer. This activation will have different values depending on the input image $\mathbf{x}$. Suppose that we took the feature vector $\mathbf{x}$ that makes the value of $a_1^{(1)}$ largest and visualized the image. What we would likely find is that this is an image of something like the edge a cats back or the edges of a cats feet. This image is thought of as the feature that $a_1^{(1)}$ has learned to represent. If we perform the same procedure for all of the activations in the first layer, and would likely find that each activation represents a different feature, each of which is slightly more complex than the raw pixel values.

We can of course perform the same procedure for the activations in the second layer. What we would likely find is that the activations in the second layer represent more complex features, such as the shape of the cat's head or the cat's torso. The features that some of the activations might learn are shown in Fig. 10.8. In general as you move along the layers of

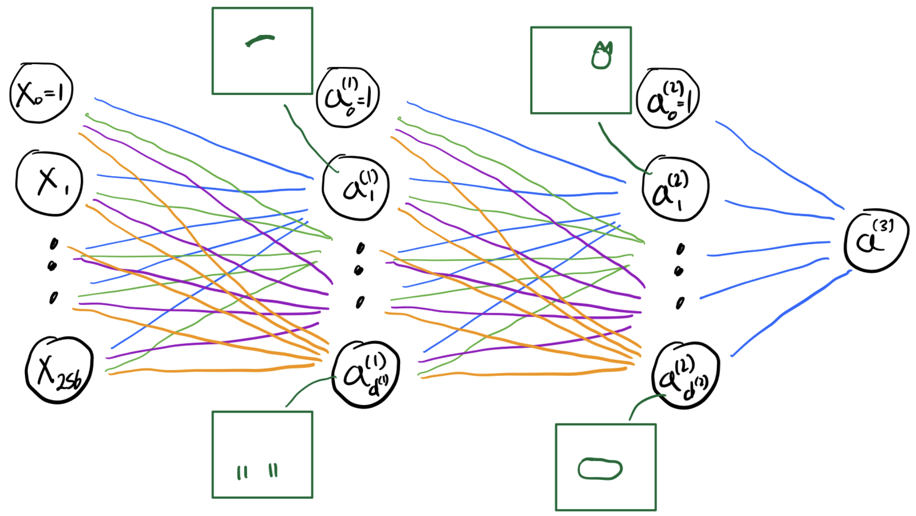


Figure 10.8: The features learned by the activations in the first and second layers of the NN.

the NN, the activations will represent more and more complex features. Since the features that the activations represent depend on the problem at hand, the NN is thought of as learning problem specific features, unlike the polynomial feature map ϕ_p which is fixed and might contain irrelevant features. □

10.5 Softmax as a Neural Network

The careful reader may have noticed that so far we have compared all the functions we have covered in previous chapters to the NN, except for the softmax function. The reason for this is that softmax cannot be represented as a single layer NN. In this section we will show that for this reason the softmax is often added as a post-processing step to the output of a single layer NN.

In Section 9.2.1 we introduced the softmax function to predict a vector of probabilities. The softmax function was applied to K different linear functions. To get a sense for why the softmax function cannot be represented as a signle layer NN, consider the case where $K = 3$. We would have 3 weight vectors $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ and the softmax function would be

$$\begin{aligned}&\sigma(\mathbf{x}^\top\mathbf{w}_1, \mathbf{x}^\top\mathbf{w}_2, \mathbf{x}^\top\mathbf{w}_3)\\ &\quad = \Big(\sigma_1(\mathbf{x}^\top\mathbf{w}_1, \mathbf{x}^\top\mathbf{w}_2, \mathbf{x}^\top\mathbf{w}_3), \sigma_2(\mathbf{x}^\top\mathbf{w}_1, \mathbf{x}^\top\mathbf{w}_2, \mathbf{x}^\top\mathbf{w}_3), \sigma_3(\mathbf{x}^\top\mathbf{w}_1, \mathbf{x}^\top\mathbf{w}_2, \mathbf{x}^\top\mathbf{w}_3)\Big)^\top.\end{aligned}$$

If we try to represent the softmax function as a NN with one layer $B = 1$ we would need $d^{(1)} = 3$ neurons since the output is $K = 3$ dimensional. If we set the weight vectors as $\mathbf{w}_1^{(1)} = \mathbf{w}_1, \mathbf{w}_2^{(1)} = \mathbf{w}_2, \mathbf{w}_3^{(1)} = \mathbf{w}_3$, then the output of the NN would be

$$f(\mathbf{x}) = \mathbf{a}^{(1)} = \Big(h^{(1)}(\mathbf{x}^\top\mathbf{w}_1), h^{(1)}(\mathbf{x}^\top\mathbf{w}_2), h^{(1)}(\mathbf{x}^\top\mathbf{w}_3)\Big)^\top.$$

This is shown in Fig. 10.9. Unfortunately, there is no way to set the activation function

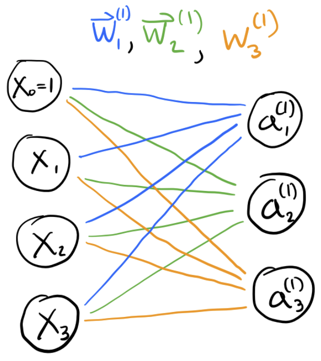


Figure 10.9: A NN attempting to represent the softmax function.

$h^{(1)}$ such that the output of the NN is the same as the softmax function. The problem arises in the fact that each index of the output of the softmax function depends on all of the linear functions, while in the NN each output activation can only depend on the input feature vector $\mathbf{x}$ and the weights connecting to it.

One way to get around this is to add an extra post-processing step to the output of the NN. In particular if we set the activation function $h^{(1)}$ to be the identity function, then the output of the NN is

$$f(\mathbf{x}) = \mathbf{a}^{(1)} = \left(\mathbf{x}^\top \mathbf{w}_1, \mathbf{x}^\top \mathbf{w}_2, \mathbf{x}^\top \mathbf{w}_3\right)^\top .$$

Then if we apply the softmax function to the output of the NN, we get the same output as the softmax function.

$$\sigma(f(\mathbf{x})) = \sigma\left(\mathbf{x}^\top \mathbf{w}_1, \mathbf{x}^\top \mathbf{w}_2, \mathbf{x}^\top \mathbf{w}_3\right).$$

Exercise 10.1: Let $d = 2$. Define a NN f that satisfies $f(\mathbf{x}) = x_1 + x_2$. □

Exercise 10.2: Let $d = 2$. Define a NN f that satisfies $f(\mathbf{x}) = x_1 - x_2$. □

Exercise 10.3: Let $d = 2$. Assume $x_1, x_2 > 0$. Define a NN f that satisfies $f(\mathbf{x}) = x_1 \cdot x_2$. □

Exercise 10.4: Let $d = 2$. Assume $x_1, x_2 > 0$. Define a NN f that satisfies $f(\mathbf{x}) = x_1 / x_2$. □

Chapter 11

Language Models

In this chapter we will study a topic that is currently very popular: generating text. We will refer to a predictor or model that can generate text as a *language model*. Some examples of what generating text can look like are:

Example 11.1:

Input: `Once upon a time, in a land far, far away,`
Output: `there lived a wise old dragon who loved to read books.`

□

Example 11.2:

Input: `Why did the chicken cross the road?`
Output: `To get to the other side.`

□

The second example should bring to mind how you might use ChatGPT, a language model developed by OpenAI. In particular, if your input is a question, ChatGPT will generate a reasonable answer. Although there are various ways to generate text, in this chapter we will focus on a particular type of language model: the *autoregressive language model*. Roughly speaking an autoregressive language model is a model that generates text one word at a time and uses as input the original text plus the words it has already generated. Since we will only focus on autoregressive language models in this chapter, and will refer to them simply as language models.

