Notes for EE2211 Introduction to Machine Learning

Vincent Y. F. Tan
Department of Electrical and Computer Engineering,
Department of Mathematics,
National University of Singapore
vtan@nus.edu.sg

April 7, 2021

Preface

Motivation

This set of notes spawned out of the tutorials for a module I taught in the Fall of 2020 and the Spring of 2021. The module, EE2211 *Introduction to Machine Learning*, is a core module that is a requirement for all sophomores of the Faculty of Engineering at the National University of Singapore. Machine learning is, obviously, prevalent and extremely useful to the practising engineer as well as anyone interested in dealing with and systematically analyzing data. It is my hope that the bite-sized summaries of the key topics as well as the additional examples will aid in the students' understanding of the material and also pique the students' interest of this fascinating subject. Sections marked as "Optional" are more advanced and are not required based on the syllabus of EE2211. In the Spring of 2021, I added practice problems to each of the chapters. Some of the problems may require background beyond the scope of the class and precise thinking, but they serve as good vehicles for character building in mathematics.

A word of caution: There is no guarantee of these notes being error- or typo-free. Comments are very much welcome. Please contact me at `vtan@nus.edu.sg` if you spot something that can be improved.

Prerequisites

While the level of mathematical sophistication to appreciate machine learning at the level of EE2211 is not high, it is nevertheless non-zero. Exposure to comprehensive courses in engineering calculus, linear algebra, and a tiny bit of probability and statistics is necessary to understand the material herein. To further strengthen one's understanding of the algorithms, it is also useful to code them up, e.g., in Python. Hence, knowledge of a programming language as well as how to use libraries (e.g., numpy, pandas, seaborn, matplotlib, and scikit-learn) is useful.

At the Faculty of Engineering at the National University of Singapore, students are expected to have taken and passed MA1513 *Linear Algebra with Differential Equations*, GER1000H *Quantitative Reasoning* and CS1010E *Programming Methodology*. As and when needed, we will review the necessary mathematical background, but these reviews will be too quick for those who have not been exposed to the prerequisites previously.

Notational Conventions

The number of samples and dimensions of the feature vectors (also known as training samples or training examples) will always be denoted by m and d respectively. The number of classes or clusters will be denoted as c or K. Vectors are denoted by lower case bold letters (e.g., $\mathbf{a}$) or letters with an underline (e.g., $\underline{a}$). Matrices are always denoted by upper bold case letters (e.g., $\mathbf{A}$). By convention, all vectors are column vectors, unless otherwise stated. In particular, training and test samples are *column* vectors in $\mathbb{R}^d$, the d-dimensional real Euclidean space. Training samples are denoted by $\mathbf{x}_i$; labels or targets are denoted as y_i. The j^{th} component of $\mathbf{x}_i$ is denoted as $x_{i,j}$. Sets will, in general, be denoted in calligraphic font (e.g., $\mathcal{A}$). The size (or cardinality or number of elements) of a finite set $\mathcal{A}$ is denoted as $|\mathcal{A}|$. The training dataset

will be denoted as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ or $\mathcal{D}_m$ when we want to make the number of samples m explicit. The transpose and inverse (if it exists) of a matrix $\mathbf{A}$ are denoted by $\mathbf{A}^\top$ and $\mathbf{A}^{-1}$ respectively. The inner product between two vectors $\mathbf{a}$ and $\mathbf{b}$ is $\mathbf{a}^\top\mathbf{b}$ or $\langle \mathbf{a}, \mathbf{b} \rangle$. The l_2 norm of $\mathbf{a}$ is $\|\mathbf{a}\| = \sqrt{\langle \mathbf{a}, \mathbf{a} \rangle} = \sqrt{\sum_i a_i^2}$. Other notation will be introduced as needed within the chapters.

Acknowledgements

I thank Yuta Sakai (Figs. 1.1, 1.2, 1.3, 12.3, and 12.4) and Qiaosheng Zhang (Fig. 2.1) for helping to make some figures in the set of notes. I thank the lecturers of the module Profs. Haizhou Li, Kar Ann Toh, Helen Zhou and Thomas Yeo and fellow tutors Profs. Chen-Khong Tham, Hari Garg and Drs. Rajalaxmi Rajagopalan and Gu Jing for our spirited discussions on the subject—from best pedagogical approaches to how to program in Python efficiently. Finally, I thank the students of T13, T16, and T22 in the Fall of 2020 and T13 and T21 in the Spring of 2021 for pointing out typos, listening to me talk about stories related to machine learning, and for enjoying the subject during these difficult times.

Contents

Chapter 1

Types of Machine Learning Problems

In this chapter, we will summarize three main problems in machine learning.

Vectors (e.g., $\mathbf{x}$) will be denoted by boldface while scalars (e.g., y) will be denoted by san-serif font. The set of all real numbers (resp. natural numbers) is denoted as $\mathbb{R}$ (resp. $\mathbb{N}$).

1.1 Classification

The most canonical task is classification. This problem takes the following form. We have a *dataset* $\mathcal{D}$ which consists of a certain number m of *data examples* $(\mathbf{x}_i, y_i), i = 1, \ldots, m$. Each data example $(\mathbf{x}_i, y_i)$ consists of a *feature vector* (also called *training sample* or *training example*)

$$\mathbf{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,d} \end{bmatrix} \quad \text{or} \quad \mathbf{x}_i = \begin{bmatrix} x_{i,1} & x_{i,2} & \ldots & x_{i,d} \end{bmatrix}^\top, \tag{1.1}$$

(usually an element of d-dimensional Euclidean space $\mathbb{R}^d$) and a *label* y_i. The crux of classification is that the label y_i can only take on *finitely many values* so $y_i \in \{1, 2, \ldots, c\}$ for some integer $c \geq 2$, where c is the number of classes. We do not allow y_i to take on infinitely many values.

The classification problem consists in finding a *classifier* which is a function $f : \mathbb{R}^d \to \{1, 2 \ldots, c\}$ such that it accurately predicts labels given new samples, called *test samples*. This function f is constructed or learned based on the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq m\}$.

A couple of examples will illustrate this point.

- We have records of $m = 100$ emails. Let $d = 2$. Thus, there are two components in each $\mathbf{x}_i$. Furthermore the first component of $\mathbf{x}_i$, denoted as $x_{i,1}$, counts the number of times the word "love" appears. The second component of $\mathbf{x}_i$, denoted as $x_{i,2}$, counts the number of times the word "money" appears. Each label $y_i \in \{0, 1\}$ where $y_i = 1$ means that the i^{th} email is spam while $y_i = 0$ means that the i^{th} email is non-spam. Based on the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq 100\}$ of $m = 100$ emails, we are tasked to learn a classifier $f : \mathbb{R}^2 \to \{0, 1\}$ such that we can accurately predict whether the next email you receive (which is of course not in the training dataset) is spam or not. See Fig. 1.1. In this case in which there are only two classes, we refer to this as *binary* classification.

- Consider an image classification problem. We are given a dataset of size $m = 10^6$, namely, $\mathcal{D} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq 10^6\}$ where each $\mathbf{x}_i$ represents an image that is vectorized into a vector. For example, each image contains $5 \times 5 = 25$ pixels and for the sake of simplicity, each pixel value is 0 or 1. Thus $\mathbf{x}_i \in \{0, 1\}^{5 \times 5} \cong \{0, 1\}^{25}$ and $d = 25$. Of course, we can have more complicated images that have multiple quantization levels (not restricted to binary) and colors as well, but the treatment will

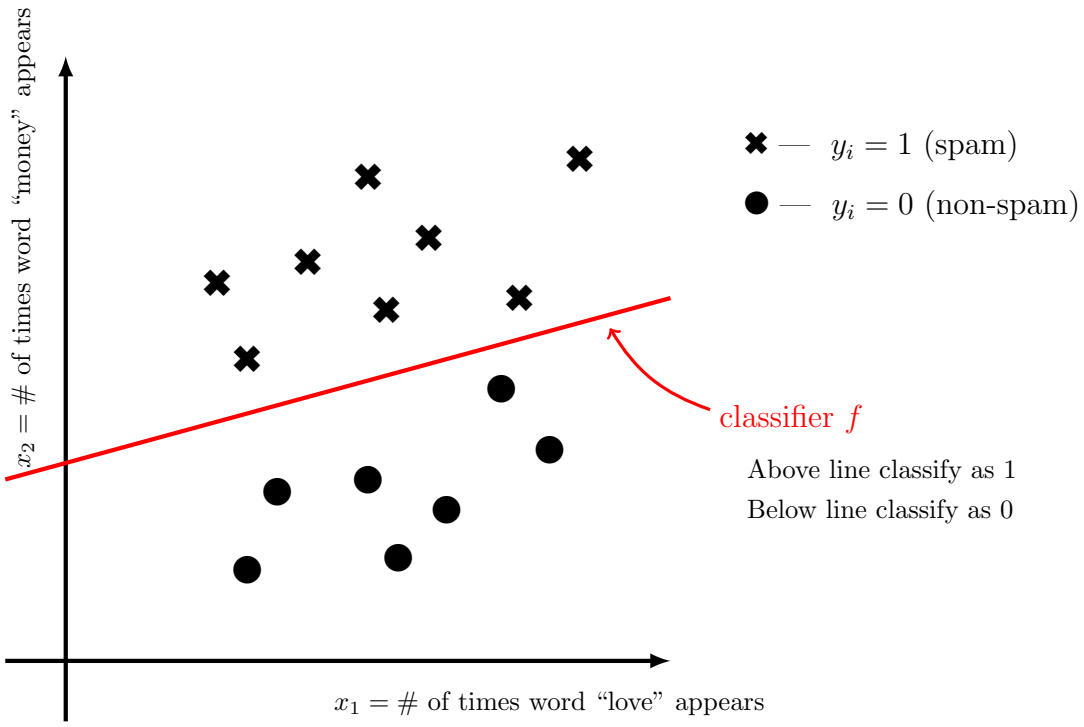


Figure 1.1: Spam classification

be the same. To each image, there is a label y_i which can take on 10 values so $c = 10$. These 10 values signify what object is in the image, e.g., $y_i \in \{\text{dog}, \text{cat}, \ldots, \text{snake}\}$. We would now like to design a image classifier $f : \{0,1\}^{25} \to \{\text{dog}, \text{cat}, \ldots, \text{snake}\}$ that "does well" (in some sense) on new images (or test images) that contain one of the 10 animals. Since there are more than two classes in this scenario, we refer to this as *multi-class* classification.

- Is the following a classification problem? There are a total of $M \geq 2$ football teams. Each football team $i \in \{1, 2, \ldots, M\}$ plays L games against every other football team j. Say there are no draws. The number of times i beats j is denoted as $b_{ij} \in \{0, 1, \ldots, L\}$ so necessarily $b_{ij} + b_{ji} = L$ (convince yourself that this is the case). We set $b_{ii} = 0$ for all i. Clearly, if one team wins all its games against other team, it is the best. When this is not case, which is common, it is not so clear what is the best strategy to *rank* the team. More precisely, given the matrix $\mathbf{B} = [b_{ij}]_{1 \leq i \leq M, 1 \leq j \leq M}$ we would like to rank the teams from top (best) to bottom (worst). This is a non-standard machine learning problem. However, notice that the total number or rankings is $M!$, which is finite, so in some sense this is like a classification problem. In what way is it not a classification problem? See [XFTF19] for some discussion of how we can use machine learning to learn the skills of tennis players.

Classification belongs to the class of *supervised* learning methods.

1.2 Regression

Regression is just like classification except that y_i are no longer restricted to belong to a finite set. Rather it can take on uncountably[1] many values. In the case of regression, we typically do not say that y_i is the label.

[1] A set $\mathcal{A}$ is *countable* if there is a bijection between $\mathcal{A}$ and the set of natural numbers $\mathbb{N}$. A set $\mathcal{A}$ is *uncountable* if its cardinality is strictly larger than that of the natural numbers. Check that the set of rationals is countable. A well-known, but non-trivial, fact (due to Cantor) is that the interval $[0, 1]$ is uncountable.

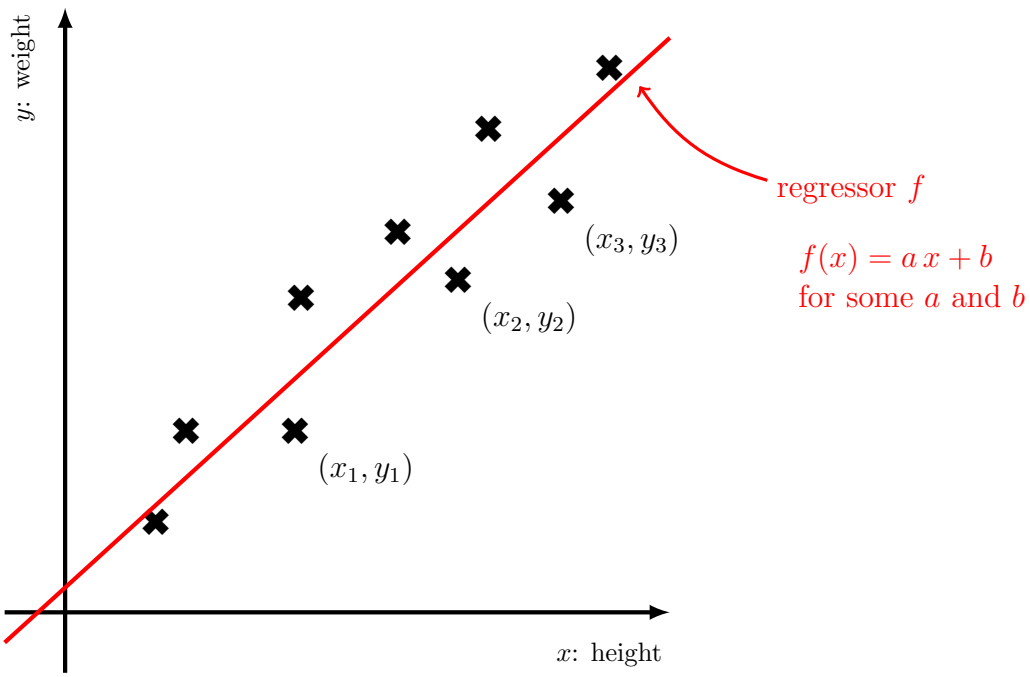


Figure 1.2: Height and weight regression example

Rather we use the term *target variable* or *outcome variable* or *dependent variable* to refer to the y_i's. For instance y_i could take any value in the interval $[0, 1]$ (which is uncountable) or it could take values in $\mathbb{R}$. In either case, the data examples in the dataset $\mathcal{D}$, denoted as $(\mathbf{x}_i, y_i)$ belong to $\mathbb{R}^d \times \mathbb{R}$, which means that each feature vector or training sample $\mathbf{x}_i \in \mathbb{R}^d$ (d-dimensional Euclidean space) and each target variable $y_i \in \mathbb{R}$ is a real number. We wish to learn a *regressor* $f : \mathbb{R}^d \to \mathbb{R}$ which is a function that brings us from the feature space to the set of real numbers. The regressor f should be designed such that given a new feature vector $\mathbf{x}'$, its corresponding target value y' is "well predicted", in some precise mathematical sense, by $f(\mathbf{x}')$.

Again some examples would make this clear.

- Say each feature vector $x_i \in \mathbb{R}$ is scalar (so $d = 1$) and represents the height of a student. The target variable y_i represents the weight of the same student. Given the data of $m = 600$ students in EE2211 captured in the dataset $\mathcal{D} = \{(x_i, y_i) : 1 \leq i \leq 600\}$, I am tasked to learn a regressor $f : \mathbb{R} \to \mathbb{R}$ so that I can use the height of Alice x' in a new class EE9999 to predict her weight $y' = f(x')$. See Fig. 1.2.
- Say each feature vector $\mathbf{x}_i \in \mathbb{R}^3$ is a 3-dimensional vector and $x_{i,1}$ represents the air pressure at location i, $x_{i,2}$ represents the amount of rainfall at location i and $x_{i,3}$ represents the "amount of greenery" at location i. Each y_i corresponds to the temperature at location i. The temperature can take on uncountably many values—it is a real number. Given the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq m\}$, we would like to learn a regressor $f : \mathbb{R}^3 \to \mathbb{R}$ such that if I give you the feature vector of new location $\mathbf{x}'$, you can tell me the temperature $y' = f(\mathbf{x}')$.

Regression also belongs to the class of *supervised* learning methods.

1.3 Clustering

In clustering, the labels or target variables y_i are no longer available and we *only* have access to the feature vectors $\mathcal{D} = \{\mathbf{x}_i : 1 \leq i \leq m\}$. There is often reason to believe that these feature vectors can be grouped or clustered into different clusters.

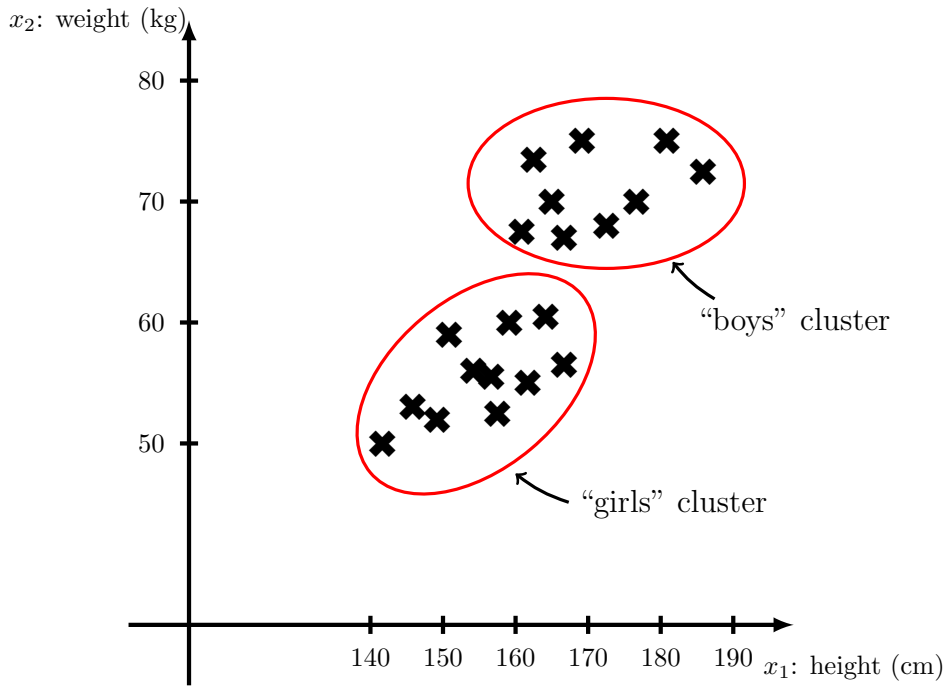


Figure 1.3: Height and weight clustering example

- Say the vector $\mathbf{x}_i = [x_{i,1}, x_{i,2}]^\top$ represents the height and weight of the i^{th} student in this semester's EE2211 cohort. There is reason to believe that girls are shorter and lighter than boys. So given $\mathcal{D} = \{\mathbf{x}_i : 1 \leq i \leq m\}$, can we assign each partition this set into 2 groups automatically? See Fig. 1.3. Note that while the clustering algorithm knows that there are 2 clusters, it does not and cannot assign semantic meanings (such as "boys" and "girls") to the clusters it discovers. It can only tell us that $\mathcal{D}$ is partitioned into two non-empty disjoint subsets $\mathcal{D}_1$ and $\mathcal{D}_2$ (i.e., $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$ and $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}$).
- Say each $\mathbf{x}_i = [x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}]^\top$ and $x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}$ respectively represent the number of times the words "winner", "victory", "stock", "prices" appeared in the i^{th} article in a bag of m news articles. There is reason to believe that sports articles would contain more of the first 2 words but finance articles would contain more of the last 2 words. Hence, it seems possible to design a clustering algorithm to group the m feature vectors into two clusters, one representing sports articles while the other representing finance articles.

Clustering belongs to the class of *unsupervised* learning methods.

1.4 Practice Problems

Exercise 1.1. *Another class of machine learning problems is known as* semi-supervised learning, *which straddles between supervised and unsupervised learning. Here, just as in classification and regression, we are given a labelled dataset* $\mathcal{D}_{\text{l}} = \{(\mathbf{x}_i, y_i) : 1 \leq i \leq m\}$. *In addition, we are given a set of unlabelled data points* $\mathcal{D}_{\text{u}} = \{\tilde{\mathbf{x}}_j : 1 \leq j \leq n\}$ *where typically* n *is much larger than* m.

(i) Provide a real-life example of when semi-supervised learning is applicable.

(ii) Suggest a reasonable procedure to utilize all the (labelled and unlabelled) data points $\mathcal{D}_{\text{l}} \cup \mathcal{D}_{\text{u}}$ *to predict the labels of other new feature vectors or test samples* $\mathbf{x}_{\text{new}}$.

Chapter 2

Types of Data

In this chapter, we will summarize the different types of data you will encounter. We will also discuss normalizing data of different scales (or units).

2.1 Numerical Data

Numerical data, as the name suggests, is data that represents *numbers*. However, simple as it is, there are different types of numerical data we should be aware of.