Example 11.3: In Example 11.2, an autoregressive language model would generate the text `To get to the other side.` as follows:

Input: `Why did the chicken cross the road?`
Output: `To`
Input: `Why did the chicken cross the road?  To`
Output: `get`

⋮

Input: `Why did the chicken cross the road?  To get to the other side`
Output: `.`
Input: `Why did the chicken cross the road?  To get to the other side.`
Output: `<EOS>`

Notice how the output at each step is used as part of the input for the next step. The text `<EOS>` is called an "end of sequence token" and is used by the model to indicate that it has finished generating text. □

In the above example you might have noticed that the model may need to output symbols that are not words, such as a period or the end of sequence token. As such we will refer to possible outputs of the model as *tokens* rather than words, and the set of all possible tokens $\mathcal{Y}$ as the *vocabulary*. An example of a vocabulary could be the set that contains all English words, punctuation marks, the end of sequence token `<EOS>`, and a padding token `<PAD>` that is used to fill in the input sequence when it is shorter than the maximum length the model can handle. Although the vocabulary can be quite large, it is finite and we will denote its size as $K = |\mathcal{Y}|$. Since there are finitely many tokens we could alternatively represent them as integers in the set $\{1, 2, \ldots, K\}$. Using the integer representation will occasionally be more convenient, so we will often change between the two representations.

As such, we can try to formally represent a language model as a function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is the vocabulary and $\mathcal{X}$ is the set of all possible sequences of tokens. Unfortunately, if we try to use ERM to learn a good model, we will quickly run into two problems. The first is that the domain of the model $\mathcal{X}$ is not continuous, which will make optimization difficult. The second is that even if the domain was continuous, the output of the model is discontinuous, since the model outputs discrete tokens, which will again make optimization difficult.

11.1 Representing Tokens as Vectors

First, we will address the issue of the input being discontinuous. So far in the notes we have always assumed that the input to the model is a feature vector in $\mathbb{R}^{d+1}$. To make use of all the tools we have developed so far, we will need a way to represent a sequence of tokens as feature vector. The first step in this process is to notice that each token in a sequence of token belongs to the same vocabulary $\mathcal{Y}$. Thus, we can represent a sequence of tokens as a tuple $s \in \mathcal{Y}^a$ where a is the length of the sequence. Then, we can make use of what is called an *embedding* function to represent each token as a vector in $\mathbb{R}^{d'}$. The embedding function will be denoted as $E : \mathcal{Y} \to \mathbb{R}^{d'}$. Although there is a lot of details about how to pick a good embedding function, in these notes we will assume that a good embedding function has been given to us. In Section 11.1.1 we will discuss some properties of embedding functions and how to intuitively think of tokens as vectors.

Now we need to represent a sequence of tokens $s \in \mathcal{Y}^a$ as a feature vector. We can try to first pass each token through the embedding function and then concatenate the resulting vectors. The only issue with this approach is that the length of the sequence a can vary, and we need a feature vector of fixed dimension $d + 1$. As such, we will choose some fixed length $c \in \{1, 2, \ldots\}$ called the *context length*. If the sequence is shorter than c, we will pad it with the padding token `<PAD>`. If the sequence is longer than c, we will only keep the last c tokens.

Example 11.4: Suppose that $c = 3$ and the input text is `Why did`, which contains two tokens `Why` and `did` and can be equivilently represented as the tuple $(\texttt{Why}, \texttt{did}) \in \mathcal{Y}^2$. However, we need to represent this as a tuple in $\mathcal{Y}^3$. We can do this by padding it with the token `<PAD>` to get $(\texttt{<PAD>}, \texttt{Why}, \texttt{did}) \in \mathcal{Y}^3$. □

Example 11.5: Suppose that $c = 3$ and the input text is `Why did the chicken cross the road?`, which contains eight tokens and can be equivilently represented as the tuple $(\texttt{Why}, \texttt{did}, \texttt{the}, \texttt{chicken}, \texttt{cross}, \texttt{the}, \texttt{road}, \texttt{?}) \in \mathcal{Y}^8$. However, we need to represent this as a tuple in $\mathcal{Y}^3$. We can do this by keeping only the last three tokens to get $(\texttt{the}, \texttt{road}, \texttt{?}) \in \mathcal{Y}^3$. □

Then, if we have a seuquence of tokens $s \in \mathcal{Y}^c$, we can represent it as a feature vector $\mathbf{x} \in \mathbb{R}^{d+1}$ by passing each token through the embedding function, concatenating the resulting vectors and adding a bias term. This process can be represented by a concatenation embedding function $\bar{E} : \mathcal{Y}^c \to \mathbb{R}^{d+1}$, where $d = c \cdot d'$. We can visualize this in Fig. 11.1, where the blue box represents the concatenation embedding function. The tokens in the sequence are denoted as $v_1, \ldots, v_c$ and the corresponding embeddings are denoted as $\mathbf{e}_1, \ldots, \mathbf{e}_c$. The output of $\bar{E}(s)$ is a feature vector $\mathbf{x} \in \mathbb{R}^{d+1}$ as desired.

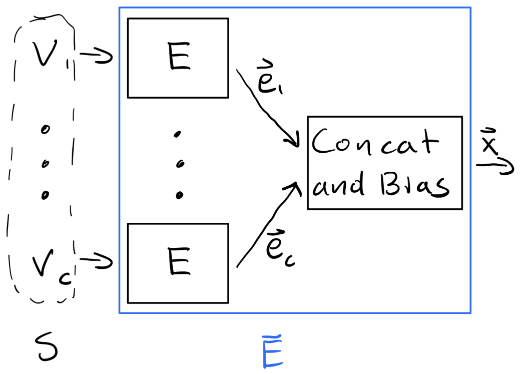


Figure 11.1: A concatenation embedding function.

Exercise 11.1: Suppose that $s = (\texttt{Why}, \texttt{did}, \texttt{the})$ and $E(\texttt{Why}) = (1, 2)^\top$, $E(\texttt{did}) = (3, 4)^\top$, and $E(\texttt{the}) = (5, 6)^\top$. What is $\bar{E}(s)$? □

11.1.1 Token Embeddings

In this section we will discuss some properties of embedding functions and how to intuitively think of tokens as vectors. The first property that most embedding functions have is that words that are similar in meaning should have similar embeddings. For instance, if you compare the embeddings of the words `big` and `large`, you will find that they are quite similar. Another property is that doing arithmetic with embeddings can give meaningful results. Suppose that we have the embeddings of the words `man` and `woman`. If we subtract $E(\texttt{woman}) - E(\texttt{man})$ we will get a vector that represents something like the concept of femininity. If we add this vector to the embedding of the word `king`, we will get something close to the embedding of the word `queen`. As one last example, suppose that we would like to know the capital of `France` and we know that the capital of `Italy` is `Rome`. Then, if we compute $E(\texttt{Rome}) - E(\texttt{Italy}) + E(\texttt{France})$ we will get something close to the embedding of the word `Paris`. The examples described above are illistrated in Fig. 11.2. Since it is difficult to visualize more than 2 dimensions we have assumed that $d' = 2$; however, in practice d' is usually much larger.

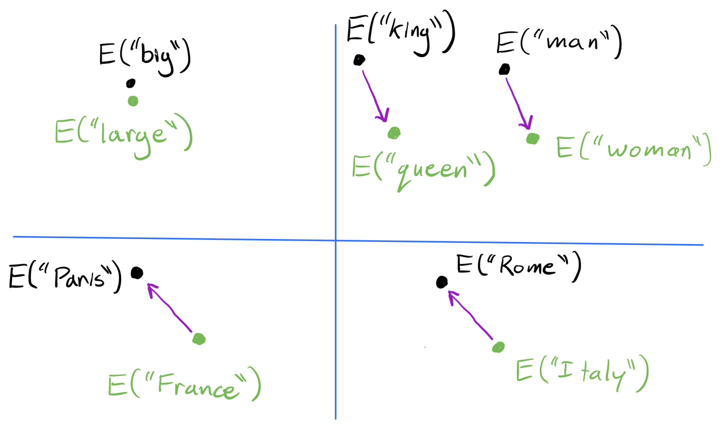


Figure 11.2: Some properties of token embeddings.

Exercise 11.2: Would you expect the embedding of the word `great` to be closer to the embedding of the synonym `good` or the antonym `bad`? ☐

Exercise 11.3: Suppose that you have the embeddings of the words `far`, `farthest`, and `near`. How would you use these embeddings to get an estimate of the embedding of the word `nearest`? ☐

11.2 Predicting the Next Token

Next, we will address the issue of the output being discontinuous. To do this we will make use of a strategy we have already discussed in Section 9.2 of first predicting the probability of each token in the vocabulary, and then selecting the token with the highest probability. Formally, we need to first get an estimate of the conditional pmf $p(y|\mathbf{x})$ for each $y \in \mathcal{Y}$ and $\mathbf{x} \in \mathcal{X}$ and then select the token y that maximizes this probability. Getting an estimate of $p(y|\mathbf{x})$ can be throught of as a regression problem in the same way we have seen Section 9.2.1. In particular, since the vocabulary $\mathcal{Y}$ is finite, then for a fixed $\mathbf{x}$ we can represent $p(y|\mathbf{x})$ as a vector $\mathbf{y} \in [0,1]^K$ where $y_k = p(y = k|\mathbf{x})$. Then, the new set of labels becomes $\mathcal{Y}_{\text{prob}} = [0,1]^K$. A model that outputs a vector of probabilities $\mathbf{y}$ will be denoted as $f_{\text{prob}} : \mathcal{X} \to \mathcal{Y}_{\text{prob}}$ and called a probability model.

In Section 9.2.1 we saw that one option for f_{prob} is to use a softmax function that takes as input K linear functions of the input $\mathbf{x}$. We can generalize this further by passing the features $\mathbf{x}$ through a neural network (NN) before applying the softmax function. The NN will often use the identity activation function for the output layer $h^{(B)}(z) = z$. If we let $f_{\text{NN}}(\mathbf{x}) = \mathbf{z}^{(B)}$ be the output of the NN, then $f_{\text{prob}}(\mathbf{x}) = \sigma(\mathbf{z}^{(B)})$. We can visualize this in Fig. 11.3, where the blue box represents f_{prob}.

Exercise 11.4: Suppose that the vocabulabry $\mathcal{Y}$ contains 100 tokens. How many neurons does the output layer of f_{NN} need to have, and will the sum of all the output neuron values be equal to 1? ☐

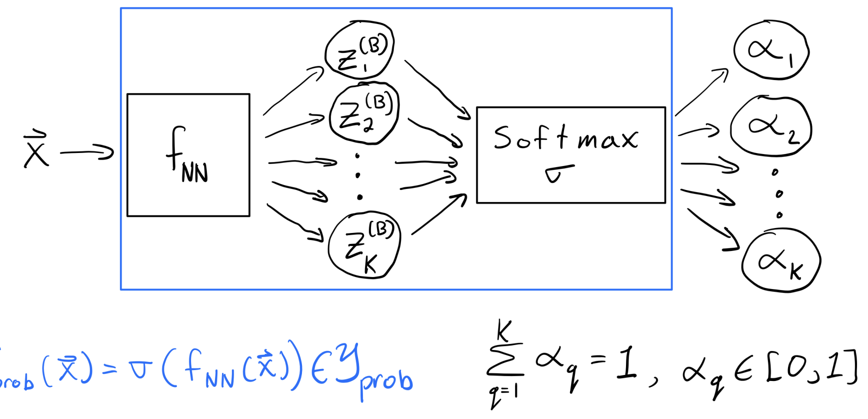

$\vec{x}$
f_{NN}
$z_1^{(B)}$
$z_2^{(B)}$
$z_K^{(B)}$
Softmax σ
α_1
α_2
α_K
$f_{prob}(\vec{x}) = \sigma(f_{NN}(\vec{x})) \in \mathcal{Y}_{prob}$
$\sum_{q=1}^{K} \alpha_q = 1, \alpha_q \in [0,1]$

Figure 11.3: A NN with a softmax applied to the output.

11.3 Creating a Dataset

In Section 11.1 we discussed how to represent a sequence of tokens as a feature vector in $\mathbb{R}^{d+1}$. In Section 11.2 we chose to represent the probability model $f_{\text{prob}} : \mathcal{X} \to \mathcal{Y}_{\text{prob}}$ as the softmax of the output of a NN. What we have not yet discussed is how to learn a good probability model. To do this we can use ERM; however, we will first need a dataset. We will assume that we have access to a long sequence of tokens $s \in \mathcal{Y}^n$, for example, the text of many books concatenated together. In this section we will discuss how to use this data to create a dataset that can be used learn a good probability model with ERM.

Since the probability model f_{prob} is a function that takes as input a feature vector $\mathbf{x} \in \mathbb{R}^{d+1}$ and outputs a probability vector $\mathbf{y} \in [0,1]^K$, we will need to create a dataset that contains feature-label pairs of the same type. First, we will need to split the sequence of tokens into input sequences s_i of length c and output tokens y_i. Roughly speaking, we can do this by using a sliding window of length c to create the input sequences and then take the next token as the output token. More formally, if we have a sequence of tokens $s = (v_1, v_2, \dots, v_{n+1})$, we can create the input-output pairs (s_i, y_i) where $s_i = (v_{i-c+1}, v_{i-c+2}, \dots, v_i)$ and $y_i = v_{i+1}$ for $i \in \{c, c+1 \dots, n\}$. To make sure we have some data points where the input sequence is shorter than c, we will also create input-output pairs where the input sequence is padded with the token `<PAD>`. In particular, for $i \in \{1, \dots, c-1\}$ we will create the input-output pairs (s_i, y_i) where $s_i = (\texttt{<PAD>}, \dots, \texttt{<PAD>}, v_1, \dots, v_i) \in \mathcal{Y}^c$ and $y_i = v_{i+1}$.

Example 11.6: Suppose that $c = 3$ and the sequence of tokens is

$$(\texttt{Why}, \texttt{did}, \texttt{the}, \texttt{chicken}, \texttt{cross}, \texttt{the}, \texttt{road}, \texttt{?}, \texttt{<EOS>}).$$

Then, the input-output pairs would be:

$$\begin{aligned}
(s_1, y_1) &= ((\texttt{<PAD>}, \texttt{<PAD>}, \texttt{Why}), \texttt{did}),\\
(s_2, y_2) &= ((\texttt{<PAD>}, \texttt{Why}, \texttt{did}), \texttt{the}),\\
(s_3, y_3) &= ((\texttt{Why}, \texttt{did}, \texttt{the}), \texttt{chicken}),\\
(s_4, y_4) &= ((\texttt{did}, \texttt{the}, \texttt{chicken}), \texttt{cross}),\\
(s_5, y_5) &= ((\texttt{the}, \texttt{chicken}, \texttt{cross}), \texttt{the}),\\
(s_6, y_6) &= ((\texttt{chicken}, \texttt{cross}, \texttt{the}), \texttt{road}),\\
(s_7, y_7) &= ((\texttt{cross}, \texttt{the}, \texttt{road}), \texttt{?}),\\
(s_8, y_8) &= ((\texttt{the}, \texttt{road}, \texttt{?}), \texttt{<EOS>}),
\end{aligned}$$

□

Exercise 11.5: Suppose that we have created the following input-output pairs from some sequence of tokens:

$$\begin{aligned}
(s_1, y_1) &= ((\texttt{<PAD>}, \texttt{<PAD>}, \texttt{<PAD>}, \texttt{<PAD>}, \texttt{Why}), \texttt{did}),\\
(s_2, y_2) &= ((\texttt{<PAD>}, \texttt{<PAD>}, \texttt{<PAD>}, \texttt{Why}, \texttt{did}), \texttt{the}),\\
(s_3, y_3) &= ((\texttt{<PAD>}, \texttt{<PAD>}, \texttt{Why}, \texttt{did}, \texttt{the}), \texttt{chicken}),\\
(s_4, y_4) &= ((\texttt{<PAD>}, \texttt{Why}, \texttt{did}, \texttt{the}, \texttt{chicken}), \texttt{cross}),\\
(s_5, y_5) &= ((\texttt{Why}, \texttt{did}, \texttt{the}, \texttt{chicken}, \texttt{cross}), \texttt{the}),\\
(s_6, y_6) &= ((\texttt{did}, \texttt{the}, \texttt{chicken}, \texttt{cross}, \texttt{the}), \texttt{road}),\\
(s_7, y_7) &= ((\texttt{the}, \texttt{chicken}, \texttt{cross}, \texttt{the}, \texttt{road}), \texttt{?}),\\
(s_8, y_8) &= ((\texttt{chicken}, \texttt{cross}, \texttt{the}, \texttt{road}, \texttt{?}), \texttt{<EOS>}),
\end{aligned}$$