- Discrete Data: These are data that are distinct and separate and can only take on certain values (usually finitely many values). This type of data can be counted. For example, the "number of heads in 100 coin flips" is discrete numerical data because they can only take on values in the set of 101 values $\{0, 1, 2, \ldots, 99, 100\}$. The "outcomes of a digital thermometer" is also discrete data because the real temperature is necessarily quantized, say to the range $\{35.0, 35.1, 35.2, \ldots, 40.4, 40.5\}$ centigrade. To ascertain whether you're dealing with discrete data, you can ask yourself the following questions: "Can you count it?" and "Can it be divided up in to smaller and smaller parts?". If the first answer is "yes" and the second is "no", you know you're dealing with discrete numerical data. If the discrete data only takes on the two values (e.g., in $\{0, 1\}$), then it is called *binary* data.
- Continuous Data: These are data that can't be counted but they can be measured. The height of a person is a real number. Note that no matter how fine the markings are of a ruler, you can't nail down the height of a person to infinite precision (because your measurements will be "off by a bit"). Similarly, the temperature is a real number; no matter how accurate your thermometer, the reading quantized and so what you get is only an *approximation* (to a certain number of significant digits/decimal points) to the true temperature. Think of it this way, how many decimal points are needed to describe the number $\pi \approx 3.14159\ldots$ or $e \approx 2.71828\ldots$? Infinitely many! This is analogous to the temperature. If the outcome variable is continuous, it makes sense to build a regression model.

Let me show you a real life example in which discrete data is present and we would like to predict discrete data given discrete data. Consider the Netflix problem[1] in which we have n users and m movies. The rating of each user $i \in \{1, 2, \ldots, n\}$ to each movie $j \in \{1, 2, \ldots, m\}$ is a number in the set $\{1, 2, 3, 4, 5\}$ in which 5 means user i very much likes movie j and 1 means s/he does not like it. Thus, the ratings constitute *discrete* data. In the Netflix problem, n is of the order of 10^6 and m is of the order 10^4. Because each user only rates a small subset of movies (s/he does not have enough time to rate all m movies), what we observe is a small subset of the rating matrix $M \in \{1, 2, 3, 4, 5\}^{n \times m}$. The indices of the entries in M that are observed are captured by the set $\Omega = \{(i, j) \in \{1, 2, \ldots, n\} \times \{1, 2, \ldots, m\} : \text{user } i \text{ has rated movie } j\}$. Thus, we observe M_Ω, defined as the entries of matrix we observe. There is also reason to believe that the full rating matrix

[1] Read more about it here: `https://en.wikipedia.org/wiki/Netflix_Prize`

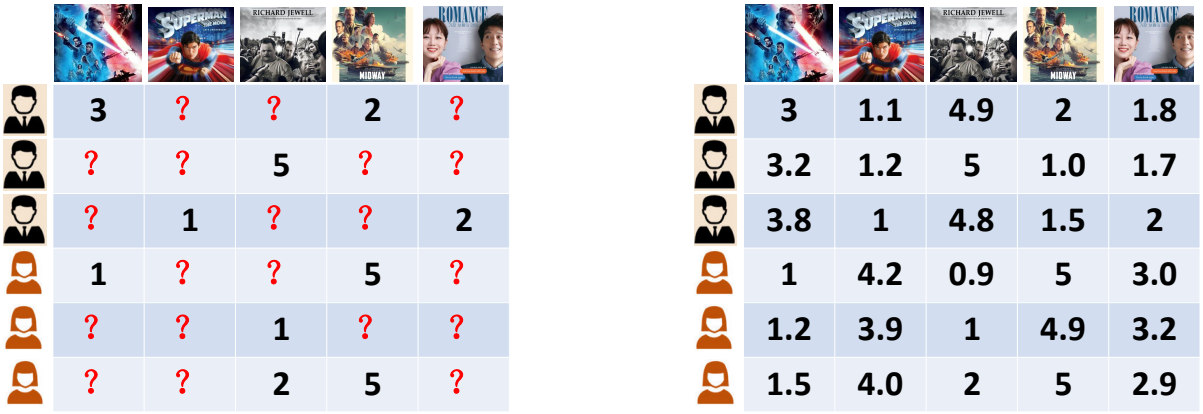


Figure 2.1: We observe the partially filled matrix M_Ω on the left. We would like to use an algorithm to learn the filled (approximately low-rank) matrix on the right. What do you expect the singular value decomposition of the matrix on the right to look like?

Person	Race	Gender
Ravi	Indian	Male
Alice	Eurasian	Female
Li Ting	Chinese	Female
Ali	Malay	Male
Aisha	Malay	Female
Teck Seng	Chinese	Male
Giselle	Eurasian	Female
Jerry	Eurasian	Male

Table 2.1: A small dataset of 8 Singaporeans. "Race" and "gender" are the two features here.

M, which seems large, is low rank because the number of types of movies (column rank) is small (romance, action, documentary) and the number of types of users (row rank) is also small. The fact that M is low rank constitutes side-information for us to exploit to learn $M_{\Omega^c} = \{M_{i,j} : (i,j) \in \Omega^c\}$, the entries of the matrix we do not observe. See Fig. 2.1. One simple formulation that exploit the low rank nature of M is the following convex optimization problem:

$$\hat{M} \in \arg\min \left\{ \|\tilde{M}\|_* : \tilde{M}_{i,j} = M_{i,j}, \forall\, (i,j) \in \Omega \right\} \tag{2.1}$$

where $\|\cdot\|_*$ is the nuclear norm (or sum of singular values), which is a proxy of the rank.[2] Thus, we are seeking a real-valued matrix $\hat{M} \in \mathbb{R}^{n \times m}$ that minimizes the nuclear norm and is consistent with the given entries in M_Ω. There are many algorithms and provable guarantees for solving the problem in (2.1). I am personally interested in this problem.

2.2 Categorical Data

- Nominal data: These data represent discrete units and are used to represent variables that have no natural quantitative value. They are nothing but "labels". For example, consider the dataset in Table 2.1.

  Categorical data represent *characteristics*. Again there are several types but we focus on the two below.

[2]It is known that the nuclear norm $\|\cdot\|_*$ is the convex envelope of the rank function $\text{rank}(\cdot)$ on the unit ball $\{\mathbf{X} : \|\mathbf{X}\|_2 \leq 1\}$.

Person	isChinese	isMalay	isIndian	isEurasian	isMale	isFemale
Ravi	0	0	1	0	1	0
Alice	0	0	0	1	0	1
Li Ting	1	0	0	0	0	1
Ali	0	1	0	0	1	0
Aisha	0	1	0	0	0	1
Teck Seng	1	0	0	0	1	0
Giselle	0	0	0	1	0	1
Jerry	0	0	0	1	1	0

Table 2.2: A small dataset of 8 Singaporeans after one-hot encoding

We see that there are four categories for race $\{\text{Chinese}, \text{Indian}, \text{Malay}, \text{Eurasian}\}$ and two categories for gender $\{\text{Male}, \text{Female}\}$. So the features "race" and "gender" are categorical nominal data. To deal with categorical variables within a machine learning framework (which is more amenable to numerical data), we may use *one-hot encoding*, converting the above table/dataset to Table 2.2. Is the last column of Table 2.2 necessary for subsequent data analysis?

In Python, you can do one-hot encoding by using LabelEncoder and OneHotEncoding. Please play with the following code and use it to do one-hot encoding of the two nominal variables in Table 2.1.

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

# define example
data = ['cold', 'cold', 'warm', 'cold', 'hot', 'hot', 'warm', 'cold', 'warm', 'hot']
values = array(data)
print(values)

# integer encode
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
print(integer_encoded)

# binary encode
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)

# invert first example
inverted = label_encoder.inverse_transform([argmax(onehot_encoded[0, :])])
print(inverted)
```

- Ordinal data: These data represent discrete and *ordered* units. It is therefore nearly the same as nominal data, except that its ordering matters. For example, the set of all possible educational backgrounds in a survey form could be

$$\{\text{primary}, \text{secondary}, \text{pre-university}, \text{university}, \text{postgraduate}\}.$$

One can see that this values of this feature are *ordered* – hence the term *ordinal*. Indeed, the "difference" between primary and secondary is "smaller than" primary and postgraduate. This is the main limitation of ordinal data – the differences between the values are not really known. Because of that, ordinal scales (e.g., integers) are usually used to measure non-numeric features like happiness, customer satisfaction and so on. Thus when you rate your Grab driver, you rate her/him on a scale of 1 to 5 (likened to discrete numerical data), which is more quantitative, and amenable to machine learning, than

$$\{\text{very unsatisfactory}, \text{unsatisfactory}, \text{satisfactory}, \text{good}, \text{excellent}\}.$$

2.3 Normalizing Raw Data

Often we have feature vectors in which features are on different scales. We let m be the number of training samples and d be the number of features as usual. Say the feature vectors are

$$\mathbf{x}_1 = \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,d} \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,d} \end{bmatrix}, \quad \ldots \quad \text{and} \quad \mathbf{x}_m = \begin{bmatrix} x_{m,1} \\ x_{m,2} \\ \vdots \\ x_{m,d} \end{bmatrix}. \tag{2.2}$$

The first feature $x_{i,1}$ for $1 \leq i \leq m$ could be the height of students measured in centimeters so the dynamic range is $[x_{\min,1}, x_{\max,1}] = [140, 190]$. The second feature $x_{i,2}$ for $1 \leq i \leq m$ could measure the (American) shoe size of the students so it ranges from $[x_{\min,2}, x_{\max,2}] = [6, 13]$. So even if both features are deemed equally "important", unfortunately, any machine learning method would place more importance on the first feature because of its larger values, which is not ideal. Thus, we have to scale or normalize the features so that their dynamic ranges are roughly the same. We can use (at least) two methods to do so:

- Z-score scaling: First we calculate the empirical mean and empirical standard deviation of each feature, say the i^{th}. These are

$$\hat{x}_1 = \frac{1}{m}\sum_{i=1}^{m} x_{i,1}, \quad \text{and} \quad \hat{\sigma}_1 = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(x_{i,1} - \hat{x}_1)^2}. \tag{2.3}$$

  Then we create the normalized 1^{st} features associated to each training sample as

$$\bar{x}_{i,1} := \frac{x_{i,1} - \hat{x}_1}{\hat{\sigma}_1}. \tag{2.4}$$

  We can do this for all features so that, in some sense, they are all "normalized". What is the empirical and empirical standard of the set of numbers $\{\bar{x}_{i,1} : 1 \leq i \leq m\}$ now? Why do we divide by $m-1$ in the second equation in (2.3)? This is not so easy to answer actually. Can we divide by m? Read up on unbiased estimator of (population) variance or look at the notes for Lecture 3. Is the estimator for standard deviation $\hat{\sigma}_1$ or the estimator for variance $\hat{\sigma}_1^2$ unbiased? Can you prove it and what assumptions do we need to make? What happens as $m \to \infty$? The reason why this method of normalizing is called "Z-score" is because the standard normal random variable with density $x \in \mathbb{R} \mapsto \frac{1}{\sqrt{2\pi}} \exp\big(-x^2/2\big)$ is usually denoted as $Z \sim \mathcal{N}(0,1)$.

  You can try this code for Question 1 in Tutorial 2 as follows.

```
av_ExpenditureList = statistics.mean(expenditureList)
sd_ExpenditureList = statistics.stdev(expenditureList)

subtraction = list(np.array(expenditureList) - np.array(av_ExpenditureList))
norm_ExpenditureList = list(np.array(subtraction)/np.array(sd_ExpenditureList))
print(norm_ExpenditureList)
```

Alternatively, you can use `StandardScaler` in `sklearn.preprocessing` to do this "automatically". See the documentation.

- Min-Max scaling: Define the minimum and maximum values of feature 1 to be

$$x_{\min,1} := \min_{1\le i\le m} x_{i,1}, \quad \text{and} \quad x_{\max,1} := \max_{1\le i\le m} x_{i,1}. \tag{2.5}$$

Then we create the normalized 1^{st} features associated to each training sample as

$$\bar{x}_{i,1} := \frac{x_{i,1} - x_{\min,1}}{x_{\max,1} - x_{\min,1}}. \tag{2.6}$$

Now what are the minimum and maximum of $\{\bar{x}_{i,1} : 1 \le i \le m\}$? Create your own code to verify this. You can use `MinMaxScaler` in `sklearn.preprocessing` to implement this instead of doing it manually. See the documentation.

2.4 Practice Problems

Exercise 2.1. *This problem pertains to data normalization discussed in Section 2.3.*

(i) Find the empirical mean and the empirical (unbiased) standard deviation of the set of numbers $\{\bar{x}_{i,1} : 1 \le i \le m\}$ *defined in* (2.4)*; that is, find*

$$\hat{\bar{x}}_1 = \frac{1}{m}\sum_{i=1}^{m} \bar{x}_{i,1}, \quad \textit{and} \quad \hat{\bar{\sigma}}_1 = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m} (\bar{x}_{i,1} - \hat{\bar{x}}_1)^2}. \tag{2.7}$$

(ii) Find the maximum and minimum of the set of numbers $\{\bar{x}_{i,1} : 1 \le i \le m\}$ *defined in* (2.6).

Chapter 3

Probability and Estimation

In this chapter, we will summarize some key concepts from probability and provide some examples of maximum likelihood estimation. The topics of probability, statistics, and estimation theory are so rich and vast that a few pages will not do justice to them. Nevertheless, we will try.

A word about notation we adopt in this chapter. Random variables and the values that they take on will be denoted in upper (e.g., X) and lower case (e.g., x, x_i, x) respectively. Sets (in which the random variables assume their values) will be denoted by calligraphic font (e.g., $\mathcal{X}$). The probability, expectation and variance operators are respectively denoted by $\Pr(\,\cdot\,)$, $\mathbb{E}[\,\cdot\,]$ and $\mathrm{Var}(\,\cdot\,)$.

3.1 Basic Probability Theory

Because all of modern statistical machine learning deals with uncertainty, it seems appropriate to start of by reminding ourselves of the definitions of events, probabilities, joint probabilities and conditional probabilities. For more details, the reader is encouraged to consult any standard probablity textbook such as those by Bertsekas and Tsitsiklis [BT02] or Ross [Ros12].

Let us motivate probability and statistical (Bayesian) inference by considering an example from Bishop's machine learning book [Bis08].

Example 3.1. *There is a red box and a blue box. In the red box there are a total of 8 fruits, 2 apples and 6 oranges. In the blue box, there are a total of 4 fruits, 3 apples and 1 orange. The probability of selecting the red box is* $\Pr(B = \mathsf{r}) = 2/5$ *and probability of selecting the blue box is* $\Pr(B = \mathsf{b}) = 3/5$*. Having selected a box, selecting any item within the box is equally likely. Some of the questions we would like to ask include:*

- *What's the probability that we select an orange?*
- *Given that we've selected an orange, what's the probability that we chose it from the blue box?*

3.1.1 Joint and Conditional Probabilities for Discrete Random Variables

Now, let us consider a more general example involving two random variables X and Y. Suppose, as in the example above, the random variables are only permitted to take on *finitely many* values. So X can only take on values in the finite set $\mathcal{X} = \{x_1, \ldots, x_M\}$ and Y takes on values in $\mathcal{Y} = \{y_1, \ldots, y_L\}$. Such random variables are known as *discrete* random variables (in which no mathematical peculiarities arise). It should be clear that

$$\sum_{i=1}^{M} \Pr(X = x_i) = 1. \tag{3.1}$$

Consider m trials (of sampling X and Y) and let the number of trials for which $X = x_i$ and $Y = y_j$ be m_{ij}. Then, if m is large, we can assume that

$$\Pr(X = x_i, Y = y_j) = \frac{m_{ij}}{m}. \tag{3.2}$$

The argument of $\Pr(\,\cdot\,)$ is known as an *event*. Roughly speaking, an event is a set in which a random variable (or multiple random variables) assume some value(s) in some set, e.g., $\{X = x_i, Y = y_j\}$. What's the probability that $X = x_i$? Well, we simply sum up those m_{ij}'s for which the first index equals to i. In other words,

$$\Pr(X = x_i) = \sum_{j=1}^{L} \frac{m_{ij}}{m} =: \frac{c_i}{m}, \tag{3.3}$$

where $c_i := \sum_{j=1}^{L} m_{ij}$. Expressed slightly differently, we have

$$\Pr(X = x_i) = \sum_{j=1}^{L} \Pr(X = x_i, Y = y_j). \tag{3.4}$$

This is the important *sum rule* in probability and will be used extensively in statistical machine learning so the reader is urged to internalize this. Similarly, the marginal probability that $Y = y_j$ is

$$\Pr(Y = y_j) = \sum_{i=1}^{M} \Pr(X = x_i, Y = y_j). \tag{3.5}$$

Now, we introduce the important notion of conditional probabilities. Given that $X = x_i$, what is the probability that $Y = y_j$? Clearly,

$$\Pr(Y = y_j \mid X = x_i) = \frac{m_{ij}}{c_i}. \tag{3.6}$$

But note also that

$$\Pr(X = x_i, Y = y_j) = \frac{m_{ij}}{m} = \frac{m_{ij}}{c_i} \cdot \frac{c_i}{m} = \Pr(Y = y_j \mid X = x_i) \Pr(X = x_i). \tag{3.7}$$

We have derived the important *product-rule* in probability. The (two basic) rules of probability are summarized as follows:

$$\Pr(X = x) = \sum_{y \in \mathcal{Y}} \Pr(X = x, Y = y), \tag{3.8}$$

$$\Pr(X = x, Y = y) = \Pr(Y = y \mid X = x) \Pr(X = x). \tag{3.9}$$

A note about notation. We will usually write $p_X(x) := \Pr(X = x)$ or simply denote this function as $p(x)$ when the random variable is clear from the context. The function $p(x)$ is known as the *probability mass function* or pmf and it satisfies

$$p(x) \geq 0 \quad \forall\, x \in \mathcal{X} \qquad \text{and} \qquad \sum_{x \in \mathcal{X}} p(x) = 1. \tag{3.10}$$

You will also often hear the colloquial term "distribution". This is often used synonymously with "pmf". Similarly, the joint pmf of random variables X and Y is denoted as $p(x, y) := \Pr(X = x, Y = y)$. Finally, the conditional will be denoted as $p(y \mid x) := \Pr(Y = y \mid X = x)$.

By combining the sum rule in (3.8) and the product rule in (3.9), we can derive Bayes' rule:

$$p(y \mid x) = \frac{p(x,y)}{p(x)} = \frac{p(x \mid y)p(y)}{p(x)} = \frac{p(x \mid y)p(y)}{\sum_{y' \in \mathcal{Y}} p(x \mid y')p(y')}. \tag{3.11}$$

This is a central relationship in pattern recognition, machine learning and statistical physics. Note that we have "inverted the causal relationship" between X and Y. On the left, Y "depends on" X while on the right, we have expressed the same causality relationship in terms of the causal dependence of X on Y.

Bayes' theorem can be written alternatively as follows:

$$p(x \mid y) \propto p(y \mid x)p(x), \tag{3.12}$$

where $\propto$ denotes equality up to a constant (not depending on x). If x designates an unknown variable, something we would like to infer and $p(x)$ is its *prior probability*, then $p(x \mid y)$ denotes the *posterior probability*, the belief we have about x *after*[1] we know that $Y = y$. In the parlance of statistical inference, Bayes' rule in (3.11) can be written as

$$\text{posterior } = \frac{\text{likelihood } \times \text{prior}}{\text{model evidence}} \propto \text{likelihood } \times \text{prior }. \tag{3.13}$$

We can now use this relation to solve Exercise 3.1 that is based on Example 3.1. Once you solve it, you see that if F is the fruit chosen, $\Pr(F = \mathsf{o}) = 9/20$ and $\Pr(B = \mathsf{b} \mid F = \mathsf{o}) = 1/3$.

Note the following: Prior to having any additional information about what fruit we chose, the *prior probability* of choosing from a blue box is $\Pr(B = \mathsf{b}) = 3/5$. However, if we know the identity of the fruit we chose, say orange, then the *posterior probability* of choosing from the blue box is $\Pr(B = \mathsf{b} \mid F = \mathsf{o}) = 1/3$. This is the simplest non-trivial example of statistical inference. Intuitively, this is true because the blue box contains far fewer oranges so knowing that we chose an orange biases our *belief* about the box we chose from.

3.1.2 Continuous Random Variables

We now segue into the land of continuous random variables. To introduce continuous random variables formally, we would require too much mathematical machinery that goes beyond the scope of the class; see [Ros12] for a gentle introduction. However, very roughly speaking, X is said to be a *continuous random variable* if its *cumulative distribution function* (cdf) $x \in \mathbb{R} \mapsto F_X(x) = \Pr(X \leq x)$ is differentiable on $\mathbb{R}$.[2] Its derivative is then called the *probability density function* (pdf) of X and written as

$$f_X(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_X(x), \quad x \in \mathbb{R}. \tag{3.14}$$

Any pdf has the following properties (why?):

$$f_X(x) \geq 0 \quad \forall\, x \in \mathbb{R} \qquad \text{and} \qquad \int_{\mathbb{R}} f_X(x)\,\mathrm{d}x = 1. \tag{3.15}$$

Now, it is easy to see (cf. the second fundamental theorem of calculus) that for any $-\infty < a \leq b < \infty$,

$$\Pr(a < X \leq b) = F_X(b) - F_X(a) = \int_a^b f_X(x)\,\mathrm{d}x. \tag{3.16}$$

We can define the *joint density* (or *joint probability density function*) of a pair of continuous random variables as

$$f_{X,Y}(x,y) = \frac{\partial^2}{\partial x\,\partial y} F_{X,Y}(x,y), \quad (x,y) \in \mathbb{R}^2. \tag{3.17}$$

Conditional probability and independence can be defined analogously to the discrete case. Clearly,

$$\Pr(a < X \leq b, c < Y \leq d) = \int_c^d \int_a^b f_{X,Y}(x,y)\,\mathrm{d}x\,\mathrm{d}y. \tag{3.18}$$

[1] The word "posterior" is derived from the word "post", which means "after" (e.g., post-midnight means after midnight).

[2] Roughly speaking, a function $g : I \to \mathbb{R}$ defined on an interval $I \subset \mathbb{R}$ is said to be *differentiable at* $x \in I$ if the limit $\lim_{\epsilon \to 0}(g(x+\epsilon) - g(x))/\epsilon$ exists. In addition, $g : I \to \mathbb{R}$ is said to be *differentiable on* I if it is differentiable at each $x \in I$.

3.1.3 Independence

What does it mean for two random variables X and Y to be independent? Roughly speaking, knowledge of one does not influence our knowledge of the other. More precisely, this can be expressed in a variety of ways. Two random variables X and Y are *independent* if their joint distribution $p_{X,Y}(x,y) := \Pr(X = x, Y = y)$ factorizes, i.e.,

$$p_{X,Y}(x,y) = p_X(x)p_Y(y), \tag{3.19}$$

for every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. Note from (3.19) that if X and Y are independent if

$$p_{X|Y}(x \mid y) = p_X(x) \tag{3.20}$$

for every $x \in \mathcal{X}$ and *every* $y \in \mathcal{Y}$ *such that* $p_Y(y) > 0$ (what if $p_Y(y) = 0$?). Intuitively, (3.20) means that knowledge that $Y = y$ tells you no additional information about X.

For example, I toss a coin $2n$ times and all tosses are mutually independent. Let X be the number of heads that I observe in the first n coin tosses and let Y be the number of heads that I observe in the second n coin tosses. Since X and Y are the result of independent coin tosses, the two random variables X and Y are independent. Consider another example. Let X and Y denote respectively, the presence of rain in Jurong and in Clementi. These random variables are clearly not independent because if I know that there is rain in Jurong, due to the proximity of the two locations, it is also likely that there is rain in Clementi. So knowledge of X *does* provide some information about Y.

3.1.4 Expectation and Variance

The *expectation* of a discrete random variable X with pmf p_X is defined to be

$$\mathbb{E}[X] = \sum_{x \in \mathcal{X}} x p_X(x). \tag{3.21}$$

If X is a continuous random variable with pdf f_X, we have

$$\mathbb{E}[X] = \int_{\mathbb{R}} x f_X(x)\,\mathrm{d}x. \tag{3.22}$$

Note that the expectation is a statistical summary of the *distribution* of X, rather than depending on the realized value of X.

If g is a function from the domain of X to $\mathbb{R}$, we can obtain the expectation of $Y = g(X)$ in the same way. It can be shown that

$$\mathbb{E}[Y] = \mathbb{E}[g(X)] = \int_{\mathbb{R}} y f_Y(y)\,\mathrm{d}y = \int_{\mathbb{R}} g(x) f_X(x)\,\mathrm{d}x. \tag{3.23}$$

In particular if $g(X) = aX + b$ (for constants $a, b \in \mathbb{R}$), then $\mathbb{E}[g(X)] = a\mathbb{E}[X] + b = g(\mathbb{E}[X])$. This fact is known as the *linearity of expectation*.

The *variance* of X is the expectation of $g(X) = (X - \mathbb{E}[X])^2$, a particular function of X. Thus,

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \sum_{x \in \mathcal{X}} (x - \mathbb{E}[X])^2 p_X(x) \qquad \text{(discrete rvs)}, \tag{3.24}$$

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \int_{\mathbb{R}} (x - \mathbb{E}[X])^2 f_X(x)\,\mathrm{d}x \qquad \text{(continuous rvs)}. \tag{3.25}$$

3.2 Maximum Likelihood Estimation

In machine learning, we almost always do not have access to the underlying distributions that the data are generated from. Rather, we have access to *sample* data and we would like to use it to estimate some

parameters. For example, in class, you saw that if $\mathcal{D} = \{X_1, \ldots, X_m\}$ are independent samples from the univariate Gaussian distribution

$$f_X(x; \mu, \sigma^2) = \mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2\sigma^2}(x-\mu)^2 \Big) \quad x \in \mathbb{R}, \tag{3.26}$$

then we can use the samples in $\mathcal{D}$ to estimate the parameter vector $\theta = (\mu, \sigma^2)$. Note that we use the notation $f_X(x; \mu, \sigma^2)$ or $f_X(x; \theta)$ to emphasize that the density is *parametrized* by the parameters (μ, σ^2) or θ.[3] We obtained the maximum likelihood estimates for the mean and variance as

$$\hat{\mu}_{\text{ML}} = \frac{1}{m}\sum_{i=1}^{m} X_i \quad \text{and} \quad \hat{\sigma}^2_{\text{ML}} = \frac{1}{m}\sum_{i=1}^{m}(X_i - \hat{\mu})^2. \tag{3.27}$$

In the following, I would like to expand on this point by providing you with a couple more examples.

Example 3.2 (Bernoulli Distribution)**.** *Let us say that the samples in $\mathcal{D} = \{X_1, \ldots, X_m\}$ are generated independently from the Bernoulli (coin toss) distribution*

$$p_X(x; \theta) = \begin{cases} 1-\theta & x = 0 \\ \theta & x = 1 \end{cases}. \tag{3.28}$$

It would be convenient to write this as

$$p_X(x; \theta) = (1-\theta)^{1-x}\theta^x, \quad x \in \{0, 1\}. \tag{3.29}$$

Check that this is true. The mean of the distribution $\mathbb{E}[X]$ is clearly $\theta \in (0, 1)$ (check). How would we estimate θ from samples? Consider,

$$\hat{\theta}_{\text{ML}} = \arg\max_{\theta \in (0,1)} \quad \prod_{i=1}^{m} p_X(X_i; \theta) \tag{3.30}$$

$$= \arg\max_{\theta \in (0,1)} \quad \sum_{i=1}^{m} \log p_X(X_i; \theta) \tag{3.31}$$

$$= \arg\max_{\theta \in (0,1)} \quad \sum_{i=1}^{m} [(1 - X_i)\log(1-\theta) + X_i \log\theta] \tag{3.32}$$

$$= \arg\max_{\theta \in (0,1)} \quad (m - M_1)\log(1-\theta) + M_1 \log\theta \tag{3.33}$$

where we have used $M_1 := \sum_{i=1}^{m} X_i$ to denote the total number of ones in $\mathcal{D}$. Since the objective function is strictly concave, differentiating and setting to zero yields the (unique) maximum which is

$$-\frac{m - M_1}{1 - \hat{\theta}_{\text{ML}}} + \frac{M_1}{\hat{\theta}_{\text{ML}}} = 0 \qquad \Longrightarrow \qquad \hat{\theta}_{\text{ML}} = \frac{M_1}{m}. \tag{3.34}$$

So the mean θ is estimated by using the empirical mean M_1/m, which agrees with common sense!

Example 3.3 (Exponential Distribution)**.** *Now we consider an example involving continuous random variables. Let us say that the samples in $\mathcal{D} = \{X_1, \ldots, X_m\}$ are generated independently from the exponential distribution (this distribution models waiting times for buses)*

$$f_X(x; \theta) = \begin{cases} \theta \exp(-\theta x) & x \geq 0 \\ 0 & x < 0 \end{cases}. \tag{3.35}$$

[3] Contrast this to the notation $f_{X|Y}(x \mid y)$ in which Y is *random* (and takes on the value y). When we use the semicolon in $f_X(x; \theta)$, the parameter θ is not random; rather it is *deterministic* but *unknown*.

Here, $\theta > 0$ is the unknown (rate) parameter. In fact, $1/\theta = \mathbb{E}[X]$ is the mean of the exponential distribution. Check this by integration by parts. How would we estimate θ from samples? Consider,

$$\hat{\theta}_{\mathrm{ML}} = \arg\max_{\theta>0} \quad \prod_{i=1}^{m} f_X(X_i;\theta) \tag{3.36}$$

$$= \arg\max_{\theta>0} \quad \sum_{i=1}^{m} \log f_X(X_i;\theta) \tag{3.37}$$

$$= \arg\max_{\theta>0} \quad \sum_{i=1}^{m} (\log\theta - \theta X_i) \tag{3.38}$$

$$= \arg\max_{\theta>0} \quad m\log\theta - \theta\sum_{i=1}^{m} X_i. \tag{3.39}$$

Since the objective function is strictly concave, differentiating and setting to zero yields the (unique) maximum which is

$$\frac{m}{\hat{\theta}_{\mathrm{ML}}} = \sum_{i=1}^{m} X_i \qquad \Longrightarrow \qquad \hat{\theta}_{\mathrm{ML}} = \frac{m}{\sum_{i=1}^{n} X_i} = \left(\frac{1}{m}\sum_{i=1}^{m} X_i\right)^{-1}. \tag{3.40}$$

Notice that the result makes sense because $\frac{1}{m}\sum_{i=1}^{m} X_i$ is the empirical mean. Indeed, since θ is the reciprocal of the mean $\mathbb{E}[X]$, it seems plausible that $m/(\sum_{i=1}^{m} X_i)$ is a "good" estimate of θ. Why would it not be "good"? Hint: What happens to $\mathbb{E}[\hat{\theta}_{\mathrm{ML}}]$ when m is small?

3.3 Practice Problems

Exercise 3.1. *Let F be the random variable denoting the fruit chosen. Using the sum, product and Bayes' rules, show that the answers to the questions in Example 3.1 are $\Pr(F = \mathsf{o}) = 9/20$ and $\Pr(B = \mathsf{b} \mid F = \mathsf{o}) = 1/3$ respectively.*

Exercise 3.2. *Flip a fair coin four times. Let X be the number of Heads obtained, and let Y be the position of the first Heads i.e. if the sequence of coin flips is TTHT, then $Y = 3$, if it is THHH, then $Y = 2$. If there are no heads in the four tosses, then we define $Y = 0$.*

(i) Find the joint probability mass function (pmf) of X and Y;

(ii) Using the joint pmf, find the marginal pmf of X.

Exercise 3.3. *Check from the definitions of the variance in (3.24) and (3.25) that the variance can also be expressed as*

$$\mathrm{Var}(X) = \mathbb{E}[X^2] - \big(\mathbb{E}[X]\big)^2. \tag{3.41}$$

When is the variance of a random variable identically equal to 0?

Exercise 3.4. *Assume that the samples in $\mathcal{D} = \{X_1, \ldots, X_m\}$ are sampled independently from the Geometric distribution*

$$p_X(x;\theta) = (1-\theta)^x\theta, \quad x = 1, 2, 3, \ldots \tag{3.42}$$

Find the maximum likelihood estimate $\hat{\theta}_{\mathrm{ML}}$ of θ.

Exercise 3.5. *Assume that the samples in $\mathcal{D} = \{X_1, \ldots, X_m\}$ are sampled independently from the Poisson distribution*