What is the context length c and what was the original sequence of tokens? □

Then, to create the feature vectors we would pass each input sequence s_i through the concatenation embedding function $\bar{E} : \mathcal{Y}^c \to \mathbb{R}^{d+1}$ to get the feature vectors $\mathbf{x}_i = \bar{E}(s_i)$. To create the label vectors we would pass each output token y_i through the one-hot encoding function one-hot : $\mathcal{Y} \to [0, 1]^K$ to get the label vectors $\mathbf{y}_i = \text{one-hot}(y_i)$. The one-hot encoding function is defined as one-hot$(y) = (y_1, y_2, \ldots, y_K)$ where $y_k = 1$ if $y = k$ and $y_k = 0$ otherwise. Finally, we would have a dataset of feature-label pairs $\mathcal{D} = ((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n))$ that can be used to learn a good probability model with ERM.

Exercise 11.6: The only tokens you will encounter are `Why`, `did`, `the`, `?`, `<EOS>`, `<PAD>`. You represent each token by an integer as follows `Why` $\to$ 1, `did` $\to$ 2, `the` $\to$ 3, `?` $\to$ 4, `<EOS>` $\to$ 5, and `<PAD>` $\to$ 6. Thus, the vocabulary can be either $\mathcal{Y} = \{1, 2, 3, 4, 5, 6\}$, or $\mathcal{Y} = \{\texttt{Why}, \texttt{did}, \texttt{the}, \texttt{?}, \texttt{<EOS>}, \texttt{<PAD>}\}$ depending on the context. Suppose that you are creating a dataset and the first input-output pair is (s_1, y_1) where $s_1 = (\texttt{<PAD>}, \texttt{<PAD>}, \texttt{<PAD>}, \texttt{<PAD>}, \texttt{did})$ and $y_1 = \texttt{the}$. You use an embedding function $E : \mathcal{Y} \to \mathbb{R}^{d'}$, where $d' = 2$. What is the one-hot label vector $\mathbf{y}_1$ and what is the dimension of the feature vector $\mathbf{x}_1$? □

11.4 ERM for Language Models

Given the dataset $\mathcal{D}$ we created in the previous section what is left to define is the function class $\mathcal{F}$ and the loss function ℓ. The function class $\mathcal{F}$ will obviously be the set of all functions of the same form as f_{prob}. Formally,

$$\mathcal{F} = \{f | f : \mathbb{R}^{d+1} \to [0,1]^K \text{ where } f = \sigma(f_{\text{NN}}(\mathbf{x})) \text{ and } f_{\text{NN}} \text{ is a NN with a fixed architecture}\}.$$

We will pick the multiclass cross-entropy loss (discussed in Section 9.2.1) as the loss function, since the task of predicting the next token is a multiclass classification problem. Then, the ERM learner is

$$\mathcal{A}(\mathcal{D}) = \hat{f}_{\text{prob}} = \operatorname*{argmin}_{f \in \mathcal{F}} \hat{L}(f).$$

As discussed in Section 10.3, since each f_{NN} is defined by its weight matrices, the above minimization can be converted to a minimization over the weight matrices. Solving this minimization problem can then be done using gradient descent. We will again leave out the details of computing the gradients in these notes.

Finally, we can define our language model which takes as input a sequence of tokens $s \in \mathcal{Y}^c$ and outputs the next token. First, we would pass the input sequence through the concatenation embedding function $\bar{E} : \mathcal{Y}^c \to \mathbb{R}^{d+1}$ to get the feature vector $\mathbf{x} = \bar{E}(s)$. Then, we would pass the feature vector through the learned probability model $\hat{f}_{\text{prob}}$ to get the probability vector $\hat{\mathbf{y}} = \hat{f}_{\text{prob}}(\mathbf{x})$. Finally, we would select the token with the highest probability as the next token.

Exercise 11.7: Based on the above paragraph, if the input sequence is $s \in \mathcal{Y}^c$, is the output of the language model

$$\operatorname*{argmax}_{k \in \mathcal{Y}} \hat{y}_k \quad \text{where} \quad \hat{y} = \hat{f}_{\text{prob}}(\bar{E}(s))?$$

□

Chapter 12

Exercise Solutions

Chapter 2 Solutions

Exercise 2.15

This can be solved using the chain rule. Let $u(x) = -x^3$. Then we can rewrite $f(x)$ as $f(x) = \exp(u(x))$. We know that $\frac{du}{dx} = -3x^2$. Thus, using the chain rule, we get:

$$\frac{df}{dx}(x) = \frac{df}{du}\frac{du}{dx}(x) = \exp(-x^3)(-3x^2) = -3x^2\exp(-x^3).$$

Chapter 3 Solutions

Exercise 3.2

$$\begin{aligned}
\mathbb{E}_{\mathcal{X}}[\mathbb{E}_{\mathcal{Y}}[Y|X]] &= \sum_{x\in\mathcal{X}} p(x) \sum_{y\in\mathcal{Y}} yp(y|x) \\
&= \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} yp(y|x)p(x) \\
&= \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} yp(x,y) \\
&= \sum_{y\in\mathcal{Y}} y \sum_{x\in\mathcal{X}} p(x,y) \\
&= \sum_{y\in\mathcal{Y}} yp(y) \\
&= \mathbb{E}[Y]
\end{aligned}$$

Exercise 3.3

1. $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$

2. Let $X \in \mathcal{X}$ be the random variable representing the outcome of the die roll. Since it is a fair die X has a discrete uniform distribution, which implies its pmf is $p(x) = \frac{1}{6}$ for $x \in \mathcal{X}$. Getting an even number can be represented as the event $\mathcal{E} = \{2, 4, 6\}$. The probability of observing an even number is then

$$\mathbb{P}(\mathcal{E}) = \sum_{x\in\mathcal{E}} p(x) = p(2) + p(4) + p(6) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}$$

3. Let event $\mathcal{E} = \{3\}$, then the event of not observing 3 is $\mathcal{E}^c$ and its probability is

$$\mathbb{P}(X \in \mathcal{E}^c) = 1 - \mathbb{P}(X \in \mathcal{E}) = 1 - \frac{1}{6} = \frac{5}{6}$$

4. Let event $\mathcal{E} = \{3, 4\}$, then $\mathbb{P}(X \in \mathcal{E}) = p(3) + p(4) = 2/6$. The probability of not observing 3 or 4 is $\mathbb{P}(X \in \mathcal{E}^c) = 1 - \mathbb{P}(X \in \mathcal{E}) = 4/6$

Exercise 3.4
In a Bernoulli trial, if the probability of success is α, then the probability of failure is $1 - \alpha$ which will be 0.3.
Exercise 3.5

1.

$$\begin{aligned}\mathbb{P}(X \leq 4.5) &= \int_{-5}^{4.5} p(x)dx \\ &= \int_{-5}^{4.5} \frac{1}{b-a}dx \\ &= \int_{-5}^{4.5} \frac{1}{10}dx \\ &= \frac{1}{10}(4.5 - (-5)) \\ &= 0.95\end{aligned}$$

2.

$$\begin{aligned}\mathbb{P}(-3 \leq X \leq 3) &= \int_{-3}^{3} p(x)dx \\ &= \int_{-3}^{3} \frac{1}{10}dx \\ &= \frac{1}{10}(6) \\ &= 0.6\end{aligned}$$

Exercise 3.6

1. Since $\sum_{x\in\{1,2,3\},y\in\{1,2,3\}} p(x, y) = 1$, we expand and see that

$$\sum_{x\in\{1,2,3\},y\in\{1,2,3\}} p(x, y) = 15/18 + c = 1.$$

   Thus. c must be $\frac{3}{18}$ for this to hold.

2. $p(x = 3) = \sum_{y\in\{1,2,3\}} p(x = 3, y) = \frac{5}{18}$

3. $p(y = 1) = \sum_{x\in\{1,2,3\}} p(x, y = 1) = \frac{5}{18}$

Exercise 3.7

1. The outcome space for X is $\mathcal{X} = [0, 1]$ and the outcome space for Y is $\mathcal{Y} = [1, \infty)$

2. The joint outcome space for the new random variable $Z = (X, Y)$ is $\mathcal{Z} = [0, 1] \times [1, \infty)$.

3.

$$\begin{aligned}1 &= \int_{\mathcal{X}}\int_{\mathcal{Y}} p(x,y)dydx\\ &= \int_0^1\int_1^\infty \frac{cx^2}{y^2}dydx\\ &= \int_0^1 \left(-\frac{cx^2}{y}\right)\Big|_1^\infty dx\\ &= \int_0^1 cx^2dx\\ &= \frac{cx^3}{3}\Big|_0^1 = \frac{c}{3} - 0 = \frac{c}{3}\\ \implies c &= 3\end{aligned}$$

4. Notice that $p(y|x=0.5) = \frac{p(x=0.5,y)}{p(x=0.5)}$ and that $p(x=0.5,y) = 3(0.5)^2/y^2 = 0.75/y^2$. Now we just need $p(x=0.5)$.

$$\begin{aligned}p(x=0.5) &= \int_{\mathcal{Y}} p(x=0.5,y)dy\\ &= \int_{\mathcal{Y}} \frac{0.75}{y^2}dy\\ &= 0.75\int_1^\infty y^{-2}dy\\ &= 0.75(-y^{-1})\Big|_1^\infty\\ &= 0.75(0-(-1^{-1})) = 0.75\end{aligned}$$

Therefore $p(y|x=0.5) = 1/y^2$.

Chapter 6 Solutions

Exercise 6.1
No you can not be certain. We know when $\eta^t \to 0$ that $\mathbf{w}^{(t)} \to \mathbf{w}^*$, but we do not know if $\mathbf{w}^{(t)} = \mathbf{w}^*$ for any specific t. Also, since we are using a fixed step size, it does not even satisfy that $\eta^t \to 0$.
Exercise 6.2
If you set $\lambda = 0$, then the step size becomes $\eta^t = \eta\exp(-\lambda t) = \eta\exp(0) = \eta$ for all $t \in \mathbb{N}$, which is a constant step size.
Exercise 6.3
By following the same steps as for the squared loss it can be shown that the BGD update rule with this new loss would be:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)}\frac{4}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top\mathbf{w}^{(t)} - y_i)^3\mathbf{x}_i. \qquad (12.1)$$

In the 1-dimensional case, the first derivative of $\hat{L}(w)$ becomes:

$$\frac{d}{dw}\hat{L}(w) = \frac{4}{n}\sum_{i=1}^{n}(x_i^\top w - y_i)^3 x_i \tag{12.2}$$

To check convexity we take the second derivative:

$$\frac{d^2}{dw^2}\hat{L}(w) = \frac{12}{n}\sum_{i=1}^{n}(x_i^\top w - y_i)^2 x_i^2 \tag{12.3}$$

Since this is always positive, $\hat{L}(w)$ is convex.
Exercise 6.4
See algorithm 5 for the algorithm.

Algorithm 5: BGD Linear Regression Learner (with an exponential decaying step size)

```
1: input: $\mathcal{D} = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$, step size parameters $\eta_0, \lambda$, number of epochs $T$
2: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^{d+1}$
3: for $t = 1, \ldots, T$ do
4:     $\nabla\hat{L}(\mathbf{w}) \leftarrow \frac{2}{n}\sum_{i=1}^{n}(\mathbf{x}_i^\top \mathbf{w} - y_i)\mathbf{x}_i$
5:     $\eta \leftarrow \eta_0 \exp(-\lambda(t-1))$
6:     $\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla\hat{L}(\mathbf{w})$
7: return $\hat{f}(\mathbf{x}) = \mathbf{x}^T\mathbf{w}$
```

Exercise 6.5

$$\hat{L}_{M+1}(\mathbf{w}) = \frac{1}{n - Mb}\sum_{i=Mb+1}^{n-Mb}(\mathbf{x}_i^\top \mathbf{w} - y_i)^2.$$

Exercise 6.6
It is not certain that $f_3 \in \mathcal{F}_2$ because f_3 is a degree 3 polynomial, and $\mathcal{F}_2$ is the set of all degree 2 polynomials. For example if $f_3(x) = x^3$, then $f_3 \notin \mathcal{F}_2$. However, it is certain that $f_3 \in \mathcal{F}_4$ because $\mathcal{F}_4$ is the set of all degree 4 polynomials, and f_3 is a degree 3 polynomial.
Exercise 6.7
Since $d = 3, p = 2$, we have $\bar{p} = \binom{3+2}{2} = 10$. Let $\mathbf{x} = (x_0 = 1, x_1, x_2, x_3)^\top \in \mathbb{R}^{d+1} = \mathbb{R}^4$. Then $\phi_2(\mathbf{x}) = (1, x_1, x_2, x_3, x_1^2, x_1x_2, x_1x_3, x_2^2, x_2x_3, x_3^2)^\top \in \mathbb{R}^{10}$.
Exercise 6.8
Yes $\bar{\mathcal{F}}_1 \subset \bar{\mathcal{F}}_2$. To see this notice that any function $\bar{f}_1 \in \bar{\mathcal{F}}_1$ can be written as $\exp(\phi_1(\mathbf{x})^\top\mathbf{w}) = \exp(\mathbf{x}^\top\mathbf{w})$ for some $\mathbf{w} = (w_0, \ldots, w_d) \in \mathbb{R}^{d+1}$. Also, $\mathbf{x}^\top\mathbf{w} = \phi_2(\mathbf{x})^\top\mathbf{w}'$ for some $\mathbf{w}' = (w_0, \ldots, w_d, 0, \ldots, 0) \in \mathbb{R}^{\bar{p}}$ where $\bar{p} = \binom{d+2}{2}$. Thus, $\bar{f}_1 \in \bar{\mathcal{F}}_2$. Since $\bar{f}_1$ was arbitrary, this shows that $\bar{\mathcal{F}}_1 \subset \bar{\mathcal{F}}_2$.

$\bar{\mathcal{F}}_1 \not\subset \mathcal{F}_1$. To see this, consider the function $f(x) = \exp(x) \in \bar{\mathcal{F}}_1$. However, $f(x) \notin \mathcal{F}_1$ because $f(x)$ is not a polynomial. Similarly, $\mathcal{F}_1 \not\subset \bar{\mathcal{F}}_1$ because there are polynomials in $\mathcal{F}_1$ that are not in $\bar{\mathcal{F}}_1$.

It is not true that $\min_{f\in\bar{\mathcal{F}}_1}\hat{L}(f) \leq \min_{f\in\bar{\mathcal{F}}_2}\hat{L}(f)$. This follows immediately from the fact that $\bar{\mathcal{F}}_1 \subset \bar{\mathcal{F}}_2$, thus there might be a $\bar{f}_2 \in \bar{\mathcal{F}}_2$ that has a lower value of $\hat{L}$ than any $\bar{f}_1 \in \bar{\mathcal{F}}_1$. However, it is true that $\min_{f\in\bar{\mathcal{F}}_1}\hat{L}(f) \geq \min_{f\in\mathcal{F}_2}\hat{L}(f)$.

Exercise 6.9

With gradient descent we can not be certain how good our solution will be after T iterations. So we can not be certain that $\hat{L}_1(w_1^{(100)}) \geq \hat{L}_2(w_2^{(1000)})$. However, we can be pretty confident that it is true since $\mathcal{F}_1 \subset \mathcal{F}_2$, and we are told that the number of iterations that gradient descent is run is likely enough to find a good approximation of the minimizer.