$$p_X(x;\theta) = \frac{e^{-\theta}\theta^x}{x!}, \quad x = 0, 1, 2, \ldots \tag{3.43}$$

Find the maximum likelihood estimate $\hat{\theta}_{\mathrm{ML}}$ of θ.

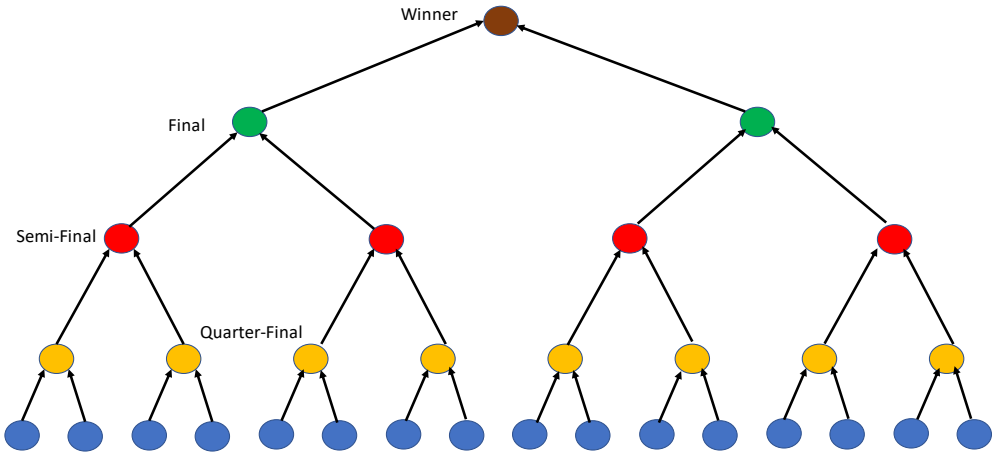


Figure 3.1: Figure for Exercise 3.7 for 16 teams

Exercise 3.6 (Challenging)**.** *Assume that the samples in* $\mathcal{D} = \{X_1, \ldots, X_m\}$ *are sampled independently from the uniform distribution*

$$f_X(x;\theta) = \begin{cases} 1/\theta & 0 \leq x \leq \theta \\ 0 & else \end{cases} . \tag{3.44}$$

Find the maximum likelihood estimate $\hat{\theta}_{\mathrm{ML}}$ *of* θ*. Discuss the (peculiar) properties of the estimate.*

Exercise 3.7 (Challenging)**.** *In the knockout phase of a football tournament, there are* 16 *teams* of equal skill *that compete in an elimination tournament. This proceeds in a number of rounds in which teams compete in pairs; any losing team retires from the tournament. See Fig. 3.1 for an illustration. What is the probability that two given teams will compete against each other? Generalize your answer to* 2^k *teams where* k *is an arbitrary positive integer.*

The following argument is wrong but the answer is right. There has to be 15 *games to knock out all but the ultimate winner. There are* $\binom{16}{2}$ *possible pairs, so that the probability of a given pair being selected for a particular match is* $1/\binom{16}{2} = 1/(8 \cdot 15)$*. Since the selection of the teams in the different matches is mutually exclusive, the probability of a given pair being selected is* 15 *times this, which is* $1/8$*. Why is this wrong and what's the correct way of doing it?*

This problem is taken from Problem 297 of Five Hundred Mathematical Challenges *(Mathematical Association of America, 1996).*

Chapter 4

Systems of Linear Equations

In this chapter, we summarize some key concepts involving the solution set of a system of linear equations. We will first review some basics of linear algebra. Most of the material here can be found in standard linear algebra texts such as Strang [Str16].

4.1 Review of Linear Algebra

Here we review some linear algebra which you should have seen before.

Definition 4.1. *A* vector space *over the reals consists of a set $\mathcal{V}$, a vector sum operation $+ : \mathcal{V} \times \mathcal{V} \to \mathcal{V}$ and a scalar multiplication operation $\cdot : \mathbb{R} \times \mathcal{V} \to \mathcal{V}$ satisfying the following properties.*

- *Commutativity: $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{V}$;*
- *Associativity: $(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$ for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{V}$;*
- *Identity element of addition: $\mathbf{x} + \mathbf{0} = \mathbf{x}$ for all $\mathbf{x} \in \mathcal{V}$;*
- *Inverse element of addition: For every $\mathbf{x} \in \mathcal{V}$, there exists an element $-\mathbf{x} \in \mathcal{V}$ such that $\mathbf{x} + (-\mathbf{x}) = 0$;*
- *Associativity of scalar multiplication: For all $a, b \in \mathbb{R}$ and $\mathbf{x} \in \mathcal{V}$, $a(b\mathbf{x}) = (ab)\mathbf{x}$;*
- *Identity element of scalar multiplication: $1\mathbf{x} = \mathbf{x}$ for $\mathbf{x} \in \mathcal{V}$.*
- *Distributivity of scalar multiplication with respect to vector addition: $a(\mathbf{x} + \mathbf{y}) = a\mathbf{x} + a\mathbf{y}$ for all $a \in \mathbb{R}$ and $\mathbf{x}, \mathbf{y} \in \mathcal{V}$;*
- *Distributivity of scalar multiplication with respect to addition in $\mathbb{R}$: $(a + b)\mathbf{x} = a\mathbf{x} + b\mathbf{x}$ for all $a, b \in \mathbb{R}$ and $\mathbf{x} \in \mathcal{V}$.*

Definition 4.2. *A set of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ from a vector space $\mathcal{V}$ is* linearly independent *if*

$$\beta_1 \mathbf{x}_1 + \beta_2 \mathbf{x}_2 + \ldots + \beta_k \mathbf{x}_k = \mathbf{0} \tag{4.1}$$

implies that $\beta_1 = \beta_2 = \ldots = \beta_k = 0$.

In Exercise 4.1, you will show that $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ being linearly independent is equivalent to the fact that no vector $\mathbf{x}_i$ can be expressed as a linear combination of the other vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_k$.

To illustrate the above definition, let us consider a couple of examples in which $\mathcal{V} = \mathbb{R}^2$.

Example 4.1. *Consider the vectors* $\mathbf{x}_1 = [1,1]^\top$ *and* $\mathbf{x}_2 = [-3,-3]^\top$. *We claim that these vectors are not linearly independent. This is because if we form the equation* $\beta_1\mathbf{x}_1 + \beta_2\mathbf{x}_2 = \mathbf{0}$, *we can find non-zero* β_1, β_2 *such that this equation holds true. In particular, check that* $(\beta_1, \beta_2) = (3, 1)$ *does the job. Thus, according to the contrapositive of Definition 4.2,* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *are not linearly independent. In fact, for all* $t \neq 0$, $(\beta_1, \beta_2) = (3t, t)$ *does the job.*

Example 4.2. *Now consider the vectors* $\mathbf{x}_1 = [1,1]^\top$ *and* $\mathbf{x}_2 = [-3,2]^\top$. *We claim that these vectors are linearly independent. This is because if we form the equation* $\beta_1\mathbf{x}_1 + \beta_2\mathbf{x}_2 = \mathbf{0}$, *we get the equations* $\beta_1 - 3\beta_2 = 0$ *and* $\beta_1 + 2\beta_2 = 0$. *Convince yourself that the only way that these two equations can hold simultaneously is if* $(\beta_1, \beta_2) = (0, 0)$. *Thus, according to Definition 4.2, the vectors are linearly independent.*

Definition 4.3. *A set of vectors* $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ *is a* basis *for a vector space* $\mathcal{V}$ *if*

- $\mathcal{V} = \text{span}\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$;
- $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ *is a linearly independent set of vectors.*

Equivalently, every $\mathbf{x} \in \mathcal{V}$ can be *uniquely* written as $\sum_{i=1}^k \beta_i \mathbf{x}_i$ for some $\{\beta_i\}_{i=1}^k \subset \mathbb{R}$. The number of vectors in any basis of $\mathcal{V}$ is called the *dimension* of $\mathcal{V}$, written as $\dim(\mathcal{V})$.

Definition 4.4. *The* nullspace *(also called* kernel*) of a matrix* $\mathbf{A} \in \mathbb{R}^{m\times d}$ *is defined as*

$$\mathcal{N}(\mathbf{A}) := \{\mathbf{x} \in \mathbb{R}^d : \mathbf{A}\mathbf{x} = \mathbf{0}\}. \tag{4.2}$$

The range *or* column space *of* $\mathbf{A}$ *is defined as*

$$\mathcal{R}(\mathbf{A}) := \{\mathbf{A}\mathbf{x} : \mathbf{x} \in \mathbb{R}^d\} \subset \mathbb{R}^m. \tag{4.3}$$

Definition 4.5. *The* column rank *or* rank *of a matrix* $\mathbf{A} \in \mathbb{R}^{m\times d}$ *is defined as*

$$\text{rank}(\mathbf{A}) = \dim(\mathcal{R}(\mathbf{A})). \tag{4.4}$$

In other words, the column rank of $\mathbf{A}$ *is the dimension of the column space of* $\mathbf{A}$.

The *rank-nullity theorem* says that

$$\text{rank}(\mathbf{A}) + \dim(\mathcal{N}(\mathbf{A})) = d. \tag{4.5}$$

It is always true that $\text{rank}(\mathbf{A}) \leq \min\{m, d\}$. We say that a matrix is *full rank* if $\text{rank}(\mathbf{A}) = \min\{m, d\}$. A matrix is *full column rank* (resp. *full row rank*) if the set of columns (resp. rows) of the matrix is linearly independent. If the matrix $\mathbf{A}$ is square (i.e., $m = d$) and it is full rank, then the inverse $\mathbf{A}^{-1}$ exists.

4.2 Nature of Solutions to Linear Systems

Often in engineering, we would like to "solve" systems of equations of the form

$$\mathbf{X}\mathbf{w} = \mathbf{y} \qquad \text{or} \qquad \begin{bmatrix}\underline{x}_1 & \underline{x}_2 & \cdots & \underline{x}_d\end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \tag{4.6}$$

where $\underline{x}_i, 1 \leq i \leq d$ are the *columns*[1] of $\mathbf{X}$ so

$$\underline{x}_i = \begin{bmatrix} x_{1,i} \\ x_{2,i} \\ \vdots \\ x_{m,i} \end{bmatrix} \in \mathbb{R}^m \quad \text{for all } i = 1, 2, \ldots, d. \tag{4.7}$$

[1]We usually denote the training samples, *rows* of $\mathbf{X}$, as $\mathbf{x}_i \in \mathbb{R}^d$. Hence, we use a different notation $\underline{x}_i \in \mathbb{R}^m$ to denote the *columns* of $\mathbf{X}$.

The matrix $\mathbf{X} \in \mathbb{R}^{m\times d}$ and vector $\mathbf{y} \in \mathbb{R}^m$ are given and $\mathbf{w} \in \mathbb{R}^d$ is to be found. As mentioned, if the matrix $\mathbf{X}$ is square and full rank, $\mathbf{X}^{-1}$ exists and so we can solve for $\mathbf{w}$ by simple matrix inversion and multiplication $\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$. However, most of the time in engineering, $m \neq d$ and more care is needed to discuss the existence and uniqueness of solutions to the linear system in (4.6). For this, we appeal to the Rouché-Capelli Theorem. We need the notion of the *augmented matrix*

$$\tilde{\mathbf{X}} = \begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix} \in \mathbb{R}^{m\times(d+1)}. \tag{4.8}$$

Note that the augmented matrix $\tilde{\mathbf{X}}$ has rank at least as large as that of $\mathbf{X}$, i.e., $\text{rank}(\mathbf{X}) \leq \text{rank}(\tilde{\mathbf{X}})$. This is because $\tilde{\mathbf{X}}$ has more columns than $\mathbf{X}$ so the dimension of its column space must be as large as that of $\mathbf{X}$.

Theorem 4.1 (Rouché-Capelli Theorem)**.** *For the linear system in* (4.6)*, the following hold:*

(i) The system in (4.6) *admits a unique solution if and only if*

$$\text{rank}(\mathbf{X}) = \text{rank}(\tilde{\mathbf{X}}) = d; \tag{4.9}$$

(ii) The system in (4.6) *has no solution if and only if*

$$\text{rank}(\mathbf{X}) < \text{rank}(\tilde{\mathbf{X}}); \tag{4.10}$$

(iii) The system in (4.6) *has infinitely many solutions if and only if*

$$\text{rank}(\mathbf{X}) = \text{rank}(\tilde{\mathbf{X}}) < d. \tag{4.11}$$

Proof sketch (Only the $\Longleftarrow$ directions). For part (i), we note that the condition that $\text{rank}(\mathbf{X}) = \text{rank}(\tilde{\mathbf{X}})$ means that $\mathbf{y}$ is in the column space of $\mathbf{X}$. This means that there exists $\{w_i\}_{i=1}^d \subset \mathbb{R}$ such that $\sum_{i=1}^d w_i \underline{x}_i = \mathbf{y}$ where the $\underline{x}_i$'s are the d columns of $\mathbf{X}$. Since $\text{rank}(\mathbf{X}) = d$, $\{\underline{x}_i\}_{i=1}^d$ span $\mathbb{R}^d$ and the representation $\sum_{i=1}^d w_i \underline{x}_i = \mathbf{y}$ is unique (see discussion after Definition 4.3), which means there is a unique solution.

For part (ii), the condition that $\text{rank}(\mathbf{X}) < \text{rank}(\tilde{\mathbf{X}})$ means that $\mathbf{y}$ is not in the column space of $\mathbf{X}$ so there is no solution.

For part (iii), since $\text{rank}(\mathbf{X}) < d$, $\{\underline{x}_i\}_{i=1}^d$ do not span $\mathbb{R}^d$ and the dimension of the nullspace of $\mathbf{X}$ is non-zero. This means that if $\hat{\mathbf{w}}_\text{p}$ is a particular solution so is $\hat{\mathbf{w}}_\text{p} + \mathbf{w}_0$ where $\mathbf{w}_0 \in \mathcal{N}(\mathbf{X})$. Hence, are infinitely many solutions. □

Let us consider a few examples.

- Consider the following over-determined system in which $m = 3$ and $d = 2$:

$$\mathbf{X} = \begin{bmatrix}2 & 1\\ 4 & 3\\ 5 & 6\end{bmatrix}, \qquad \text{and} \qquad \mathbf{y} = \begin{bmatrix}1\\ 2\\ 3\end{bmatrix}. \tag{4.12}$$

  The augmented matrix is

$$\tilde{\mathbf{X}} = \begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix} = \begin{bmatrix}2 & 1 & 1\\ 4 & 3 & 2\\ 5 & 6 & 3\end{bmatrix}. \tag{4.13}$$

  In this case $\text{rank}(\mathbf{X}) = 2$ and $\text{rank}(\tilde{\mathbf{X}}) = 3$. This is case (ii) of the Rouché-Capelli Theorem and there is no solution. This is the usual case for over-determined systems. Note that in Python, you can find the rank of a matrix (2D array) $\mathbf{A}$ using `np.linalg.matrix_rank(A)`.

- Consider the following over-determined system in which $m = 3$ and $d = 2$:

$$\mathbf{X} = \begin{bmatrix} 2 & 1 \\ 4 & 3 \\ 5 & 6 \end{bmatrix}, \qquad \text{and} \qquad \mathbf{y} = \begin{bmatrix} 4 \\ 10 \\ 17 \end{bmatrix}. \tag{4.14}$$

  In this case $\text{rank}(\mathbf{X}) = 2$ and $\text{rank}(\tilde{\mathbf{X}}) = 2$. This is case (i) of the Rouché-Capelli Theorem and there is a unique solution even though the system is over-determined. Note that $\mathbf{y}$ is one times the first column of $\mathbf{X}$ plus two times the second column of $\mathbf{X}$, so it is in the linear span of the columns of $\mathbf{X}$.

- Consider the following over-determined system in which $m = 3$ and $d = 2$:

$$\mathbf{X} = \begin{bmatrix} 2 & 1 \\ 4 & 2 \\ 6 & 3 \end{bmatrix}, \qquad \text{and} \qquad \mathbf{y} = \begin{bmatrix} 8 \\ 16 \\ 24 \end{bmatrix}. \tag{4.15}$$

  In this case $\text{rank}(\mathbf{X}) = 1$ and $\text{rank}(\tilde{\mathbf{X}}) = 1$ and both these ranks are less than $d = 2$. This is case (iii) of the Rouché-Capelli Theorem and there are infinitely many solutions even though the system is over-determined. Note that the three columns of $\tilde{\mathbf{X}}$ are collinear.

- Consider the following under-determined system in which $m = 2$ and $d = 3$:

$$\mathbf{X} = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 2 & 5 \end{bmatrix}, \qquad \text{and} \qquad \mathbf{y} = \begin{bmatrix} 10 \\ 7 \end{bmatrix}. \tag{4.16}$$

  In this case, $\text{rank}(\mathbf{X}) = 2$ and $\text{rank}(\tilde{\mathbf{X}}) = 2$ but $d = 3$. This is case (iii) of the Rouché-Capelli Theorem and there are infinitely many solutions. This is the usual case for under-determined systems.

- Consider the following under-determined system in which $m = 2$ and $d = 3$:

$$\mathbf{X} = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 2 & 6 \end{bmatrix}, \qquad \text{and} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}. \tag{4.17}$$

  In this case, $\text{rank}(\mathbf{X}) = 1$ and $\text{rank}(\tilde{\mathbf{X}}) = 2$ because $\mathbf{y} \notin \mathcal{R}(\mathbf{X})$. This is case (ii) of the Rouché-Capelli Theorem and there is no solution. Note that $\mathbf{y}$ boosts the rank of $\mathbf{X}$ by 1 in the augmented matrix $\tilde{\mathbf{X}}$, i.e., $\mathbf{y}$ is not in the column space of $\mathbf{X}$, which is the ray $\{[t, 2t]^\top : t \in \mathbb{R}\}$.

- For under-determined systems ($m < d$), can we have case (i)? See Exercise 4.4.

In the above, we have indicated some *usual cases* for over- and under-determined systems. What do we mean by this? We mean that if the matrix $\mathbf{X}$ and vector $\mathbf{y}$ are chosen randomly from continuous distributions, then the chance, or probability, that an unusual scenario occurs is 0. In other words, for an unusual scenario to occur, one has to be very lucky, e.g., for the example in (4.14), $\mathbf{y}$ lies in the span of the columns of $\mathbf{X}$. If $\mathbf{y}$ were sampled from a continuous distribution, the chance that it lies in the span of $\mathbf{X}$ is exactly 0. This is because a two dimensional plane is "infinitely skinny" in three dimensional space.

4.3 Least Squares Estimation for $m > d$

We consider the case in which $\mathbf{X}$ is tall ($m > d$) and full rank. This means that $\text{rank}(\mathbf{X}) = d$; equivalently, all columns are linearly independent. This *over-determined* scenario happens a lot in engineering. For example, this happens in estimation problems, where one tries to estimate a small number d of parameters given a lot of (noisy) experimental measurements, say $m > d$. As seen from the usual case of the Rouché-Capelli Theorem (case (ii) in which $\mathbf{y}$ is not in the linear span of the columns of $\mathbf{X}$), there is no solution. Hence, one way to find "the best" solution is to minimize the sum of squares of the errors

$$\text{minimize} \quad \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2. \tag{4.18}$$

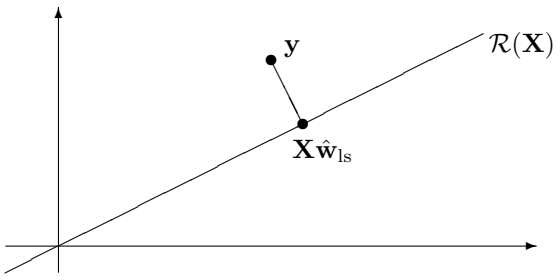


Figure 4.1: The least squares problem consists in finding $\hat{\mathbf{w}}_{\text{ls}}$, the point such that $\mathbf{X}\hat{\mathbf{w}}_{\text{ls}} \in \mathcal{R}(\mathbf{X})$ is closest to a given point $\mathbf{y}$.

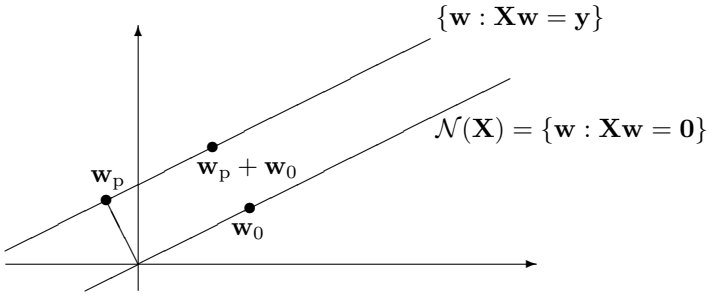


Figure 4.2: The least norm problem consists of finding the particular solution $\hat{\mathbf{w}}_{\text{p}}$ that minimizes the norm. All other solutions $\hat{\mathbf{w}}_{\text{p}} + \mathbf{w}_0$ where $\mathbf{w}_0 \in \mathcal{N}(\mathbf{X})$ have norms that are at least as large as that of $\hat{\mathbf{w}}_{\text{p}}$.

The optimal $\mathbf{w}$ is the *least squares estimate* (why can we use the article "the" here?). We can solve this problem by means of calculus (a more elegant way is through the projection theorem explored in Chapter 5). See Appendix 4.A for a recapitulation of the relevant multivariate calculus you need to know. Now, letting $f(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$, we obtain

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \nabla_{\mathbf{w}} \left(\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{y}^\top \mathbf{X}\mathbf{w} + \mathbf{y}^\top \mathbf{y} \right) = 2\mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{X}^\top \mathbf{y}. \tag{4.19}$$

Setting this to zero, we see that the optimal $\mathbf{w}$ is the least squares estimate

$$\hat{\mathbf{w}}_{\text{ls}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{4.20}$$

The existence of $(\mathbf{X}^\top \mathbf{X})^{-1}$ is guaranteed by the fact that $\mathbf{X}$ has full column rank. See the geometry of the problem in Fig. 4.1.

A few words about the matrix $\mathbf{X}^\dagger := (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$. This matrix is called the *pseudo-inverse* of the full rank tall matrix $\mathbf{X}$. It is also called the *left-inverse* of $\mathbf{X}$ because if we multiply $\mathbf{X}$ on the left with $\mathbf{X}^\dagger$, we obtain $\mathbf{X}^\dagger \mathbf{X} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{X} = \mathbf{I}$.

The left-inverse of $\mathbf{X}$ can be implemented in Python as `np.linalg.pinv(X)`. Note that this function is more powerful than computing the left-inverse; it computes the so-called Moore–Penrose pseudo-inverse. When the matrix has full column rank, the Moore–Penrose pseudo-inverse coincides with the left-inverse.

4.4 Least Norm Solution for $m < d$

Now we consider the case in which $\mathbf{X}$ is wide ($m < d$) and full rank. This means that $\text{rank}(\mathbf{X}) = m$; equivalently, all rows are linearly independent (full row rank). This *under-determined* situation also occurs a lot in engineering.

Example 4.3. *For example, in control engineering (a field of study within mechanical and electrical engineering), one often considers the following discrete-time state-space system (e.g., describing the dynamics of a robot operating over a quantized time interval):*

$$v_{i+1} = av_i + bw_i, \quad i = 0, 1, \ldots, d-1, \tag{4.21}$$

where v_i is the state *of the system at time i and w_i is our* control*. We assume the system starts at the origin $v_0 = 0$. We desire to design the w_i's such that the terminal state $v_d = y$ (for some given y) while minimizing the* cost *of the control $\sum_{i=0}^{d-1} w_i^2$. After some algebra, this can be rewritten as*