Exercise 6.10

To construct a degree 3 polynomial feature map $\phi_3(\mathbf{x})$, we need to construct all possible triples of the input features $x_0, x_1, \ldots, x_d$. We follow the example for the degree 2 case (algorithm 4), but now add an extra nested loop to consider all possible triples of the input features, instead of just pairs. The solution is given in algorithm 6.

Algorithm 6: Degree 3 Polynomial Feature Map

```
1: input: feature vector $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_d)^\top \in \mathbb{R}^{d+1}$
2: $\bar{p} \leftarrow (d+1)(d+2)(d+3)/6$
3: $\varphi \leftarrow (\varphi_0 = 0, \varphi_1 = 0, \ldots, \varphi_{\bar{p}-1} = 0)^\top \in \mathbb{R}^{\bar{p}}$
4: $j \leftarrow 0$
5: for $k = 0, \ldots, d$ do
6:    for $l = k, \ldots, d$ do
7:       for $q = l, \ldots, d$ do
8:          $\varphi_j = x_k \cdot x_l \cdot x_q$
9:          $j = j + 1$
10: return $\varphi$
```

Chapter 8 Solutions

Exercise 8.1

Since log is a monotonic function, we can minimize the negative log-likelihood instead of the likelihood. So now we have to minimize $-\log \prod_{i=1}^n \alpha^{z_i}(1-\alpha)^{1-z_i} = -\sum_{i=1}^n [z_i \log \alpha + (1-z_i)\log(1-\alpha)]$. Taking derivative with respect to α and setting it to zero, we get $-\sum_{i=1}^n \left[\frac{z_i}{\alpha} - \frac{1-z_i}{1-\alpha}\right] = 0$. Solving for α we get $\alpha = \frac{1}{n}\sum_{i=1}^n z_i$, which is the sample mean of the data.

Exercise 8.2

The negative log-likelihood for $\mathcal{N}(0, \sigma^2)$ is

$$\sum_{i=1}^n \left[\frac{1}{2}\log(2\pi\sigma^2) + \frac{(z_i - 0)^2}{2\sigma^2}\right] = \frac{n}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^n z_i^2.$$

Taking derivative w.r.t σ and setting it to 0:

$$\frac{n}{2}\frac{4\pi\sigma}{2\pi\sigma^2} - \frac{1}{\sigma^3}\sum_{i=1}^n z_i^2 = 0 \implies \sigma^2 = \frac{1}{n}\sum_{i=1}^n z_i^2.$$

Exercise 8.3

The negative log-likelihood for Poisson(λ) is

$$-\sum_{i=1}^n \log \frac{e^{-\lambda}\lambda^{z_i}}{z_i!} = n\lambda - \sum_{i=1}^n z_i \log \lambda + \sum_{i=1}^n \log z_i!\,.$$

Taking the derivative w.r.t λ of the above expression and setting it to 0, we get

$$n - \frac{1}{\lambda}\sum_{i=1}^{n} z_i = 0 \implies \lambda = \frac{1}{n}\sum_{i=1}^{n} z_i.$$

Exercise 8.4
The likelihood for Poisson(λ) is

$$p(\mathcal{D}|\lambda) = \prod_{i=1}^{n} \frac{\lambda_i^z}{z_i!} e^{-\lambda} = \frac{\lambda^{\sum_{i=1}^{n} z_i}}{z_1! \dots z_n!} e^{-n\lambda}.$$

Then the posterior with a Gamma prior (with paratmeters $k = 3$ and $\theta = 1$) is proportional to (removing terms that do not depend on λ)

$$p(\lambda|\mathcal{D}) \propto p(\mathcal{D}|\lambda)p(\lambda) = \lambda^{\sum_{i=1}^{n} z_i} e^{-n\lambda}\lambda^2 e^{-\lambda} = \lambda^{\sum_{i=1}^{n} z_i + 2} e^{-(n+1)\lambda}.$$

Taking the negative log of the above expression, we get $-(\sum_{i=1}^{n} z_i + 2)\log\lambda + (n+1)\lambda$. Taking the derivative w.r.t λ and setting it to 0, we get $-\frac{\sum_{i=1}^{n} z_i+2}{\lambda} + (n+1) = 0 \implies \lambda = \frac{\sum_{i=1}^{n} z_i+2}{n+1}$.

Chapter 9 Solutions

Exercise 9.1
For all epochs $T \in \{1, \dots, 200\}$ it holds that

$$L(f_{\text{bin}}^{(T)}) \geq L(f^*) \geq L(f_{\text{Bayes}}),$$

by the definitions of $f_{\text{bin}}^{(T)}, f^*, f_{\text{Bayes}}$. The fact that the estimated loss using binary corss-entropy is going down as T increases, implies that $\hat{f}^{(T)}$ is getting closer to $\hat{f}_{\text{ERM}}$, and is not important for this question. How the value of $\hat{L}(f_{\text{bin}}^{(T)})$ compares to $L(f_{\text{bin}}^{(T)}), L(f^*), L(f_{\text{Bayes}})$ can not be determined from the information we are given, since $\hat{L}$ and L are different functions.

Exercise 9.2

1. Yes, to both. Since $\sigma(\phi_p(\mathbf{x})^\top \mathbf{w}_{\text{MLE, p}}) = 0.5$ is equivalent to $\phi_p(\mathbf{x})^\top \mathbf{w}_{\text{MLE, p}} = 0$.

2. The left plot is $p = 1$ since the right plot must be $p = 5$ since the decision boundary is a curve and when $p = 1$ the equation $\mathbf{x}^\top \mathbf{w}_{\text{MLE, 1}} = 0$ defines a line.

3. In both plots the green region represents when the binary classification predictor outputs 1 and the red region represents when the binary classification predictor outputs 0. We know this since there are significantly more green points in the green region and more red points in the red regions, which implies that $f_{\text{MLE, p}}(\mathbf{x}) \geq 0.5$ in the green region and $f_{\text{MLE, p}}(\mathbf{x}) < 0.5$ in the red region.

4. By changing the threshold to 0.1 the features $\mathbf{x}$ for which $0.5 > f_{\text{MLE, p}}(\mathbf{x}) \geq 0.1$ would now become a green region as well. This means the green region would be at least as large as it is now, and the red would be at most as large as it is now.

5. Notice that the zero-one loss is 1 if $f_{\text{Bin, p}}(\mathbf{x}) \neq y$ and 0 otherwise. Thus, the estimated loss is the proportion of points for which $f_{\text{Bin, p}}(\mathbf{x}) \neq y$. If you count the number of add up the number of red points in the green region and the number of green points in the red region in each plot, you will see that the number is smaller for $p = 5$. This means that the estimated loss is smaller for $p = 5$.

6. $p = 5$ is more likely to overfit the dataset $\mathcal{D}$. This is because the decision boundary for $p = 5$ is more complex than the decision boundary for $p = 1$. Since the decision boundary for $p = 5$ is more complex, it is more likely to fit the noise in the dataset $\mathcal{D}$.

Exercise 9.3

1. The condition that holds when all three decision boundaries intersect is $\sigma_0 = \sigma_1 = \sigma_2$.

2. The equation of the line representing the decision boundary for classes 0 and 1 is $\mathbf{x}^\top \mathbf{w}_{\text{MLE},0} = \mathbf{x}^\top \mathbf{w}_{\text{MLE},1}$, since this implies that $\sigma_0 = \sigma_1$. This line is only plotted for the values of $\mathbf{x}$ where $\sigma_0 = \sigma_1 \geq \sigma_2$.

3. The decision boundary for some classes $y \neq k$ would now be represented by $\phi_p(\mathbf{x})^\top \mathbf{w}_{\text{MLE},y} = \phi_p(\mathbf{x})^\top \mathbf{w}_{\text{MLE},k}$. Since $p > 2$ the decision boundaries would be curves, unless a line is a better fit for the data, which does not seem likely based on the plot.