$$\textit{minimize} \quad \|\mathbf{w}\|^2 \quad \textit{subject to} \quad y = \begin{bmatrix} b & ab & a^2b & \cdots & a^{d-1}b \end{bmatrix} \begin{bmatrix} w_{d-1} \\ w_{d-2} \\ \vdots \\ w_0 \end{bmatrix}. \tag{4.22}$$

*This is exactly an under-determined problem if we make the identifications $\mathbf{X} = \begin{bmatrix} b & ab & a^2b & \cdots & a^{d-1}b \end{bmatrix}$ (for obvious reasons, this is called the d-*step reachability matrix*) and $\mathbf{w} = \begin{bmatrix} w_{d-1} & w_{d-2} & \cdots & w_0 \end{bmatrix}^\top$ and y is scalar (i.e., $m = 1$).*

We return to the equation $\mathbf{X}\mathbf{w} = \mathbf{y}$ in which $m < d$ and the matrix $\mathbf{X}$ has full row rank. It is clear that $(\mathbf{X}\mathbf{X}^\top)^{-1}$ exists and

$$\hat{\mathbf{w}}_{\mathrm{p}} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} \tag{4.23}$$

is a solution to the equation (check!). The subscript p indicates that this is a *particular* solution to the linear system. From the usual case of the Rouché-Capelli Theorem (case (iii)), we know that there are infinitely many solutions. Where are these infinitely many solutions? Let $\mathbf{w}_0 \in \mathcal{N}(\mathbf{X})$ be any vector in the nullspace of $\mathbf{X}$. Note that the nullspace has positive dimension because $\mathrm{rank}(\mathbf{X}) = m < d$ so $\mathbf{w}_0$ can be chosen to be a non-zero vector. Then $\hat{\mathbf{w}}_{\mathrm{p}} + \mathbf{w}_0$ is also a solution to (4.6) (check!). In the following, we argue that among all the solutions, $\hat{\mathbf{w}}_{\mathrm{p}}$ has a special place in our hearts because is the *least norm solution* to (4.6).

Suppose that $\mathbf{w}$ is any solution to $\mathbf{X}\mathbf{w} = \mathbf{y}$. Then since $\hat{\mathbf{w}}_{\mathrm{p}}$ is also a solution, we have $\mathbf{X}(\mathbf{w} - \hat{\mathbf{w}}_{\mathrm{p}}) = \mathbf{0}$ and

$$(\mathbf{w} - \hat{\mathbf{w}}_{\mathrm{p}})^\top \hat{\mathbf{w}}_{\mathrm{p}} \overset{(4.23)}{=} (\mathbf{w} - \hat{\mathbf{w}}_{\mathrm{p}})^\top \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} = \big(\mathbf{X}(\mathbf{w} - \hat{\mathbf{w}}_{\mathrm{p}})\big)^\top (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} = \mathbf{0}. \tag{4.24}$$

This means that $\mathbf{w} - \hat{\mathbf{w}}_{\mathrm{p}}$ is orthogonal to $\hat{\mathbf{w}}_{\mathrm{p}}$. By the Pythagorean theorem,

$$\|\mathbf{w}\|^2 = \|(\mathbf{w} - \hat{\mathbf{w}}_{\mathrm{p}}) + \hat{\mathbf{w}}_{\mathrm{p}}\|^2 = \|\mathbf{w} - \hat{\mathbf{w}}_{\mathrm{p}}\|^2 + \|\hat{\mathbf{w}}_{\mathrm{p}}\|^2 \geq \|\hat{\mathbf{w}}_{\mathrm{p}}\|^2, \tag{4.25}$$

which shows that $\hat{\mathbf{w}}_{\mathrm{p}}$ is the least norm solution to $\mathbf{X}\mathbf{w} = \mathbf{y}$, so we also denote this as

$$\hat{\mathbf{w}}_{\mathrm{ln}} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} \tag{4.26}$$

See the geometry of the least norm problem in Fig. 4.2.

Finally, we say a few words about the matrix $\mathbf{X}^\dagger = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^{-1}$. This matrix is called the *pseudo-inverse* of the full-rank wide matrix $\mathbf{X}$. It is also know as the *right-inverse* of $\mathbf{X}$ (why?). The right-inverse of a matrix with full row rank can be implemented in Python as `np.linalg.pinv(X)`.

4.5 Practice Problems

Exercise 4.1. *Prove that $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$ is linearly independent if and only if no vector $\mathbf{x}_i$ can be expressed as a linear combination of the other vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_k$.*

Exercise 4.2. *Can the linear system $\mathbf{X}\mathbf{w} = \mathbf{y}$ have* finitely *many solutions? If you think so, provide an example else formulate a precise mathematical statement and supply a proof of your statement.*

Exercise 4.3. *Prove the $\implies$ directions of the Rouché-Capelli Theorem.*

Exercise 4.4. *For under-determined systems ($m < d$), briefly explain why we cannot have case (i) of the Rouché-Capelli Theorem? We have seen that for over-determined systems ($m > d$) all three cases of the Rouché-Capelli Theorem are possible. Why is there an asymmetry here?*

Exercise 4.5. *Show that for any matrix $\mathbf{X}$, we have* $\text{rank}(\mathbf{X}) = \text{rank}(\mathbf{X}\mathbf{X}^\top) = \text{rank}(\mathbf{X}^\top\mathbf{X})$.

Exercise 4.6. *Consider the square system $\mathbf{X}\mathbf{w} = \mathbf{y}$ where*

$$\mathbf{X} = \begin{bmatrix} -3 & 2 \\ 5 & a \end{bmatrix}, \qquad \textit{and} \qquad \mathbf{y} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}. \tag{4.27}$$

Find the value of a such that there is no solution to $\mathbf{X}\mathbf{w} = \mathbf{y}$.

Exercise 4.7. *Consider the square system $\mathbf{X}\mathbf{w} = \mathbf{y}$ where*

$$\mathbf{X} = \begin{bmatrix} -4 & 2 \\ 2 & b \end{bmatrix}, \qquad \textit{and} \qquad \mathbf{y} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}. \tag{4.28}$$

Find the value of b such that there is at least one solution to $\mathbf{X}\mathbf{w} = \mathbf{y}$.

Exercise 4.8. *Consider the over-determined system $\mathbf{X}\mathbf{w} = \mathbf{y}$ where*

$$\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 5 & 10 \\ 3 & 7 \end{bmatrix}, \qquad \textit{and} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 5 \\ c \end{bmatrix}. \tag{4.29}$$

Find the value of c such that there is a unique solution to $\mathbf{X}\mathbf{w} = \mathbf{y}$.

Exercise 4.9. *Consider the under-determined system $\mathbf{X}\mathbf{w} = \mathbf{y}$ where*

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & -1 \\ -1 & -2 & d \end{bmatrix}, \qquad \textit{and} \qquad \mathbf{y} = \begin{bmatrix} -3 \\ 3 \end{bmatrix}. \tag{4.30}$$

Find the value of d such that there is no solution to $\mathbf{X}\mathbf{w} = \mathbf{y}$.

Exercise 4.10. *Consider the matrix*

$$\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 5 & 10 \\ 3 & e \end{bmatrix}. \tag{4.31}$$

Find the set of values of e such that the left-inverse of $\mathbf{X}$ exists.

Exercise 4.11. *Consider the matrix*

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 3 \\ -2 & -2 & f \end{bmatrix}. \tag{4.32}$$

Find the set of values of f such that the right-inverse of $\mathbf{X}$ does not exist.

Exercise 4.12. *Consider the linear system $\mathbf{X}\mathbf{w} = \mathbf{y}$ where $\mathbf{X}$ is an $m \times d$ matrix. This represents a linear system of m equations in d variables. Give a proof or counterexample for each of the following:*

1. *If $d = m$, there is always* at most *one solution.*
2. *If $d > m$, we can always find a solution to $\mathbf{X}\mathbf{w} = \mathbf{y}$.*
3. *If $d > m$ the nullspace of $\mathbf{X}$ has dimension greater than zero.*
4. *If $d < m$ then for some $\mathbf{y}$, there is no solution of $\mathbf{X}\mathbf{w} = \mathbf{y}$.*
5. *If $d < m$, the only solution of $\mathbf{X}\mathbf{w} = \mathbf{0}$ is $\mathbf{w} = \mathbf{0}$.*

4.A Differentiation of Scalar-Valued Functions of Several Variables

In this course, we will need to differentiate scalar-valued differentiable functions of several variables. Such a function takes the form $f : \mathbb{R}^d \to \mathbb{R}$. That is, it maps a vector of d components to a real-number (scalar). We often want to differentiate this with respect to the argument $\mathbf{w}$. The derivative, denoted as $\nabla_{\mathbf{w}} f(\mathbf{w})$, is a function that maps a vector to a vector. We write this as $\nabla_{\mathbf{w}} f : \mathbb{R}^d \to \mathbb{R}^d$. The i^{th} component of the vector $\nabla_{\mathbf{w}} f(\mathbf{w})$ is $\partial f(\mathbf{w})/\partial w_i$ and so we can express the column vector $\nabla_{\mathbf{w}} f(\mathbf{w})$ as

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \begin{bmatrix} \dfrac{\partial f(\mathbf{w})}{\partial w_1} & \dfrac{\partial f(\mathbf{w})}{\partial w_2} & \cdots & \dfrac{\partial f(\mathbf{w})}{\partial w_d} \end{bmatrix}^\top. \tag{4.33}$$

In the majority of this course (sans Chapter 12), we will only need to apply this to two examples, namely the functions $f(\mathbf{w}) = \mathbf{w}^\top \mathbf{a}$ and $g(\mathbf{w}) = \mathbf{w}^\top \mathbf{A}\mathbf{w}$ for a vector $\mathbf{a} \in \mathbb{R}^d$ and a square matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$. Note that these two functions map vectors to scalars. For the former, we have

$$\nabla_{\mathbf{w}} f(\mathbf{w}) = \nabla_{\mathbf{w}} (\mathbf{w}^\top \mathbf{a}) = \mathbf{a}. \tag{4.34}$$

This is because $\mathbf{w}^\top \mathbf{a} = \sum_{j=1}^d w_j a_j$ and differentiating this with respect to w_i (for a fixed $1 \leq i \leq d$) yields a_i. This generalizes the familiar derivative rule $\frac{\mathrm{d}}{\mathrm{d}w}(aw) = a$. For the latter function $g(\mathbf{w}) = \mathbf{w}^\top \mathbf{A}\mathbf{w}$, very careful bookkeeping yields

$$\nabla_{\mathbf{w}} g(\mathbf{w}) = (\mathbf{A} + \mathbf{A}^\top)\mathbf{w}. \tag{4.35}$$

Often, we will apply this result to *symmetric*[2] matrices (in which $\mathbf{A} = \mathbf{A}^\top$) so (4.35) reduces to $\nabla_{\mathbf{w}} g(\mathbf{w}) = 2\mathbf{A}\mathbf{w}$. This generalizes the familiar derivative rule $\frac{\mathrm{d}}{\mathrm{d}w}(aw^2) = 2aw$. For more on differentiation of such functions and further generalizations, please refer to the Matrix Cookbook [PP12].

[2] An example of a symmetric matrix is the Gram matrix $\mathbf{X}^\top \mathbf{X}$.

Chapter 5

Least Squares Estimation and the Projection Theorem

In this chapter, we will present one interesting example of least squares estimation. We will also unify the concepts of least squares estimation and the least norm solution from Chapter 4 via the Projection Theorem.

5.1 Predicting the Outcome of Presidential Elections: An Application of Least Squares

While most of news this year has been dominated by Covid-19, there is still the little matter of another event that has monumental importance to the world—the quadrennial Presidential Elections in the United States of America, pitting the incumbent President Donald J. Trump (R) against former Vice President Joe Biden (D). We will put what we have learned to do linear regression on four potentially important factors that influence the incumbent's winning (or losing) margin. This analysis is based on Nate Silver's study in November 2011, one year before President Obama beat Mitt Romney in the 2012 elections. The original article, which contains much more analysis, can be found here—`https://fivethirtyeight.blogs.nytimes.com/2011/11/18/which-economic-indicators-best-predict-presidential-elections/`.

We consider four economic indicators:

(a) Real GDP growth rate x_1;

(b) Change in non-farm payrolls x_2;

(c) ISM (Institute of Supply Management) manufacturing index x_3;

(d) Unemployment rate x_4.

We consider predicting the target variable—the *incumbent party's margin of victory* y. This can be negative if the incumbent party wins fewer votes than the challenger. For example, in 1996 the incumbent Bill Clinton (D) beat Bob Dole (R) by 8.5% so $y = 8.5$. In 2008, Barack Obama (D) beat John McCain (R) by 7.2% and the incumbent was a Republican (George W. Bush) so $y = -7.2$. Scatter plots of each of the unnormalized or raw indicators against the incumbent party's margin of victory are shown in Fig. 5.1. We have data of all the above economic indicators and margins of victory from 1948 (Truman against Dewey) to 2008 (Obama against McCain), constituting 16 presidential elections. Do note that even if a candidate wins the popular vote, s/he may not be elected president as one needs to win the so-called electoral college instead of the popular vote. For example, George W. Bush became president-elect in 2000 even though he lost the popular vote to Al Gore and more recently, Donald J. Trump became president-elect in 2016 even though he lost the popular vote (by 3 million votes) to Hillary R. Clinton.

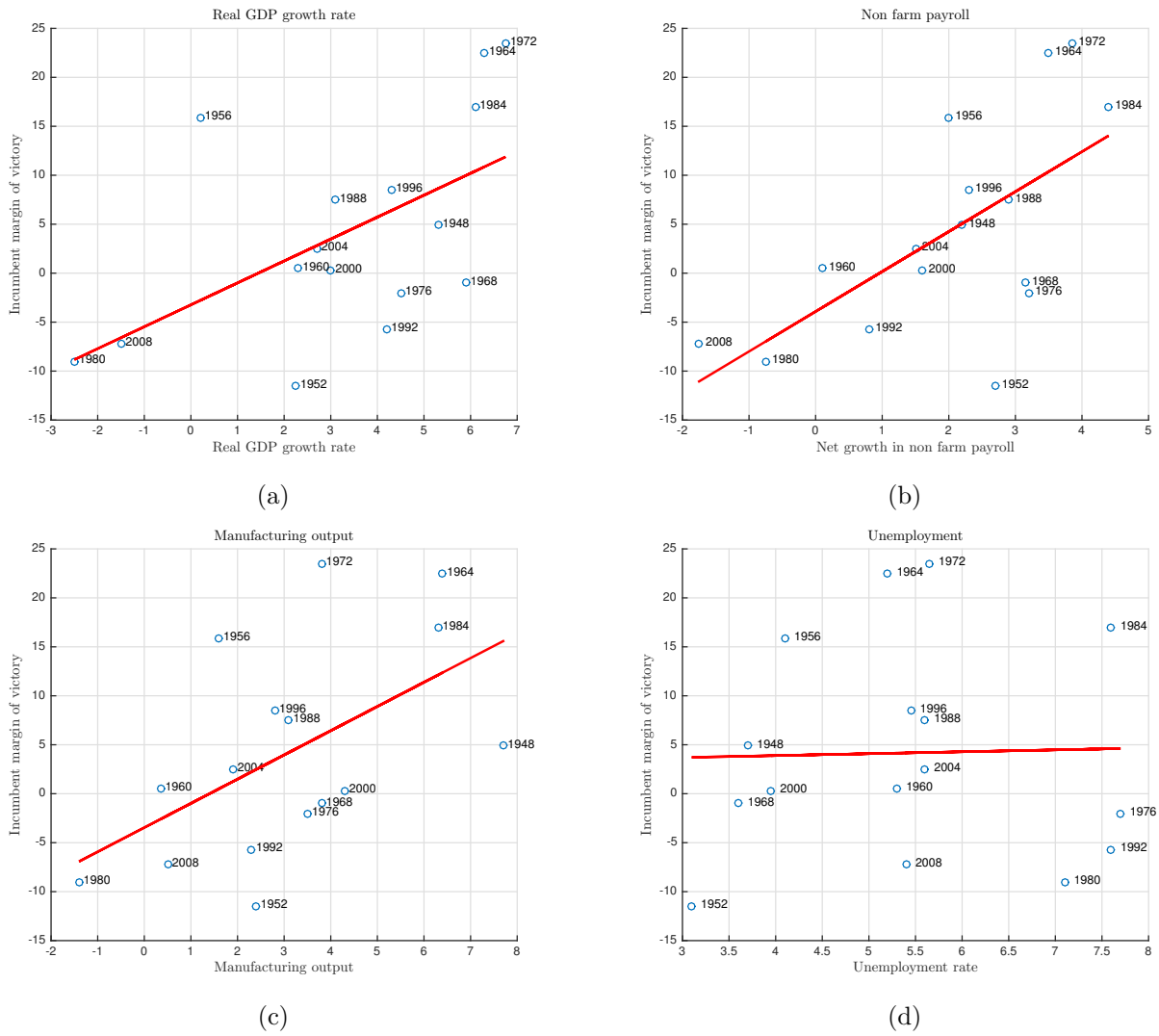


Figure 5.1: Scatter plots of incumbent party's victory margin against various economic factors

We do a min-max normalization of the features to ensure that all of them are in $[0, 1]$. We include the offset or bias term to obtain our design matrix $\mathbf{X}$ and target vector $\mathbf{y}$, i.e.,

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ 1 & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{16,1} & x_{16,2} & x_{16,3} & x_{16,4} \end{bmatrix} \in [0,1]^{16\times 5} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{16} \end{bmatrix} \in \mathbb{R}^{16}. \tag{5.1}$$

Hence, $x_{2,2}$ corresponds to the change in non-farm payrolls of the 1952 election (Eisenhower vs Stevenson).

In Fig. 5.2, we show the regression lines for each feature against y. For example, for Fig. 5.2(a), we only consider the matrix[1] $\mathbf{X}(:, 1:2) \in [0,1]^{16\times 2}$ (first two columns of $\mathbf{X}$) and learn the vector

$$\hat{\mathbf{w}}_1 = \big(\mathbf{X}(:,1:2)^\top \mathbf{X}(:,1:2)\big)^{-1} \mathbf{X}(:,1:2)^\top \mathbf{y}. \tag{5.2}$$

[1]Here, we find it convenient to use Matlab notation. In particular $\mathbf{X}(:, a:b)$ denotes the submatrix of $\mathbf{X}$ consisting of all its columns indexed from a to b.

From the plots, we visually see that the first three economic indicators seem to have correlation with y. Unemployment rate, somewhat surprisingly, has minimal impact on y. In statistics, a common measure of linear dependence is the coefficient of determination or R^2 value. This is the proportion of the variance in the dependent variable y that is predictable from the independent variable x_i. You do not need to know what the formula is or how to use it but all that is needed to appreciate is that the closer the value is to one, the larger the linear dependence. The R^2 values for each of the variables is tabulated in Table 5.1. Again the values confirm that the first three independent variables are rather correlated to y.

Variable	GDP Growth Rate	Non-Farm Payroll	Manufacturing Output	Unemployment Rate
R^2	0.3134	0.3984	0.2892	0.0007

Table 5.1: R^2 for each of the economic indicators

Now, let us use the three variables x_1, x_2 and x_4 as our variables to do linear regression, omitting x_3 (Manufacturing output) for simplicity. The design matrix is thus the horizontally concatenated matrix $\mathbf{X}' = [\mathbf{X}(:, 1:3)\ \mathbf{X}(:, 5)] \in [0,1]^{16\times 4}$ (we omit the 4th column corresponding to x_3). Then we estimate

$$\hat{\mathbf{w}}_{1,2,4} = \left((\mathbf{X}')^\top \mathbf{X}'\right)^{-1} (\mathbf{X}')^\top \mathbf{y} = \begin{bmatrix} -12.5102 \\ 4.1531 \\ 21.5454 \\ 1.9871 \end{bmatrix}. \tag{5.3}$$

So the offset term if -12.51 and the coefficients associated to GDP growth rate, change in non-farm payroll and unemployment rate are $4.15, 21.55$ and 1.99 respectively. Again, we see that the last feature seems to have a small effect due to the small coefficient.

Suppose we want to predict the winning margin in 2004 (the 15^{th} data sample). So as not to bias our model, we remove this data point from $\mathbf{X}' \in [0,1]^{16\times 4}$ and $\mathbf{y} \in \mathbb{R}^{16}$. Call the new design matrix and dependent vector $\mathbf{X}'' \in [0,1]^{15\times 4}$ and $\mathbf{y}'' \in \mathbb{R}^{15}$ respectively. The coefficient vector learned in the absence of the 2004 data point is

$$\hat{\mathbf{w}}''_{1,2,4} = \left((\mathbf{X}'')^\top \mathbf{X}''\right)^{-1} (\mathbf{X}'')^\top \mathbf{y}'' = \begin{bmatrix} -12.5310 \\ 4.1519 \\ 21.5617 \\ 1.9822 \end{bmatrix}. \tag{5.4}$$

Great! The removal of one data example did not affect the regression coefficients too much. Now let's try to predict the winning margin in 2004. We have

$$\hat{y}_{15} = \underbrace{\begin{bmatrix} 1 \\ 0.5622 \\ 0.5285 \\ 0.5435 \end{bmatrix}^\top}_{\text{2004's normalized feature vector}} \hat{\mathbf{w}}''_{1,2,4} = 2.2748\% \tag{5.5}$$

From the economic indicators, this means that George W. Bush (the incumbent in 2004) is expected to win by 2.27%. In actual fact, he won the popular vote by 2.4%, which means that our model did pretty well! Some natural questions.

1. It seems from Fig. 5.1(d) that the unemployment rate is not a very good predictor of the incumbent's margin of victory. What if we removed it from the model? Do you expect predictions to improve? Try it for yourself using the csv file, which is provided with this tutorial.

2. What if we included manufacturing output as a variable in the model? Do you expect predictions to improve?

3. Why was our prediction of 2004's result in (5.5) so good? From Fig. 5.1, we see that the data for 2004 are very close to the individual regression lines. What if we tried to predict a result in a very "atypical" year, e.g., 1956?

4. Redo everything for the electoral college to predict the president-elects (which is arguably more important). Before collecting results and doing any machine learning, do you expect your results to be more or less accurate than predicting the winning margins?

5. We predicted a result that is known (2004's result), which is not so interesting. Try predicting Donald Trump's winning or losing margin against Joe Biden this Fall. For this, you need to know how and where to get reliable data. Let me know if you find something interesting! *We now know that Joe Biden beat Donald Trump by 3.8% based on the popular vote. More importantly, Joe Biden beat Donald Trump in the electoral college (306 to 232) to become the* 46^{th} *president of the United States.*

5.2 The Projection Theorem (Optional)

In this section, we unify the solutions of the least squares estimator and the least norm solution via the Projection Theorem.

Theorem 5.1. *Let* $\mathcal{M} \subset \mathcal{V}$ *be a subspace of a vector space* $\mathcal{V}$*. Then the solution* $\hat{\mathbf{m}}$ *to the optimization problem*

$$\text{minimize} \quad \|\mathbf{y} - \mathbf{m}\| \quad \text{subject to} \quad \mathbf{m} \in \mathcal{M} \tag{5.6}$$

is unique and satisfies

$$(\mathbf{y} - \hat{\mathbf{m}}) \perp \mathcal{M}. \tag{5.7}$$

Conversely, if $\hat{\mathbf{m}}$ *satisfies* (5.7)*, it is the optimal solution to* (5.6)*.*

This means that the optimal solution $\hat{\mathbf{m}}$ is such that the induced "error vector" $\mathbf{y} - \hat{\mathbf{m}}$ is orthogonal to the any vector that lies in the subspace $\mathcal{M}$. The proof is given in Appendix 5.A.

5.2.1 Application to Least Squares Estimation

Let us see how to apply this to the least squares estimation problem, a problem considered and solved by Legendre and Gauss. In Chapter 4, we saw how to do this by means of differentiating functions of vectors. It turns out that you neither need to remember nor derive these formulate—i.e., differentiation (as was done in Appendix 4.A) is not necessary. Recall that in the least squares problem, we are given a matrix $\mathbf{X} \in \mathbb{R}^{m \times d}$ with full column rank in which $m > d$ and a vector $\mathbf{y} \in \mathbb{R}^d$ and considered the problem

$$\text{minimize} \quad \|\mathbf{y} - \mathbf{X}\mathbf{w}\|. \tag{5.8}$$