Chapter 10 Solutions

Exercise 10.1
The neural network has $B = 1$ layers, $d^{(1)} = 1$ output neuron. The activation function is $h^{(1)}(z) = z$. The weights are $\mathbf{w}_1^{(1)} = (0, 1, 1)^\top$. Then, the output activation is $a_1^{(1)} = h^{(1)}(\mathbf{x}^\top \mathbf{w}_1^{(1)}) = h^{(1)}(x_0 \cdot 0 + x_1 \cdot 1 + x_2 \cdot 1) = x_1 + x_2$.
Exercise 10.2
The neural network has $B = 1$ layers, $d^{(1)} = 1$ output neuron. The activation function is $h^{(1)}(z) = z$. The weights are $\mathbf{w}_1^{(1)} = (0, 1, -1)^\top$. Then, the output activation is $a_1^{(1)} = h^{(1)}(\mathbf{x}^\top \mathbf{w}_1^{(1)}) = h^{(1)}(x_0 \cdot 0 + x_1 \cdot 1 + x_2 \cdot 1) = x_1 + x_2$.
Exercise 10.3
Notice that $x_1 \cdot x_2 = \exp(\log(x_1 \cdot x_2)) = \exp(\log(x_1) + \log(x_2))$. We use this insight to design the network. The neural network has $B = 2$ layers, $d^{(1)} = 2, d^{(2)} = 1$ neurons. The activation functions are $h^{(1)}(z) = \log(x)$, $h^{(2)}(z) = \exp(x)$. The weights are $\mathbf{w}_1^{(1)} = (0, 1, 0)^\top, \mathbf{w}_2^{(1)} = (0, 0, 1)^\top, \mathbf{w}_1^{(2)} = (0, 1, 1)^\top$. Then, the output activations are

$$\begin{aligned}
a_1^{(1)} &= h^{(1)}(\mathbf{x}^\top \mathbf{w}_1^{(1)}) = h^{(1)}(x_0 \cdot 0 + x_1 \cdot 1 + x_2 \cdot 0) = \log(x_1), \\
a_2^{(1)} &= h^{(1)}(\mathbf{x}^\top \mathbf{w}_2^{(1)}) = \log(x_2), \\
a_1^{(2)} &= h^{(2)}(\mathbf{x}^\top \mathbf{w}_1^{(2)}) = \exp(\log(x_1) + \log(x_2)) = x_1 \cdot x_2.
\end{aligned}$$

Alternatively, we could have noticed that $x_1 \cdot x_2 = \frac{1}{2}(x_1 + x_2)^2 - \frac{1}{2}x_1^2 - \frac{1}{2}x_2^2$, and designed the network accordingly.
Exercise 10.4
Notice that $x_1/x_2 = \exp(\log(x_1/x_2)) = \exp(\log(x_1) - \log(x_2))$. We use this insight to design the network. The neural network has $B = 2$ layers, $d^{(1)} = 2, d^{(2)} = 1$ neurons.

The activation functions are $h^{(1)}(z) = \log(x)$, $h^{(2)}(z) = \exp(x)$. The weights are $\mathbf{w}_1^{(1)} = (0,1,0)^\top, \mathbf{w}_2^{(1)} = (0,0,1)^\top, \mathbf{w}_1^{(2)} = (0,1,-1)^\top$. Then, the output activations are

$$\begin{aligned}
a_1^{(1)} &= h^{(1)}(\mathbf{x}^\top \mathbf{w}_1^{(1)}) = h^{(1)}(x_1),\\
a_2^{(1)} &= h^{(1)}(\mathbf{x}^\top \mathbf{w}_2^{(1)}) = h^{(1)}(x_2),\\
a_1^{(2)} &= h^{(2)}(\mathbf{x}^\top \mathbf{w}_1^{(2)}) = \exp(\log(x_1) - \log(x_2)) = x_1/x_2.
\end{aligned}$$

Chapter 11 Solutions

Exercise 11.1
Since $\bar{E}(s)$ concatenates the embeddings of the tokens in s and adds a bias term, we have

$$\bar{E}(s) = (1,1,2,3,4,5,6)^\top.$$

Exercise 11.2
The embedding of the word `great` would be closer to the embedding of the word `good` than to the embedding of the word `bad`, since `great` and `good` are similar in meaning by definition of being synonyms.
Exercise 11.3
The vector $E($`farthest`$) - E($`far`$)$ would represent something like the concept of being the most of something, then if we add this to the embedding of `near`, we would get something close to the embedding of `nearest`. In other words $E($`nearest`$) \approx E($`near`$) + E($`farthest`$) - E($`far`$)$.
Exercise 11.4
The output layer will need to have 100 ouput neurons, one for each token in the vocabulary. The sum of all the output neuron values is not equal to 1 in general since the softmax function has not been applied to the output yet.
Exercise 11.5
The context length is 5 since $s_1, \ldots, s_8$ all have 5 tokens each. The original sequence was (`Why`, `did`, `the`, `chicken`, `cross`, `the`, `road`, `?`, `<EOS>`) due to the definition of the input-output pairs (s_i, y_i).
Exercise 11.6
The one-hot label vector $\mathbf{y}_1 = (0,0,1,0,0,0)^\top$, since $K = 6$ and `the` is represented as the number 3. The dimension of $\mathbf{x}_1$ is $= c \cdot d' + 1 = 5 \cdot 2 + 1 = 11$.
Exercise 11.7
Yes.

Appendix A Solutions

Exercise A.1
Consider an event $A \in \mathcal{E}$. By property 1 of the event space, $A^c \in \mathcal{E}$. Since the intersection $A \cap A^c = \varnothing$, and we assume that P satisfies the axioms of probability, then by axiom 2 we

have

$$\begin{aligned}
\mathbb{P}(A \cup A^c) &= \mathbb{P}(A) + \mathbb{P}(A^c) \\
\mathbb{P}(\Omega) &= \mathbb{P}(A) + \mathbb{P}(A^c) \\
1 &= \mathbb{P}(A) + \mathbb{P}(A^c) \\
\mathbb{P}(A) &= 1 - \mathbb{P}(A^c)
\end{aligned}$$

Exercise A.2
By property 3, we know $\mathcal{E} \neq \varnothing$. Thus there is some event $A \in \mathcal{E}$. By property 1, we know that $A^c \in \mathcal{E}$. By property 2 we then know that $A \cup A^c = \Omega \in \mathcal{E}$. By property 1, $\Omega^c = \varnothing \in \mathcal{E}$.
Exercise A.3
The first axiom is straightforward to prove by combining the two statements we are given.

$$\begin{aligned}
\mathbb{P}(\mathcal{X}) &= \sum_{x \in \mathcal{X}} p(x) && \text{(by second statement)} \\
&= 1 && \text{(by first statement)}
\end{aligned}$$

To prove the second axiom, first suppose that $A_1, A_2, \ldots \in \mathcal{E}$, $A_i \cap A_j = \varnothing\ \forall i, j$.

$$\begin{aligned}
\mathbb{P}(A_1, A_2, ... \in \mathcal{E}) &= \sum_{x \in \cup_i A_i} p(x) && \text{(by second statement)} \\
&= \sum_i \sum_{x \in A_i} p(x) && \text{(because } A_i \cap A_j = \varnothing\text{)} \\
&= \sum_i \mathbb{P}(A_i) && \text{(be second statement)}
\end{aligned}$$

Appendix A

Extra Details on Probabilities

In this section, we more carefully go through probabilities and random variables. The main set of notes covers only what is strictly necessary. But, the definitions for probabilities are actually quite simple, so simple in fact that we can build them up from scratch. We do that here for the interested reader.