In other words, we want to find $\mathbf{w}$ in the range of $\mathbf{X}$ such that $\mathbf{X}\mathbf{w}$ is closest to $\mathbf{y}$. By the Projection Theorem, $\hat{\mathbf{w}}_{\text{ls}}$ is optimal if and only if

$$(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}_{\text{ls}}) \perp \mathcal{R}(\mathbf{X}). \tag{5.9}$$

Note that $\mathbf{X}\hat{\mathbf{w}}_{\text{ls}}$ plays the role of $\hat{\mathbf{m}}$ in the Projection Theorem. However, (5.9) means that

$$\underline{x}_i^\top (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}_{\text{ls}}) = 0, \quad \text{for all } \; i = 1, 2, \ldots, d, \tag{5.10}$$

where $\{\underline{x}_i\}_{i=1}^d \subset \mathbb{R}^m$ is the set of d columns of $\mathbf{X}$. Stacking the conditions in (5.10) together in matrix form, we obtain

$$\begin{bmatrix} \underline{x}_1^\top \\ \underline{x}_2^\top \\ \vdots \\ \underline{x}_d^\top \end{bmatrix} (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}_{\text{ls}}) = \mathbf{0} \quad \Longleftrightarrow \quad \mathbf{X}^\top (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}_{\text{ls}}) = \mathbf{0} \quad \Longleftrightarrow \quad \hat{\mathbf{w}}_{\text{ls}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{5.11}$$

Note that the inverse of $\mathbf{X}^\top\mathbf{X}$ exists because $\mathbf{X}$ has full column rank. This recovers the least squares solution without knowledge of differentiation.

5.2.2 Application to the Least Norm Problem

Recall that in the least norm problem, we are given a matrix $\mathbf{X} \in \mathbb{R}^{m\times d}$ with full row rank in which $m < d$ and a vector $\mathbf{y} \in \mathbb{R}^d$ and we considered the problem

$$\text{minimize} \quad \|\mathbf{w}\| \quad \text{subject to} \quad \mathbf{X}\mathbf{w} = \mathbf{y}. \tag{5.12}$$

By the Rouché-Capelli Theorem (Theorem 4.1), we see that since $\text{rank}(\mathbf{X}) = \text{rank}([\mathbf{X}\ \mathbf{y}]) = m < d$, there are infinitely many solutions to the system $\mathbf{X}\mathbf{w} = \mathbf{y}$. Let

$$\hat{\mathbf{w}}_\text{p} := \mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} \tag{5.13}$$

be one of the solutions (check that this is indeed a solution!). By writing $\mathbf{w} = \hat{\mathbf{w}}_\text{p} - \mathbf{w}_0$ for some $\mathbf{w}_0$, above optimization problem can be rewritten as

$$\text{minimize} \quad \|\hat{\mathbf{w}}_\text{p} - \mathbf{w}_0\| \quad \text{subject to} \quad \mathbf{X}(\hat{\mathbf{w}}_\text{p} - \mathbf{w}_0) = \mathbf{y} \tag{5.14}$$

where the optimization variable is now $\mathbf{w}_0$. Our objective is to show that the optimal $\mathbf{w}_0 = \mathbf{0}$, which would validate that $\hat{\mathbf{w}}_\text{p}$ is the least norm solution. Since $\mathbf{X}\hat{\mathbf{w}}_\text{p} = \mathbf{y}$, the optimization problem in (5.14) is the same as

$$\text{minimize} \quad \|\hat{\mathbf{w}}_\text{p} - \mathbf{w}_0\| \quad \text{subject to} \quad \mathbf{X}\mathbf{w}_0 = \mathbf{0}. \tag{5.15}$$

This means that

$$\text{minimize} \quad \|\hat{\mathbf{w}}_\text{p} - \mathbf{w}_0\| \quad \text{subject to} \quad \mathbf{w}_0 \in \mathcal{N}(\mathbf{X}). \tag{5.16}$$

By the Projection Theorem,

$$(\hat{\mathbf{w}}_\text{p} - \mathbf{w}_0) \perp \mathcal{N}(\mathbf{X}). \tag{5.17}$$

Since

$$\mathbf{w}_0 \in \mathcal{N}(\mathbf{X}) \tag{5.18}$$

we have

$$0 \overset{(5.17)}{=} (\hat{\mathbf{w}}_\text{p} - \mathbf{w}_0)^\top\mathbf{w}_0 \overset{(5.13)}{=} (\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} - \mathbf{w}_0)^\top\mathbf{w}_0 \overset{(5.18)}{=} \mathbf{y}^\top\big((\mathbf{X}\mathbf{X}^\top)^{-1}\big)^\top \underbrace{\mathbf{X}\mathbf{w}_0}_{=\mathbf{0}} - \mathbf{w}_0^\top\mathbf{w}_0. \tag{5.19}$$

Thus, $\|\mathbf{w}_0\|^2 = 0$ which means that $\mathbf{w}_0 = \mathbf{0}$ as desired. This proof uses the Projection Theorem in (5.17) to show that the minimum norm solution $\mathbf{w} = \hat{\mathbf{w}}_\text{p} - \mathbf{w}_0$ of $\mathbf{X}\mathbf{w} = \mathbf{y}$ is such that $\mathbf{w}_0 = \mathbf{0}$.

5.3 Practice Problems

Exercise 5.1. *In the least squares problem, we wish to minimize the squared ℓ_2 norm of the error vector $\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{w}$ over $\mathbf{w}$. That is, we are interested in minimizing $\|\mathbf{e}\|^2 = e_1^2 + e_2^2 + \ldots + e_m^2$. Devise a new least squares solution if we wish to instead minimize the weighted norm of the error vector $\|\mathbf{e}\|_\mathbf{q}^2 = q_1e_1^2 + q_2e_2^2 + \ldots + q_me_m^2$ where $q_i > 0$ for all i.*

Exercise 5.2. *In the least norm problem, we are interested to minimize $\|\mathbf{w}\|$ subject to $\mathbf{X}\mathbf{w} = \mathbf{y}$. Devise a new least norm solution if we wish to instead minimize the weighted norm of the weight vector $\|\mathbf{w}\|_\mathbf{q}^2 = q_1w_1^2 + q_2w_2^2 + \ldots + q_dw_d^2$ where $q_i > 0$ for all i.*

Exercise 5.3. *This is a problem on autoregressive (AR) modelling. In econometrics (or electrical, control or mechanical engineering), an autoregressive model is a representation of a random process. The simplest such process consists in having the current value of a certain stock x_t as a linear function of its past value x_{t-1} plus some imperfections e_t. In other words,*

$$x_t = wx_{t-1} + e_t \quad \textit{for all} \quad t = 1, 2, \ldots, m. \tag{5.20}$$

Formulate this as a least squares problem to find the least squares estimate for the scalar w by minimizing the sum of squares $e_1^2 + e_2^2 + \ldots + e_m^2$. The system in (5.20) *is known as an* AR(1) *process since x_t depends on its immediate past. A slight generalization of* (5.20) *is the* AR(d) *process in which x_t depends linearly on the past d stock values $(x_{t-1}, x_{t-2}, \ldots, x_{t-d})$, i.e.,*

$$x_t = \sum_{k=1}^{d} w_k x_{t-k} + e_t \quad \textit{for all} \quad t = 1, 2, \ldots, m. \tag{5.21}$$

Again, formulate this as a least squares problem to find the least squares estimate for $(w_1, w_2, \ldots, w_d)$.

Exercise 5.4. *This is a problem on recursive least squares. Consider the system $\mathbf{X}\mathbf{w} = \mathbf{y}$ where $\mathbf{X}$ is an $m \times d$ matrix that has full column rank. Then the least squares solution is $\hat{\mathbf{w}}_{\text{ls}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$. Suppose we obtain one more data point $(\mathbf{x}_{m+1}, y_m)$ and so the design matrix becomes $\bar{\mathbf{X}} = [\mathbf{X}^\top \ \mathbf{x}_{m+1}^\top]^\top$ and it has $m + 1$ rows and d columns. The observation vector becomes $\bar{\mathbf{y}} = [\mathbf{y}, y_{m+1}]^\top$. Express the least squares estimator of the system $\bar{\mathbf{X}}\bar{\mathbf{w}} = \bar{\mathbf{y}}$, namely $\hat{\bar{\mathbf{w}}}_{\text{ls}}$, in terms of the old least squares solution $\hat{\mathbf{w}}_{\text{ls}}$, the new data point $(\mathbf{x}_{m+1}, y_m)$. You may find the Woodbury formula* (6.35) *in Appendix 6.B useful.*

The point here is that the least squares estimator can be updated in an online or recursive fashion without having to perform any computationally heavy inversion of the $(m + 1) \times (m + 1)$ Gram matrix $\bar{\mathbf{X}}^\top \bar{\mathbf{X}}$ given we know the least squares solution $\hat{\mathbf{w}}_{\text{ls}}$ to the slightly smaller $m \times d$ system.

The preceding two problems form the basis of the the *Kalman filter* (named after Rudolf E. Kalman), which was used in the *Apollo program* in the 1960s and 70s to send humans to the moon and back safely.

The following problem is inspired by notes from MIT 6.241.

Exercise 5.5 (Challenging)**.** *This is an example from mechanical engineering that makes use of the least norm solution as discussed in Section 4.4. It can be considered as a continuous-time analogue of Example 4.3. Consider a unit mass particle moving in a straight line with a force $w(t)$ with position at time t given by $y(t)$. The particle starts from rest at the origin, i.e., $y(0) = \dot{y}(0) = 0$ and we would like to use the force $w(t)$ (which is equal to the acceleration $\ddot{y}(t)$) to manoeuvre the particle to location $\bar{y}$ at time T, i.e., $y(T) = \bar{y}$ (with no constraint on $\dot{y}(T)$). At the same time, we want to minimize the total amount of force used in terms of its squared norm $\int_0^T w(t)^2 \, \mathrm{d}t$. The constraint that $y(T) = \bar{y}$ is equivalent to*

$$\bar{y} = y(T) = \int_0^T (T - \tau) w(\tau) \, \mathrm{d}\tau = \langle x(\cdot), w(\cdot) \rangle \tag{5.22}$$

where $x(t) = T - t$. Thus, we are interested to solve

$$\textit{minimize} \quad \int_0^T w(t)^2 \, \mathrm{d}t \quad \textit{subject to} \quad \bar{y} = \langle x(\cdot), w(\cdot) \rangle. \tag{5.23}$$

This is a typical under-constrained problem where there are many solutions to the constraint but only one least norm solution. Use the application of the Projection Theorem as described in Section 5.2.2 to find the least norm solution, i.e., the minimizer to (5.23)*.*

We remark that the final displacement in (5.22) *can be derived using the relation between displacement, velocity and acceleration and some multivariable calculus gymnastics as follows*

$$\bar{y} = y(T) = \int_0^T \dot{y}(s) \, \mathrm{d}s = \int_0^T \int_0^s w(\tau) \, \mathrm{d}\tau \, \mathrm{d}s = \int_0^T \int_0^T w(\tau) \mathbb{1}\{\tau < s\} \, \mathrm{d}\tau \, \mathrm{d}s \tag{5.24}$$

$$= \int_0^T w(\tau) \int_0^T \mathbb{1}\{\tau < s\} \, \mathrm{d}s \, \mathrm{d}\tau = \int_0^T (T - \tau) w(\tau) \, \mathrm{d}\tau. \tag{5.25}$$

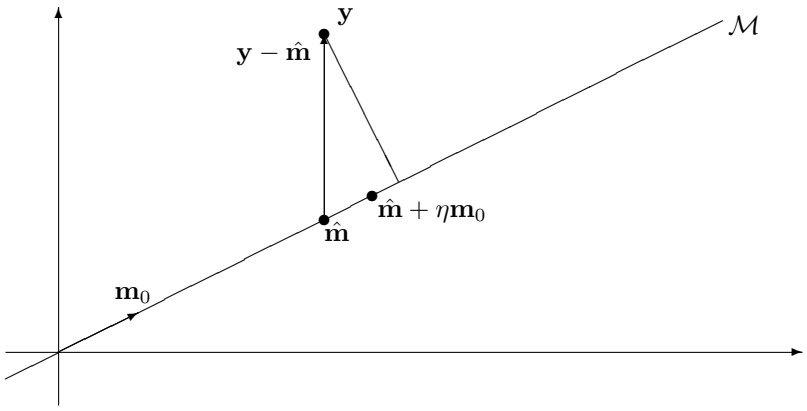


Figure 5.2: Illustration of the proof of the Projection Theorem. If $\mathbf{y} - \hat{\mathbf{m}}$ is not orthogonal to the subspace $\mathcal{M}$, shift $\hat{\mathbf{m}}$ in the direction $\mathbf{m}_0$ by an "amount" η. The distance to $\mathbf{y}$ will be reduced. In this figure, $\eta = (\mathbf{y} - \hat{\mathbf{m}})^\top \mathbf{m}_0 > 0$ as the angle between $\mathbf{y} - \hat{\mathbf{m}}$ and $\mathbf{m}_0$ is acute.

5.A Proof of the Projection Theorem

Proof of Theorem 5.1. Assume that $\mathcal{M} \neq \{\mathbf{0}\}$ otherwise the claim is immediate. Suppose, to the contrary, there exists some $\mathbf{m}_0 \in \mathcal{M} \setminus \{\mathbf{0}\}$ of unit norm such that $(\mathbf{y} - \hat{\mathbf{m}})^\top \mathbf{m}_0 =: \eta \neq 0$. Then, we claim that the vector $\hat{\mathbf{m}} + \eta\mathbf{m}_0 \in \mathcal{M}$ yields a better solution in the sense that its distance from $\mathbf{y}$ is strictly smaller. (Why can we assume $\|\mathbf{m}_0\| = 1$ and why is $\hat{\mathbf{m}} + \eta\mathbf{m}_0$ in the subspace $\mathcal{M}$?)

Consider,

$$\begin{aligned}
\|\mathbf{y} - (\hat{\mathbf{m}} + \eta\mathbf{m}_0)\|^2 &= \|(\mathbf{y} - \hat{\mathbf{m}}) - \eta\mathbf{m}_0\|^2 && (5.26)\\
&= \|\mathbf{y} - \hat{\mathbf{m}}\|^2 - 2\eta(\mathbf{y} - \hat{\mathbf{m}})^\top \mathbf{m}_0 + \eta^2 \|\mathbf{m}_0\|^2 && (5.27)\\
&\overset{(a)}{=} \|\mathbf{y} - \hat{\mathbf{m}}\|^2 - 2\eta^2 + \eta^2 \|\mathbf{m}_0\|^2 && (5.28)\\
&\overset{(b)}{=} \|\mathbf{y} - \hat{\mathbf{m}}\|^2 - 2\eta^2 + \eta^2 && (5.29)\\
&< \|\mathbf{y} - \hat{\mathbf{m}}\|^2, && (5.30)
\end{aligned}$$

where (a) is due the definition of η and (b) is because $\mathbf{m}_0$ has unit norm. Thus, we have found a vector in $\mathcal{M}$, namely $\hat{\mathbf{m}} + \eta\mathbf{m}_0$, that has strictly smaller distance to $\mathbf{y}$, contradicting the optimality of $\hat{\mathbf{m}}$. □

See Fig. 5.2 for an illustration of this proof.

Chapter 6

Ridge Regression, Linear Classification and Polynomial Regression

In this chapter, we will cover several important topics. First, we motivate the use of ridge regularization in the context of least squares estimation. We then derive the ridge regression solution. We will discuss both its primal and dual forms. The latter will allow us to segue to the (optional) topic of kernels. Second, we discuss how linear models can be used for classification. Finally, we consider a certain form of generalized linear models for regression known as polynomial regression.

6.1 Motivation for Ridge Regression

In the previous lectures, you learned about least squares estimation. Let us recap that. We assume here that we have more data points m than dimensions or attributes d. The data or training samples are denoted, as usual, as column vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \in \mathbb{R}^d$. For the sake of notational brevity, we omit the offset from our training samples. The corresponding target variables are denoted as $y_1, y_2, \ldots, y_m \in \mathbb{R}$. The data samples and the target variables are stacked in the *design matrix* and *target vector* as follows:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_m^\top \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,d} \end{bmatrix} \in \mathbb{R}^{m \times d} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m. \tag{6.1}$$

We aim to find a good *weight* or *coefficient vector* $\mathbf{w} \in \mathbb{R}^d$ such that $\mathbf{X}\mathbf{w}$ is "close to" $\mathbf{y}$. Note that the i^{th} row of $\mathbf{X}\mathbf{w}$, i.e., $\sum_{j=1}^{d} x_{i,j} w_j$ is the *prediction* of the target of the i^{th} data point. This error in predicting the i^{th} data point is denoted as

$$e_i := y_i - \sum_{j=1}^{d} x_{i,j} w_j \qquad 1 \le i \le m. \tag{6.2}$$

To ensure that all errors are small—or equivalently, $\mathbf{X}\mathbf{w}$ is close to $\mathbf{y}$—we seek to minimize the sum of squares of the e_i's over $\mathbf{w}$, i.e.,

$$\text{minimize} \quad \sum_{i=1}^{m} e_i^2. \tag{6.3}$$

It is clear that this is equivalent to

$$\text{minimize} \sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{d} x_{i,j} w_j \right)^2 \quad \Longleftrightarrow \quad \text{minimize} \sum_{i=1}^{m} \Big(y_i - [\mathbf{X}\mathbf{w}]_i \Big)^2 \quad \Longleftrightarrow \quad \text{minimize } \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 . \tag{6.4}$$

We saw that if $\mathbf{X}$ has full column rank, the solution for $\mathbf{w}$ is the least squares solution, denoted as

$$\hat{\mathbf{w}}_{\text{ls}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{6.5}$$

While this is elegant, there is substantial motivation to study refinements of the optimization problem in (6.3), or equivalently, (6.4). We state three reasons here.

1. Firstly, in modern data science applications, we often have data that is extremely high dimensional. In contrast, the number of data points is often small to moderate. Think of a clinical application where we have $m = 500$ subjects in our cohort. Getting a single person to be in a study is extremely costly. Indeed, we need to compensate the subject for her/his time. For each subject, however, we can collect a large amount of information using today's acquisition devices. Indeed, a single blood test can reveal a lot about a person's genomic information in the form of *single nucleotide polymorphisms* (SNPs). Furthermore, there are roughly 4 to 5 million SNPs in a person's genome and SNPs may give us information about whether a person is susceptible to certain diseases or ailments such as cancer. Thus d can be of the order of millions, which is certainly larger than the number of subjects in our study. As a result, $\mathbf{X}$ is now a wide matrix, which almost surely does not have full column rank and the solution (6.5) does not exist. While one can appeal to the least norm solution, there are better solutions that make explicit use of prior information about the solution.

2. Secondly, we often want to go beyond using linear models to do prediction because of the flexibility that such richer models affords us. We have seen from the lecture that we can use polynomial models (see Section 6.4) in which case the effective number of features is $\binom{p+d}{d}$ where p is the order of the polynomial (show this by looking up *monomial numbers* or refer to Appendix 6.A!). In fact, there is often motivation to use infinite-dimensional models, in which there are infinitely many features. In this case, the feature vectors can be *kernelized* using the so-called radial basis functions. Clearly, the new dimension (which may be even be infinite) exceeds that of the number of samples m and (6.5) is no longer applicable.

3. Finally, even if m is larger than d, we often want to *stabilize* and *robustify* the solution so that it *generalizes* well. By regularizing the optimization problems in (6.3) and (6.4), we often get more stable solutions that do not fluctuate as much under small (or even large) changes in the data. We will see this when we discuss about the *bias-variance tradeoff* in Chapter 7.

6.2 Ridge Regression

The way we achieve this stability or robustness in the least squares solution is to consider a *regularized* version of the optimization problem in (6.4). We consider the *ridge regression* or *Tikhonov regularization* problem

$$\text{minimize} \quad \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2, \tag{6.6}$$

where $\lambda > 0$ is known as the *regularization parameter*. Let us try to understand this a bit further. The first part of the objective function $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$ is the same; it forces the solution $\mathbf{w}$ to be such that the predictions contained in the vector $\mathbf{X}\mathbf{w}$ are close to the targets contained in $\mathbf{y}$. This is the so-called *data fidelity* or *risk* term. The second term in (6.6)—also known as the *regularization term*—forces the solution $\mathbf{w}$ to be small. This has the effect of stabilizing or robustifying the solution as large values of $\mathbf{w}$ do not lead to favorable generalization on new test examples.

6.2.1 Primal Form

To find the solution $\hat{\mathbf{w}}_\lambda$ to (6.6), we differentiate the objective function with respect to $\mathbf{w}$. By noting that $\nabla_{\mathbf{w}} \lambda \|\mathbf{w}\|^2 = 2\lambda \mathbf{w}$ and $\nabla_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = \nabla_{\mathbf{w}} (-2\mathbf{y}^\top \mathbf{X}\mathbf{w} + \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w}) = -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\mathbf{w}$ (see Appendix 4.A), we obtain

$$\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}_\lambda) - \lambda \hat{\mathbf{w}}_\lambda = \mathbf{0}. \tag{6.7}$$

In other words,

$$\hat{\mathbf{w}}_\lambda = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{6.8}$$

This is the ridge regression solution in *primal form*. Note that the matrix $\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}$ is always non-singular if $\lambda > 0$ because it is positive definite. (Indeed, $\mathbf{z}^\top (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}) \mathbf{z} = \|\mathbf{X}\mathbf{z}\|^2 + \lambda \|\mathbf{z}\|^2 > 0$ for any $\mathbf{z} \neq \mathbf{0}$.)

We note two extreme cases. If $\lambda \to 0^+$, then $\hat{\mathbf{w}}_\lambda$ reduces to $\hat{\mathbf{w}}_{\text{ls}}$. In this case, we place our entire faith in the data $(\mathbf{X}, \mathbf{y})$ and omit the regularization term. If $\lambda \to \infty$, then $\hat{\mathbf{w}}_\lambda$ reduces to the zero vector. In this case, we do not trust the data at all and our predictions of any test example is 0 because $\mathbf{x}_{\text{test}}^\top \hat{\mathbf{w}}_0 = \mathbf{0}$ for any $\mathbf{x}_{\text{test}} \in \mathbb{R}^d$.

6.2.2 Dual Form

Let us examine the optimization problem (6.6) in greater detail. The objective function can be written as

$$\sum_{i=1}^m (y_i - \mathbf{x}_i^\top \mathbf{w})^2 + \lambda \|\mathbf{w}\|^2. \tag{6.9}$$

Differentiating this meticulously, we obtain that

$$-2 \sum_{i=1}^m (y_i - \mathbf{x}_i^\top \mathbf{w}) \mathbf{x}_i + 2\lambda \mathbf{w} = \mathbf{0}. \tag{6.10}$$

Rearranging, we see that the solution $\hat{\mathbf{w}}_\lambda$ satisfies

$$\hat{\mathbf{w}}_\lambda = \frac{1}{\lambda} \sum_{i=1}^m \left(y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}_\lambda \right) \mathbf{x}_i. \tag{6.11}$$

For each $1 \leq i \leq m$, define the real number $a_i := \lambda^{-1} (y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}_\lambda)$. Hence, $\hat{\mathbf{w}}_\lambda$ can be written as

$$\hat{\mathbf{w}}_\lambda = \sum_{i=1}^m a_i \mathbf{x}_i \in \text{span}\big(\{\mathbf{x}_i\}_{i=1}^m \big). \tag{6.12}$$

While this is simple, the result in (6.12) is surprisingly deep and is a special instance of the *Representer Theorem* in statistical learning theory; see the seminal works by Kimeldorf and Wahba [KW70, Lemma 2.2] and Schölkopf, Herbrich and Smola [SHS01]. It says that no matter how high dimensional the data samples $\mathbf{x}_i$ are—they may even be infinite dimensional—we do not need to solve the high-dimensional optimization problem in (6.6) to find the optimal $\mathbf{w}$. All that is required to solve for $\hat{\mathbf{w}}_\lambda$ within a ridge regression framework is the set of m coefficients $\{a_i\}_{i=1}^m$! So we have reduced a d-dimensional optimization problem to one that is "only" m dimensional and m may be much smaller than d. Here, we note the importance of λ being positive. If $\lambda = 0$ (no regularization), this will no longer be true because (6.11) would not hold.

Now we notice from (6.12) that $\hat{\mathbf{w}}_\lambda$ can be written as

$$\hat{\mathbf{w}}_\lambda = \mathbf{X}^\top \mathbf{a} \tag{6.13}$$

where $\mathbf{a} = (a_1, a_2, \ldots, a_m)^\top \in \mathbb{R}^m$ is the vector of coefficients that defines the optimal ridge regularized weight vector $\hat{\mathbf{w}}_\lambda$. In other words, $\hat{\mathbf{w}}_\lambda$ is in the the column space (range) of $\mathbf{X}^\top$. Substituting this into (6.6), we obtain the optimization problem (in $\mathbf{a}$)

$$\begin{aligned} & \text{minimize} \quad \left\| \mathbf{y} - \mathbf{X}\mathbf{X}^\top \mathbf{a} \right\|^2 + \lambda \|\mathbf{X}^\top \mathbf{a}\|^2 \\ \iff \quad & \text{minimize} \quad \mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\mathbf{X}^\top \mathbf{a} + \mathbf{a}^\top (\mathbf{X}\mathbf{X}^\top)^\top (\mathbf{X}\mathbf{X}^\top) \mathbf{a} + \lambda \mathbf{a}^\top \mathbf{X}\mathbf{X}^\top \mathbf{a}. \end{aligned} \tag{6.14}$$

Differentiating and solving for $\mathbf{a}$ yields

$$\mathbf{X}\mathbf{X}^\top\mathbf{y} = (\mathbf{X}\mathbf{X}^\top)^\top(\mathbf{X}\mathbf{X}^\top)\mathbf{a} + \lambda\mathbf{X}\mathbf{X}^\top\mathbf{a} \tag{6.15}$$

$$= (\mathbf{X}\mathbf{X}^\top)(\mathbf{X}\mathbf{X}^\top)\mathbf{a} + \lambda(\mathbf{X}\mathbf{X}^\top)\mathbf{a} \tag{6.16}$$

$$= (\mathbf{X}\mathbf{X}^\top)\big(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\big)\mathbf{a}. \tag{6.17}$$

We note that (6.16) holds because $\mathbf{X}\mathbf{X}^\top$ is symmetric. The set of $\mathbf{a}$ that satisfies (6.18) is not a singleton if $\mathbf{X}$ does not have full row rank. In fact, the set of solutions $\mathbf{a}$ that satisfies (6.17) is

$$\Big\{\big(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\big)^{-1}\mathbf{y} + \mathbf{z} : \mathbf{z} \in \mathcal{N}(\mathbf{X}\mathbf{X}^\top)\Big\}. \tag{6.18}$$

However, we note from a simple argument[1] that $\mathcal{N}(\mathbf{X}\mathbf{X}^\top) = \mathcal{N}(\mathbf{X}^\top)$ and so substituting (6.18) into (6.13), we see that the *dual form* of the solution is

$$\hat{\mathbf{w}}_\lambda = \mathbf{X}^\top\mathbf{a} = \mathbf{X}^\top\Big(\big(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\big)^{-1}\mathbf{y} + \mathbf{z}\Big) = \mathbf{X}^\top\big(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}\big)^{-1}\mathbf{y}, \tag{6.19}$$

since $\mathbf{X}^\top\mathbf{z} = \mathbf{0}$. Even though $\mathbf{a}$ may not be unique, $\hat{\mathbf{w}}_\lambda$ is.

Note that the dual solution in (6.19) is really cool. Originally in the primal form of the solution in (6.8), we needed to invert a $d \times d$ matrix $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_d$. This costs $O(d^3)$ operations[2] (additions and multiplications) in general. As mentioned, d could be really large and could even be infinite. In the dual form, however, we (only) need to invert an $m \times m$ matrix $\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}_m$. This costs $O(m^3)$, which is much less than $O(d^3)$! We note that m is always finite because we always only have finite number of training samples.

In Appendix 6.B, we use the Woodbury matrix identity to show that the primal and dual forms of the least squares solution are equal for any $\lambda > 0$.

6.2.3 Interpretation In Terms of Kernels (Optional)

Substituting $\sum_{i=1}^m a_i\mathbf{x}_i$ for $\mathbf{w}$ in (6.9), we see that the optimization over $\mathbf{w}$ is equivalent to optimizing the following function over the vector $(a_1, a_2, \ldots, a_m) \in \mathbb{R}^m$:

$$J(a_1, a_2, \ldots, a_m) = \sum_{i=1}^m \left(y_i - \left(\sum_{j=1}^m a_i\mathbf{x}_j\right)^\top \mathbf{x}_i\right)^2 + \lambda\left\|\sum_{j=1}^m a_i\mathbf{x}_j\right\|^2 \tag{6.20}$$

$$= \sum_{i=1}^m \left(y_i - \sum_{j=1}^m a_i\mathbf{x}_j^\top\mathbf{x}_i\right)^2 + \lambda\sum_{i=1}^m\sum_{j=1}^m a_i a_j\mathbf{x}_i^\top\mathbf{x}_j \tag{6.21}$$

$$= \sum_{i=1}^m \left(y_i - \sum_{j=1}^m a_i K(\mathbf{x}_i, \mathbf{x}_j)\right)^2 + \lambda\sum_{i=1}^m\sum_{j=1}^m a_i a_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{6.22}$$

where $K(\mathbf{x}, \mathbf{x}') = \langle\mathbf{x}, \mathbf{x}'\rangle$ is the inner product between two vectors. This says that even if d is much larger than m, we simply have to solve a lower-dimensional optimization problem of m dimensions—the m components in $\mathbf{a}$—to obtain $\mathbf{w}$ in the form in (6.12). In fact, $\mathbf{a}$ is given exactly in (6.18). Furthermore, all that is needed in this endeavour are the pairwise inner products between the feature vectors $\mathbf{K} = [K(\mathbf{x}_i, \mathbf{x}_j)]$, a matrix of size $m \times m$. There is no need to "look at" the feature vectors; all we need are pairwise similarities between them. So we do not even need the $\mathbf{x}_i$'s to be in $\mathbb{R}^d$; they could be text or string data [LSST+02] or other categorical variables. There is a lot more we can say here, but we need to introduce the notion of *kernels* [SSB18], which unfortunately is beyond the scope of the course.

[1] The only non-trivial direction is $\mathcal{N}(\mathbf{X}\mathbf{X}^\top) \subset \mathcal{N}(\mathbf{X}^\top)$. Take $\mathbf{z} \in \mathcal{N}(\mathbf{X}\mathbf{X}^\top)$. Then $\mathbf{X}\mathbf{X}^\top\mathbf{z} = \mathbf{0}$. Pre-multiply by $\mathbf{z}^\top$ to obtain $\mathbf{z}^\top\mathbf{X}\mathbf{X}^\top\mathbf{z} = 0$. Hence, $\|\mathbf{X}^\top\mathbf{z}\|^2 = 0$ and so $\mathbf{X}^\top\mathbf{z} = \mathbf{0}$ or equivalently $\mathbf{z} \in \mathcal{N}(\mathbf{X}^\top)$.

[2] We say that $f(n) = O(g(n))$ if $\limsup_{n\to\infty} |f(n)/g(n)| < \infty$. Roughly speaking, when we say the number of operations is $O(d^3)$, it is $\leq cd^3$ for some large enough (but finite) constant $c > 0$.

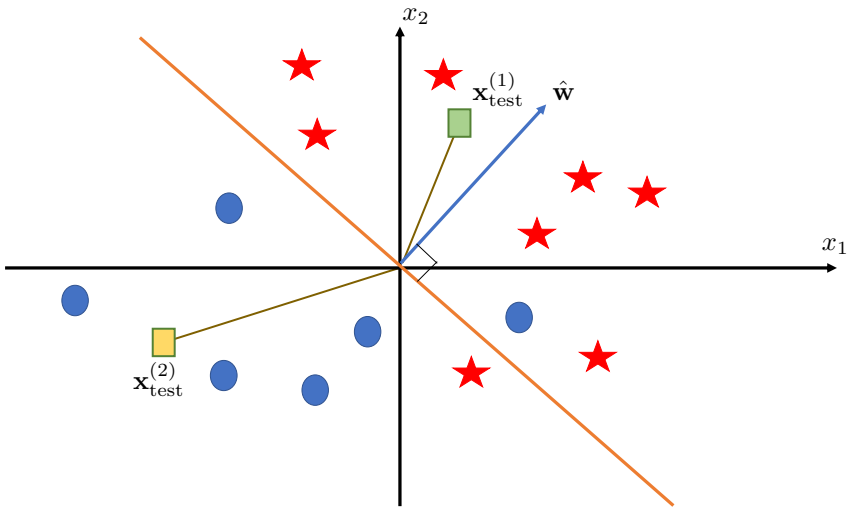


Figure 6.1: The positively (resp. negatively) labelled training points are the red stars (resp. blue circles). The learned linear classifier has normal vector given by $\hat{\mathbf{w}}$. If a given test point $\mathbf{x}_{\text{test}}^{(1)}$ (resp. $\mathbf{x}_{\text{test}}^{(2)}$) makes an acute (resp. obtuse) angle with $\hat{\mathbf{w}}$, it is classified as positively (resp. negatively) labelled.

6.3 Linear Models for Classification

Recall that in classification, the label y_i belongs to a finite set. We now briefly discuss how to extend the preceding framework into the realm of classification.

6.3.1 Binary Classification

In the simplest form of classification, one assumes that each y_i can only take on two values; this is called *binary classification*. The two values can be encoded in any reasonable way, but for the sake of concreteness, clarity, and simplicity, we assume that $y_i \in \{-1, +1\}$. The samples in $\mathcal{P} = \{\mathbf{x}_i : y_i = +1\}$ are from the *positive class* and the samples $\mathcal{N} := \{\mathbf{x}_i : y_i = -1\}$ are from the *negative class*.

Given what we have learned about linear regression thus far, the binary classification formulation and algorithm that we propose here is conceptually rather straightforward. The dataset $\mathcal{D} := \{(\mathbf{x}_i, y_i) : 1 \leq i \leq m\}$ is available to us where now $y_i \in \{-1, +1\}$. If we decide to use an offset (i.e., a bias term), then similarly to (6.1), we form the design matrix and target vector, except that we note that the target or label vector now lives in $\{-1, +1\}^m$. The same procedures—least squares, regularized least squares (ridge regression) in both the primal and dual forms—can be used to learn a weight vector $\mathbf{w} \in \mathbb{R}^{d'}$ where $d' = d$ if no offset is used or $d' = d + 1$ if an offset term is included into the model. Call the learned weight vector $\hat{\mathbf{w}}$.

Let us assume that there is no offset. For prediction purposes, we are given a new example $\mathbf{x}_{\text{test}} \in \mathbb{R}^d$ and would like to predict its class, i.e., whether it belongs to the positive or negative class. Our prediction is

$$\hat{y}_{\text{test}} = \text{sgn}\big(\mathbf{x}_{\text{test}}^\top \hat{\mathbf{w}}\big) \tag{6.23}$$

where sgn returns 1 of the argument is non-negative and -1 otherwise.[3]

While the formula in (6.23) is relatively simple, the interpretation is more important for the purposes of generalizing to the multiclass case in Section 6.3.2. What the rule in (6.23) is telling us is that we should declare the class of $\mathbf{x}_{\text{test}}$ to be positive (resp. negative) if its inner product with $\hat{\mathbf{w}}$ is positive (resp. negative), i.e., the angle it makes with $\hat{\mathbf{w}}$ is acute (resp. obtuse). Roughly speaking, this means that $\mathbf{x}_{\text{test}}$ is "more similar" (resp. "more dissimilar") to the samples in the positive class $\mathcal{P}$ (resp. the negative class $\mathcal{N}$). See Fig. 6.1 for an illustration of this explanation.

[3]The definition of sgn(0) is immaterial in most engineering applications.

6.3.2 Multiclass Classification

In real-world applications, y_i may take on many possibilities. For instance, the image classification example we discussed in Chapter 1 posits that each image contains one out of a possible $c = 10$ animals.[4] Thus, we must expand the scope of our discussion for linear classification. We perform one-hot encoding of the class label and this is stored in a *label matrix* $\mathbf{Y} = [y_{i,k}] \in \{0,1\}^{m \times c}$, where $y_{i,k} = 1$ means that training example $i \in \{1, 2, \ldots, m\}$ belongs to class $k \in \{1, 2, \ldots, c\}$. Note that each row of $\mathbf{Y}$ contains only one 1; the rest of the entries are 0. This is because we assume that each training example belongs to *exactly* one class.

In this framework, we aim to find $\mathbf{W} \in \mathbb{R}^{d \times c}$ such that $\mathbf{XW} \approx \mathbf{Y}$. To quantify the approximation error, we generalize (6.4) and minimize the sum of squares of all the entries in the error matrix $\mathbf{E} := \mathbf{Y} - \mathbf{XW}$; this is known as the square of the *Frobenius norm* of $\mathbf{E}$. That is, we consider the optimization problem

$$\text{minimize} \quad \|\mathbf{Y} - \mathbf{XW}\|_{\mathrm{F}}^2 = \sum_{i=1}^{m} \sum_{k=1}^{c} \left(y_{ik} - \sum_{j=1}^{d} x_{i,j} w_{j,k} \right)^2 . \tag{6.24}$$

Going through the same procedure to optimize over the matrix $\mathbf{W}$, we find that its least squares solution is

$$\hat{\mathbf{W}}_{\mathrm{ls}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y} \in \mathbb{R}^{d \times c}, \tag{6.25}$$

assuming again that $\mathbf{X}$ has full column rank. Given a new test sample $\mathbf{x}_{\mathrm{test}} \in \mathbb{R}^d$, analogously to the rule for binary classification in (6.23), we declare its class to be

$$\hat{y}_{\mathrm{test}} = \underset{k \in \{1,2,\ldots,c\}}{\arg\max} \; \mathbf{x}_{\mathrm{test}}^\top \hat{\mathbf{W}}_{\mathrm{ls}}(:, k), \tag{6.26}$$

where $\hat{\mathbf{W}}_{\mathrm{ls}}(:, k)$ denotes the k-th column of $\hat{\mathbf{W}}_{\mathrm{ls}}$. That is, we find the label $k \in \{1, 2, \ldots, c\}$ that maximizes the "correlation" or "similarity" between the new test sample $\mathbf{x}_{\mathrm{test}}$ and the weights corresponding to that class, i.e., $\hat{\mathbf{W}}_{\mathrm{ls}}(:, k)$.

6.4 Polynomial Regression and Classification

Up till now, our discussion has focused on *linear* models. However, there is significant motivation to go beyond linear models. Consider the dataset in Fig. 6.2, which is labelled as follows:

$$\begin{aligned}
\mathbf{x}_1 &= \begin{bmatrix} +1 & +1 \end{bmatrix}^\top & y_1 &= +1 \\
\mathbf{x}_2 &= \begin{bmatrix} -1 & +1 \end{bmatrix}^\top & y_2 &= -1 \\
\mathbf{x}_3 &= \begin{bmatrix} +1 & -1 \end{bmatrix}^\top & y_3 &= -1 \\
\mathbf{x}_4 &= \begin{bmatrix} -1 & -1 \end{bmatrix}^\top & y_4 &= +1.
\end{aligned} \tag{6.27}$$

For obvious reasons, this is known as the XOR problem. No matter how hard we try, we *cannot* find a weight vector (with offset term) $\mathbf{w} = (w_0, w_1, w_2)^\top \in \mathbb{R}^3$ such that all the training samples are classified correctly, i.e.,

$$y_i = \operatorname{sgn}\left(\begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix}^\top \mathbf{w} \right), \qquad \text{for all } \; 1 \le i \le 4. \tag{6.28}$$

This dataset is *not linearly separable*. However, notice that there is a simple nonlinear classifier that does the job. If we take the products of the first and second components of each feature vector, we obtain

$$x_{1,1} x_{1,2} = +1, \quad x_{2,1} x_{2,2} = -1, \quad x_{3,1} x_{3,2} = -1, \quad x_{4,1} x_{4,2} = +1. \tag{6.29}$$

[4] The well known ImageNet dataset contains $m \approx 14 \times 10^6$ images and $c \approx 2 \times 10^4$ labels or categories.

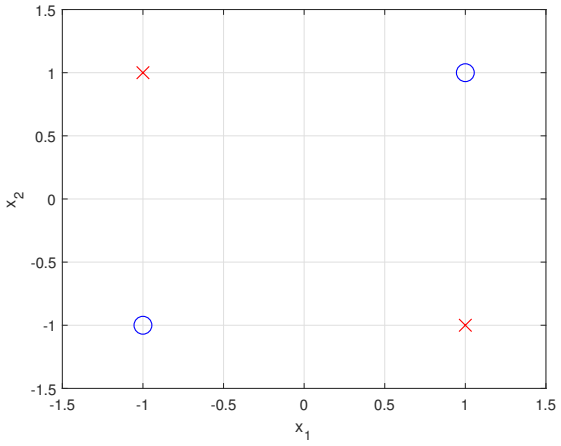


Figure 6.2: XOR example which is not linearly separable

Voila! Now we have transformed our dataset into one that is linearly separable because I can simply choose w_{12}, the weight associated to the product of the first and second components to be 1. Then it is easy to check that $\text{sgn}(w_{12}x_{i,1}x_{i,2}) = y_i$ for all $1 \leq i \leq 4$. We have achieved zero training error.

This simple observation motivates the use of products (or, in general, nonlinear functions) of individual features. Suppose our feature vectors are one-dimensional, i.e., $x_i \in \mathbb{R}$ for all $1 \leq i \leq m$. Then, for regression and similarly for classification, we might want to consider going beyond the linear model $y = w_0 + w_1 x$ to learn a richer model involving powers of the x, i.e.,

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_p x^p. \tag{6.30}$$

This is known as a *degree-p (univariate) polynomial* as the maximum power of the sole indeterminate x is p. In machine learning however, we have many features in most datasets. For the sake of discussion, say we have two features—x_1 and x_2. In this case, we have to introduce the notion of a *degree-p (bivariate and, in general, multivariate) monomial* which is a term of the form $x_1^a x_2^b$ where a and b are non-negative integers satisfying $a + b = p$. Hence, for two features x_1 and x_2, an *degree-2 (bivariate) polynomial* model is

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2. \tag{6.31}$$

Here the vector of weights to be learned is $\mathbf{w} = [w_0, w_1, w_2, w_{12}, w_{11}, w_{22}]^\top$. Note that for the XOR problem a $\mathbf{w}$ that works in the sense of achieving zero training error is $\mathbf{w} = [0, 0, 0, 1, 0, 0]^\top$. In effect, what this $\mathbf{w}$ does it to partition the (x_1, x_2) plane as in Fig. 6.2 into two parts—the first and third quadrant to be declared as the region that contains the samples from the positive class and the second and fourth quadrant as the region that contains negatively labelled samples.

While this theory allows us to learn complicated (more precisely, nonlinear) decision boundaries, the number of weights grows rapidly with the degree of the polynomial p and the original number of features d. In the above example with $p = d = 2$, we had a total of 6 weights to be learned—$w_0, w_1, w_2, w_{12}, w_{11}, w_{22}$. In general, one can prove that the total number of weights is $\binom{p+d}{d}$. However, there is an easy way to ameliorate this explosive growth in the number of weights; one exploits a theory known as *kernels* (discussed briefly in Section 6.2.3). This is somewhat similar to the discussion leading to the dual solution in Section 6.2.2.

6.5 Practice Problems

Exercise 6.1. *Show that* $\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_m$ *is non-singular for any* $\lambda > 0$.

Exercise 6.2. *Derive the solution to the following weighted ridge regression problem*

$$\text{minimize} \quad \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|_{\mathbf{q}}^2, \tag{6.32}$$

where $\|w\|_{\mathbf{q}}^2 = \sum_{i=1}^d q_i w_i^2$ *where* $q_i > 0$ *for all* i.

Exercise 6.3. *Derive the solution to the following shifted ridge regression problem*

$$\text{minimize} \quad \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w} - \mathbf{v}\|^2, \tag{6.33}$$

where $\mathbf{v} \in \mathbb{R}^d$ *is a fixed vector.*

Exercise 6.4. *Repeat Exercise 5.4 for the regularized least squares solution* $\hat{\mathbf{w}}_\lambda$. *That is, find the new regularized least squares solution in terms of the old one when one new data point is given.*

Exercise 6.5. *Let* $\mathbf{w}$ *be a random vector that is a multivariate Gaussian with zero mean and covariance matrix* $\lambda\mathbf{I}$. *Let* $\mathbf{y}$ *be a correlated Gaussian random vector dependent on* $\mathbf{x}$ *according to the linear model* $\mathbf{y} = \mathbf{X}\mathbf{w} + \mathbf{e}$ *where* $\mathbf{e}$ *is a multivariate Gaussian with zero mean and covariance matrix* $\mathbf{I}$. *Show that the maximum a posteriori (MAP) estimate of* $\mathbf{w}$ *given* $\mathbf{y}$ *(see* (3.13)*) is exactly the ridge regression solution in* (6.8). *This shows that the ridge regression solution can be interpreted as a Bayesian estimate of an unknown vector corrupted in Gaussian noise.*