We can build up rules of probability, based on an elegantly simple set of axioms called the *axioms of probability.* Let the *sample space* (Ω) be a non-empty set of outcomes and the *event space* (σ) be a non-empty set of subsets of Ω. For example, $\Omega = \{1, 2, 3\}$ and one possible *event* is $\mathcal{E} = \{1, 3\} \in \sigma$, where the event is that a 1 or a 3 is observed. Another possible event is $\mathcal{E} = \{2\}$, where the event is that a 2 is observed. The **event space** σ must satisfy the following properties[1]

1. $\mathcal{E} \in \sigma \quad \Rightarrow \quad \mathcal{E}^c \in \sigma$ (where $\mathcal{E}^c$ is the complement of the event $\mathcal{E}$: $\mathcal{E}^c = \Omega \backslash \mathcal{E}$)
2. $\mathcal{E}_1, \mathcal{E}_2, \ldots \in \sigma \quad \Rightarrow \quad \bigcup_{i=1}^{\infty} \mathcal{E}_i \in \sigma$
3. σ is non-empty.

If σ satisfies these three properties, then (Ω, σ) is said to be a *measurable space.* The symbol $\varnothing$ means the empty set. Note that these three conditions imply that $\varnothing \in \sigma$ and $\Omega \in \sigma$.[2]

Now we can define the axioms of probability, which make it more clear why these two conditions are needed for our event space to define meaningful probabilities over events. A function $\mathbb{P} : \sigma \to [0, 1]$ satisfies the **axioms of probability** if

1. $\mathbb{P}(\Omega) = 1$
2. $\mathcal{E}_1, \mathcal{E}_2, \ldots \in \sigma,\ \mathcal{E}_i \cap \mathcal{E}_j = \varnothing\ \forall i, j \quad \Rightarrow \quad \mathbb{P}(\cup_{i=1}^{\infty} \mathcal{E}_i) = \sum_{i=1}^{\infty} \mathbb{P}(\mathcal{E}_i)$

is called a *probability measure* or a *probability distribution.* The tuple $(\Omega, \sigma, \mathbb{P})$ is called the *probability space.*

The second condition means that the probability of the union of disjoint sets equals the sum of their probabilities. This is an intuitive requirement. In it simplest form, this requirement states that if two events $\mathcal{E}_1, \mathcal{E}_2$ are disjoint, then $\mathbb{P}(\mathcal{E}_1 \cup \mathcal{E}_2) = \mathbb{P}(\mathcal{E}_1) + \mathbb{P}(\mathcal{E}_2)$: the probability of either event occurring is the sum of their probabilities, because there is no overlap in the outcomes in the events. More generally, a finite union should also satisfy this

[1] Such a set is usually called a *sigma algebra* or *sigma field.* This terminology feels daunting and is only due to historical naming conventions. Because the sigma algebra is simply the set of events to which we can assign probabilities (measure), we will use this more clear name. If you would like to learn more about this topic, it is more formally discussed in a branch of mathematics called Measure Theory.

[2] In fact, it is common to use the condition that $\Omega \in \sigma$ rather than using the condition that σ is non-empty. However, it is equivalent and requiring that we have a non-empty event space is a more obvious condition to include.

equality, namely that for any $\mathcal{E}_1, \mathcal{E}_2, \dots \mathcal{E}_N \in \sigma$, $\mathcal{E}_i \cap \mathcal{E}_j = \varnothing \ \forall i, j \Rightarrow \mathbb{P}(\cup_{i=1}^N \mathcal{E}_i) = \sum_{i=1}^N \mathbb{P}(\mathcal{E}_i)$. The above condition requires this equality for the union of infinitely many events, and implies that $\mathbb{P}$ satisfies this condition for finitely many sets. This is because all $\mathcal{E}_{N+j}$ for $j \geq 1$ can be set to the empty set $\varnothing$, where $\mathcal{E}_i \cap \varnothing = \varnothing$ and $\varnothing \cap \varnothing = \varnothing$.

The beauty of these axioms lies in their compactness and elegance. Many useful expressions can be derived from the axioms of probability. For example, it is straightforward to show that $\mathbb{P}(\mathcal{E}^c) = 1 - \mathbb{P}(\mathcal{E})$. This makes it more clear why we required that if an event is in the event space, then its complement should also be in the event space: if we can measure the probability of an event, then we know that the probability of that event not occurring is 1 minus that probability. Another property we can infer is that we always have $\Omega, \varnothing \in \sigma$, where $\varnothing$ corresponds to the event where nothing occurs—which must have zero probability.

Exercise A.1: Show that for any event $\mathcal{E} \in \sigma$, $\mathbb{P}(\mathcal{E}^c) = 1 - \mathbb{P}(\mathcal{E})$. Assume that $\mathbb{P}$ satisfies the axioms of probability. □

Exercise A.2: Show that the above three conditions on σ imply that $\varnothing \in \sigma$ and $\Omega \in \sigma$. □

We can now introduce *random variables*, and from here on will deal strictly with random variables. A random variable lets us more rigorously define transformations of probability spaces; once we execute that transformation, we can forget about the underlying probability space and can focus on the events and distribution only on the random variable. This is in fact what you do naturally when defining probabilities over variables, without needing to formalize it mathematically. Of course, here we will formalize it.

Consider again the dice example, where now instead you might want to know: what is the probability of seeing a low number (1-3) or a high number (4-6)? We can define a new probability space with $\mathcal{X} = \{\text{low}, \text{high}\}$, $\sigma_X = \{\varnothing, \{\text{low}\}, \{\text{high}\}, \mathcal{X}\}$ and $\mathbb{P}_X(\{\text{low}\}) = 1/2 = \mathbb{P}_X(\{\text{high}\})$. The transformation function $X : \Omega \to \mathcal{X}$ is defined as

$$X(\omega) \overset{\text{def}}{=} \begin{cases} \text{low} & \text{if } \omega \in \{1,2,3\} \\ \text{high} & \text{if } \omega \in \{4,5,6\} \end{cases}$$

The distribution $\mathbb{P}_X$ is immediately determined from this transformation. For example, $\mathbb{P}_X(\{\text{low}\}) = \mathbb{P}(\{\omega : X(\omega) = \text{low}\})$,[3] because the underlying probability space indicates the likelihood of seeing a $1, 2$ or 3. Now we can answer questions about the probability of seeing a low number or a higher number.

This function X is called a random variable.[4] The utility of this terminology is that we move from talking about probabilities of sets—which can be cumbersome—to writing boolean statements. For example, we write $\mathbb{P}_X(X = x)$ or $\mathbb{P}_X(X \in \mathcal{E})$, rather than $\mathbb{P}(\{\omega : X(\omega) = x\})$ or $\mathbb{P}(\{\omega : X(\omega) \in \mathcal{E}\})$. For correctness, we can remember that it is a function defined on a more complex underlying probability space. But, in practice, we can start thinking directly in terms of the random variable X and the associated probabilities.

[3]You can read $\{\omega : X(\omega) = \text{low}\}$ as "The event where the outcome was low" and more explicitly as "The set of outcomes where the outcome was low". You can read this whole expression $\mathbb{P}(\{\omega : X(\omega) = \text{low}\})$ as "The probability of the event where the outcome was low".

[4]This terminology might be slightly confusing consider X is a function, so it is neither random, nor a variable. But, we end up using X as if its a variable that can take random outcomes, by writing statements like $X = x$. In this sense, the terminology is reasonable.

Similarly, even for the dice role, we can acknowledge that there is a more complex underlying probability space, defined by the dynamics of the dice. When considering only the probabilities of discrete outcomes from 1-6, we have already implicitly applied a transformation on top of probabilities of the physical system.

Once we have a random variable, it defines a valid probability space $(\mathcal{X}, \sigma_X, \mathbb{P}_X)$. Therefore, all the same rules of probability apply, the same understanding of how to define distributions, etc. In fact, we can always define a random variable X that corresponds to no transformation, to obtain the original probability space. For this reason, we can move forward assuming we are always dealing with random variables, without losing any generality. We will drop the subscripts, and consider $(\mathcal{X}, \sigma, \mathbb{P})$ to be defined for X. To define $\mathbb{P}$, we can use the definitions in Section 3.2.

Exercise A.3: Recall that for a given pmf p, we defined the probability of any event $\mathcal{E} \in \sigma$ as

$$\mathbb{P}(\mathcal{E}) \overset{\text{def}}{=} \sum_{x \in \mathcal{E}} p(x).$$

Show that this $\mathbb{P}$ satisfies the axioms of probability. □