Exercise 6.6. *As we have seen from Section 6.2.3, the dual solution can be interpreted in terms of the coefficients* $(a_1, a_2, \ldots, a_m)$. *To do so, we need to compute the kernel matrix* $K(\mathbf{x}, \mathbf{x}')$ *evaluated at two training points* $\mathbf{x}$ *and* $\mathbf{x}'$. *Suppose that we consider two-dimensional feature vectors* $\mathbf{x} = (x_1, x_2)$ *and* $\mathbf{x}' = (x_1', x_2')$ *and a polynomial feature vector of order two, i.e., a polynomial of the form* (6.31). *Show that if the polynomial features corresponding to* $\mathbf{x}$ *are given by* $\left[1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2\right]$, *the equivalent kernel* $K(\mathbf{x}, \mathbf{x}')$ *is* $(1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^2 = (1 + x_1x_1' + x_2x_2')^2$.

The point here is that for the purpose of polynomial regression, we do not have to form the high-dimensional feature vector $[1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]$; *all we have to do to compute the kernel matrix* $[K(\mathbf{x}_i, \mathbf{x}_j)]_{i,j}$ *is to compute* $(1 + \langle \mathbf{x}_i, \mathbf{x}_j \rangle)$ *(raised to the desired power) for each* i, j. *This demonstrates the power and utility of kernels.*

6.A Dimension of Feature Space of Polynomial Kernel (Optional)

The polynomial kernel (of order p) is defined as $K_p(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^p$. We claim that dimension of the feature space of the polynomial kernel $K_p(\cdot, \cdot)$ is $\binom{p+d}{d}$. Expanding the polynomial kernel using the binomial theorem we have $K_p(\mathbf{x}, \mathbf{x}') = \sum_{s=0}^p \binom{p}{s} (\langle \mathbf{x}, \mathbf{x}' \rangle)^s$. A possible feature space is given by all monomials of degree exactly s, namely $x_1^{i_1} x_2^{i_2} \ldots x_d^{i_d}$ where $i_j \in \mathbb{N}$ and $\sum_{j=1}^d i_j = s$.

So we just have to show that the number of monomials of degree at most p in d variables is $\binom{p+d}{p} = \binom{p+d}{d}$. This original constraint is $0 \leq \sum_{j=1}^d i_j \leq p$ where $(i_1, i_2, \cdots, i_d) \in \{0, 1, \cdots, p\}^d$. If we introduce $i_{d+1} \in \{0, 1, \cdots, p\}$, this is equivalent to $\sum_{j=1}^{d+1} i_j = p$. The term i_{d+1} takes care of the "slack" in the inequality. So it would be like dividing a row of p balls into $d+1$ partitions, which requires d dividers. Then it boils down to choosing which d of the $p + d$ objects to be the dividers, hence $\binom{p+d}{d}$.

6.B Equivalence of Primal and Dual Solutions (Optional)

Here, we show that for any $\lambda > 0$ and for any $(\mathbf{X}, \mathbf{y})$, we have

$$(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})^{-1} \mathbf{y}. \tag{6.34}$$

For this, we use the *Woodbury matrix identity*,

$$(\mathbf{I} + \mathbf{U}\mathbf{V})^{-1} = \mathbf{I} - \mathbf{U}(\mathbf{I} + \mathbf{V}\mathbf{U})^{-1}\mathbf{V}. \tag{6.35}$$

Starting from the expression on the right of (6.34), we have

$$\begin{aligned}
\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}\mathbf{y} &= \lambda^{-1}\mathbf{X}^\top(\mathbf{I} + \lambda^{-1}\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{y} && (6.36)\\
&= \lambda^{-1}\mathbf{X}^\top\left[\mathbf{I} - \lambda^{-1}\mathbf{X}(\mathbf{I} + \lambda^{-1}\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\right]\mathbf{y} && (6.37)\\
&= \lambda^{-1}(\mathbf{X}^\top\mathbf{y} - \mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}) && (6.38)\\
&= \lambda^{-1}(\mathbf{I} - \mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1})\mathbf{X}^\top\mathbf{y} && (6.39)\\
&= \lambda^{-1}\left[\mathbf{I} - (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1} + \lambda\mathbf{I}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\right]\mathbf{X}^\top\mathbf{y} && (6.40)\\
&= (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}, && (6.41)
\end{aligned}$$

where (6.37) follows from the Woodbury matrix identity with the identifications $\mathbf{U} \equiv \lambda^{-1}\mathbf{X}$ and $\mathbf{V} \equiv \mathbf{X}^\top$. This is an alternative proof, based solely on matrix algebra, that the primal and dual forms of the least squares solutions in (6.8) and (6.19) are the same.

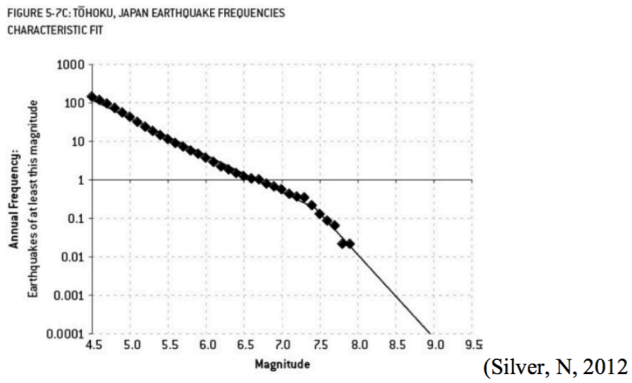
Chapter 7

Overfitting and the Bias-Variance Tradeoff

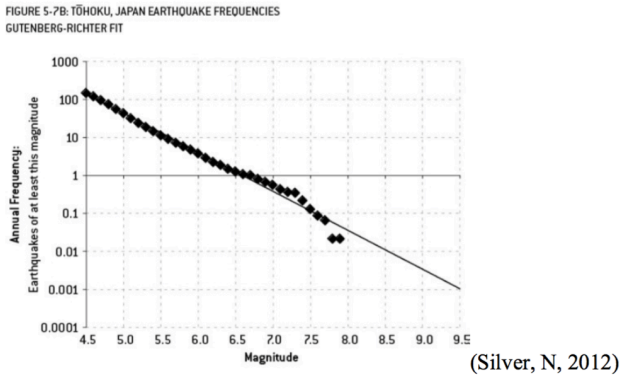
In this chapter, we motivate overfitting by provide a interesting real-life example illustrating its perils. We then use polynomial regression to describe overfitting and underfitting. Finally, we use linear regression to clearly illustrate the bias-variance tradeoff.

7.1 Beware of Overfitting!

As you have learned, in machine learning, we want to learn a model that fits the *pattern* of the data and not the data itself. Learning an overly complex model that "hugs" the data too tightly may result in overfitting. In this section, we describe an example in which overfitting has catastrophic consequences. This example, which should be of interest to the structural engineers in the class, is taken from Brian Stacey's report [Sta16], with additional analysis from Nate Silver [Sil12], whose book [Sil12] I strongly recommend.



(a) Overfitting



(b) Linear Fit (Gutenberg-Richter)

Figure 7.1: Plots of the predictions based on historical data

As you probably have heard of, in March 2011, there was a major earthquake in Japan that affected the Fukushima nuclear power plant. In fact, the design of the plant was based on historical earthquake data over more than 400 years. It was designed to withstand an earthquake of magnitude 8.6 (on the Richter scale). However, the earthquake in March 2011 had a magnitude of 9.0. Could this have been prevented with better

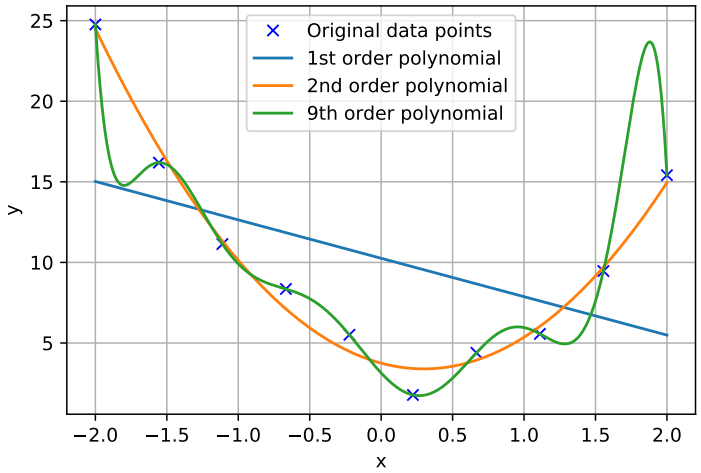


Figure 7.2: Polynomial regression with various orders

engineering? Perhaps if the engineers had taken NUS EE2211, they would have been in a better position to prevent the Fukushima disaster by using a more conservative and sturdy design for the plant.

The historical earthquake data is shown as scatter plots in Fig. 7.1. As can be seen, small earthquakes occur frequently while massive earthquakes occur rarely. The engineers saw the data and because of the kink around magnitude 7.3, they fitted a *polynomial* model (of order > 1), resulting in Fig. 7.1(a). Note that the regression curve "hugs" the points very closely. There is, however, another method known as the Gutenberg-Richter model which uses simple *linear* regression to predict frequency versus magnitude. It looks like Fig. 7.1(b). The models in Fig. 7.1(a) and Fig. 7.1(b) respectively say that an earthquake of magnitude 9.0 will occur on average once every 10^4 years and 500 years. Since the data is collected over 400 years, an earthquake of magnitude 9.0 is rather likely based on the second model. Hence, the overfitting error that the structural engineers made resulted in them designing a power plant that was not sufficiently strong to withstand a 9.0-earthquake, one that may occur once every 500 (as opposed to 10^4) years. This shows the importance of fitting the *pattern of the data*, and not the *actual data*.

7.2 Overfitting for Polynomial Regression

Now let us refer to a more concrete example based on what we have learned regarding polynomial regression in Section 6.4. In Fig. 7.2, we generated $m = 10$ data points from the one-dimensional quadratic model

$$y = 4x^2 - 2x + 3 + e, \tag{7.1}$$

where e is a zero-mean unit variance Gaussian random variable. More precisely, the points on the abscissa x are equally spaced between -2 and 2, while the points on the ordinate are generated according to (7.1) given the x's. The 10 points are plotted as blue crosses in Fig. 7.2. If we only observed these points, in general, we will not be able to know the order of the polynomial and, of course, the coefficients. We found and plotted the least squares curves assuming polynomials with orders 1, 2 and 9.

- For the polynomial of order 1, we are using affine functions of the form $y = w_0 + w_1 x$. The fit is not good as the model is too simple and the error between the training points and the learned line is bad. If we have a new data point x_{new}, we expect that the model fit is also bad. For example, if $x_{\text{new}} = 1.8$, the prediction is around 6.2 which is very far from the ground truth $4(1.8)^2 - 2(1.8) + 3 = 12.36$. This

Order	Fit on Training Data	Fit on Test Data	MSE	Remarks
1	Bad	Bad	High	Underfitting
2	Good	Good	Low	Correct Order
9	Perfect	Very Bad	High	Overfitting

Table 7.1: Summary of effect of various polynomial orders for the data points in Fig. 7.2

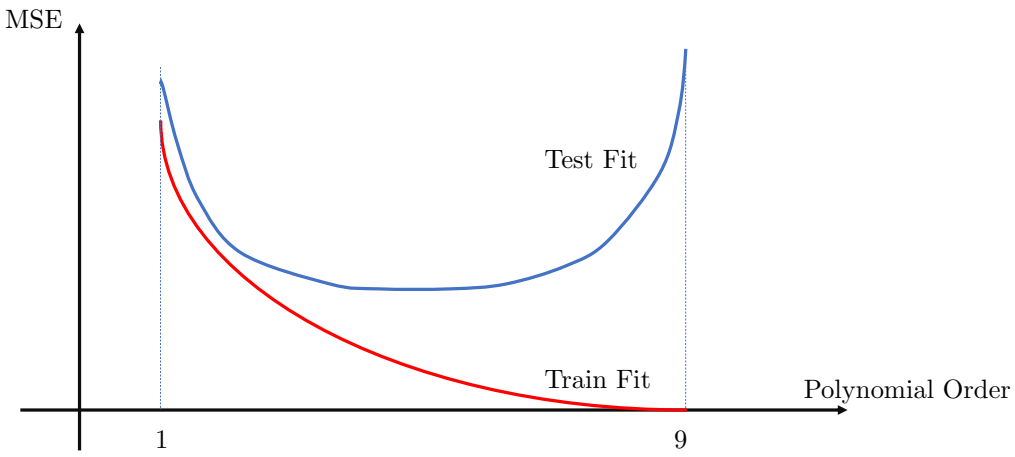


Figure 7.3: Schematic of the training and test errors in terms of the MSE with respect to polynomial order (complexity). The minimum of the test fit curve should be near the true polynomial order; in the case of Fig. 7.2, the optimal polynomial order is 2.

can be quantified more precisely in terms of the mean-squared error (MSE), which will be high. This is a case of underfitting.

- For the polynomial of order 9, we are using functions of the form $y = \sum_{i=0}^{9} w_i x^i$. Since there are only 10 data points, it can seen that the fit on the training points is perfect.[1] However, if we have a new data point x_{new}, we also expect that the model fit is very bad. For example if $x_{\text{new}} = 1.8$, the prediction is around 23, which is again very far from the ground truth 12.36. This is a case of severe overfitting, which may be disastrous as described in Section 7.1, leading to the Fukushima disaster.

- For the polynomial of order 2, we are using quadratic functions $y = w_0 + w_1 x + w_2 x^2$. The fit to the training data points is not bad; the errors are all reasonably small. More importantly, on new data points, we can see that the prediction is also good. This is unsurprising as the model we posit, a quadratic, is the same as the true one.

This discussion is summarized in Table 7.1 and Fig. 7.3. Thus, the point we want to make here is that the choice of the model is of paramount importance. If we choose a model that is either too simple or too complex, predictions made on new data points (that are not part of the training set) will be bad, i.e., the MSEs of predictions on new data points will be high. We should choose a model that fits the pattern of the data and not the actual data itself. One can choose the (polynomial) model by cross-validation, a topic we discuss in Chapter 10. The code used to generate Fig. 7.2 is available in Appendix 7.A. You can play with it to generate polynomials of other orders.

[1] This is because the training points are distinct and so the 10 by 10 polynomial design matrix, a so-called Vandermonde matrix, is non-singular.

7.3 Bias-Variance Tradeoff for Linear Models with Ridge Regression

Another way we can control the complexity of models is via regularization. In this section, we illustrate the use of regularization in controlling the mean-squared error. En route, we also discuss the bias-variance tradeoff. First, consider the linear model for d-dimensional data

$$y = w_0^* + w_1^* x_1 + \ldots + w_d^* x_d + e \tag{7.2}$$

where e is zero-mean noise with variance σ^2 (not necessarily Gaussian). We can also write (7.2) more compactly as

$$y = f(\mathbf{x}) + e \tag{7.3}$$

where $f(\mathbf{x}) = \tilde{\mathbf{x}}^\top \mathbf{w}^*$ and the *bias-augmented* sample is $\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \in \mathbb{R}^{d+1}$ and $\mathbf{w}^* = (w_0^*, w_1^*, \ldots, w_d^*)^\top \in \mathbb{R}^{d+1}$ is the unknown weight or coefficient vector. We observe samples from a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ for $1 \le i \le m$ are respectively the feature vectors and targets. Stacking these observations into matrix form, we obtain

$$\mathbf{y} = \mathbf{X}\mathbf{w}^* + \mathbf{e} \tag{7.4}$$

where $\mathbf{X} \in \mathbb{R}^{m \times (d+1)}$ is the design matrix with a vector of ones in the first column, $\mathbf{y} = (y_1, y_2, \ldots, y_m)^\top \in \mathbb{R}^m$ is the length-m vector of targets and $\mathbf{e} = (e_1, e_2, \ldots, e_m)^\top \in \mathbb{R}^m$ is the vector of i.i.d. noises. Note that $\mathbb{E}[\mathbf{e}] = \mathbf{0}$ and $\mathrm{Cov}(\mathbf{e}) = \sigma^2 \mathbf{I}$.

Suppose we do least squares regression with $\mathbf{X}$ assumed to be full column rank (usually the case when $m > d+1$). Then you know that the least squares estimate of the unknown $\mathbf{w}^*$ is

$$\hat{\mathbf{w}}_{\mathrm{ls}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \in \mathbb{R}^{d+1}. \tag{7.5}$$

Given a *new* (or *test*) sample $\mathbf{x} \in \mathbb{R}^d$, we can obtain its prediction as the following inner product

$$\hat{f}_{\mathcal{D}}(\mathbf{x}) = \tilde{\mathbf{x}}^\top \hat{\mathbf{w}}_{\mathrm{ls}} = \tilde{\mathbf{x}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}. \tag{7.6}$$

Let us evaluate the bias and variance of $\hat{f}_{\mathcal{D}}(\mathbf{x})$ in (7.6); note that $\hat{f}_{\mathcal{D}}(\mathbf{x})$ is only random through the noise $\mathbf{e}$ (not the dataset $(\mathbf{X}, \mathbf{y})$, the true coefficients $\mathbf{w}^*$, or the test sample $\mathbf{x}$). Also recall that the bias is $\mathrm{Bias}(\hat{f}_{\mathcal{D}}(\mathbf{x})) = \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})] - f(\mathbf{x})$. The term $\mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]$, which is an expectation (or colloquially an average) over all training datasets $\mathcal{D}$, was denoted as $\hat{f}_{\mathrm{avg}}(\mathbf{x})$ in the lecture notes. In the following calculation, we will keep the training samples $\mathbf{x}_i$ the same but only average over their noises e_i, or equivalently, their targets y_i. We have

$$\begin{aligned}
\mathrm{Bias}(\hat{f}_{\mathcal{D}}(\mathbf{x})) &= \mathbb{E}\big[\hat{f}_{\mathcal{D}}(\mathbf{x}) - f(\mathbf{x})\big] && (7.7)\\
&\overset{(7.6)}{=} \mathbb{E}\big[\tilde{\mathbf{x}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} - f(\mathbf{x})\big] && (7.8)\\
&\overset{(7.4)}{=} \mathbb{E}\big[\tilde{\mathbf{x}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top (\mathbf{X}\mathbf{w}^* + \mathbf{e}) - f(\mathbf{x})\big] && (7.9)\\
&\overset{(7.3)}{=} \mathbb{E}\big[\tilde{\mathbf{x}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} (\mathbf{X}^\top \mathbf{X}) \mathbf{w}^* + \tilde{\mathbf{x}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{e} - \tilde{\mathbf{x}}^\top \mathbf{w}^*\big] && (7.10)\\
&= \tilde{\mathbf{x}}^\top \mathbf{w}^* + \tilde{\mathbf{x}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbb{E}[\mathbf{e}] - \tilde{\mathbf{x}}^\top \mathbf{w}^* = 0, && (7.11)
\end{aligned}$$

so the least squares estimator is *unbiased*. We emphasize that the only quantity that is random in the above calculation is the noise $\mathbf{e}$. For the variance, we first note from (7.6) that $\hat{f}_{\mathcal{D}}(\mathbf{x}) = \tilde{\mathbf{x}}^\top \mathbf{w}^* + \tilde{\mathbf{x}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{e}$ and since $\tilde{\mathbf{x}}^\top \mathbf{w}^*$ is deterministic, the variance of $\hat{f}_{\mathcal{D}}(\mathbf{x})$ is that of $\tilde{\mathbf{x}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{e}$. Let $\mathbf{c} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \tilde{\mathbf{x}}$.

Then

$$\mathrm{Var}\big(\hat{f}_{\mathcal{D}}(\mathbf{x})\big) \overset{(7.6)}{=} \mathrm{Var}\big(\mathbf{c}^\top \mathbf{e}\big) \tag{7.12}$$

$$= \sum_{i=1}^{m} c_i^2 \mathrm{Var}(e_i) \tag{7.13}$$

$$= \|\mathbf{c}\|^2 \sigma^2 \tag{7.14}$$

$$= \big\|\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\tilde{\mathbf{x}}\big\|^2 \sigma^2 \tag{7.15}$$

$$= \tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\tilde{\mathbf{x}}\sigma^2 \tag{7.16}$$

$$= \tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X})^{-1}\tilde{\mathbf{x}}\sigma^2. \tag{7.17}$$

Note that (7.13) holds because the e_i's (noises added on to the training samples) are assumed to be independent. This result is intuitive because $\mathrm{Var}(\hat{f}_{\mathcal{D}}(\mathbf{x}))$ is proportional to σ^2 and as the number of training samples m increases, the design matrix $\mathbf{X}^\top\mathbf{X} = \sum_{i=1}^m \mathbf{x}_i\mathbf{x}_i^\top \in \mathbb{R}^{(d+1)\times(d+1)}$ (sum of rank-one outer products) also increases linearly and so $(\mathbf{X}^\top\mathbf{X})^{-1}$ decreases as $1/m$.[2] Recall that the bias-variance formula says that

$$\mathrm{MSE}(\hat{f}_{\mathcal{D}}(\mathbf{x})) = \mathbb{E}\big[(f(\mathbf{x}) + e - \hat{f}_{\mathcal{D}}(\mathbf{x}))^2\big] \tag{7.18}$$

$$= \big(\mathrm{Bias}(\hat{f}_{\mathcal{D}}(\mathbf{x}))\big)^2 + \mathrm{Var}(\hat{f}_{\mathcal{D}}(\mathbf{x})) + \text{Irreducible Noise}. \tag{7.19}$$

See Appendix 7.B for a proof of this formula. The three terms can be explained as follows:

- The *bias* quantifies the error caused by simplifying assumptions in the model. For example, when we use a linear function to approximate a model which is inherently quadratic, we will suffer some bias.
- The *variance* quantifies how much the estimated solution $\hat{f}_{\mathcal{D}}(\mathbf{x})$ fluctuates around its mean.
- The *irreducible error* quantifies the measurement noise that is inherent in the new test sample.

For the least squares predictor in (7.6), putting the bias in (7.11) and variance in (7.17) together, we see that

$$\mathrm{MSE}(\hat{f}_{\mathcal{D}}(\mathbf{x})) = \sigma^2\big(\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X})^{-1}\tilde{\mathbf{x}} + 1\big). \tag{7.20}$$

Now, we go a bit further and calculate explicit expressions for the bias and variance when we use ridge regression with regularization parameter $\lambda > 0$. In this case, we can jettison the assumption that $\mathbf{X}$ has full column rank and we have

$$\hat{\mathbf{w}}_\lambda = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y} \in \mathbb{R}^{d+1}. \tag{7.21}$$

The prediction of the target of a new sample $\mathbf{x}$ (another column vector), denoted as $\hat{f}_{\mathcal{D},\lambda}(\mathbf{x})$, is

$$\hat{f}_{\mathcal{D},\lambda}(\mathbf{x}) = \tilde{\mathbf{x}}^\top\hat{\mathbf{w}}_\lambda = \tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}, \tag{7.22}$$

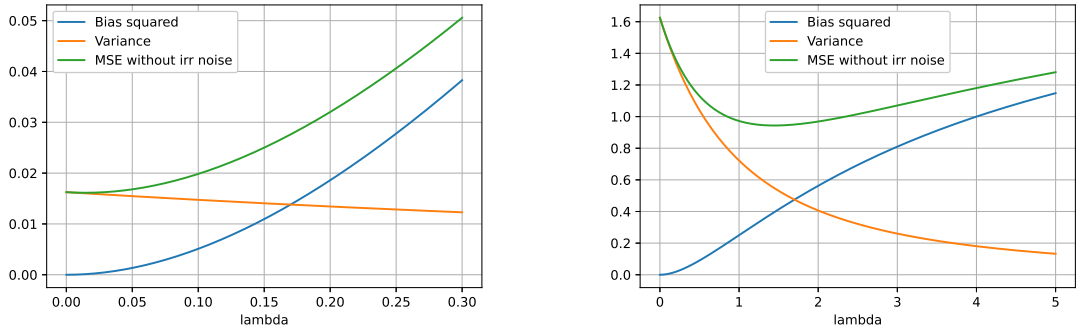
where $\tilde{\mathbf{x}} \in \mathbb{R}^{d+1}$ is the bias-augmented version of $\mathbf{x}$. Note that for $\lambda > 0$, the prediction is smaller than the unregularized case because of the additional term $+\lambda\mathbf{I}$ in the inverse. Thus, this is sometimes called *weight shrinkage*. Using similar but more tedious calculations (see Appendix 7.C), we obtain the bias and variance of the ridge regularized prediction in (7.22) as follows:

$$\mathrm{Bias}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x})) = -\lambda\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{w}^* \quad \text{and} \tag{7.23}$$

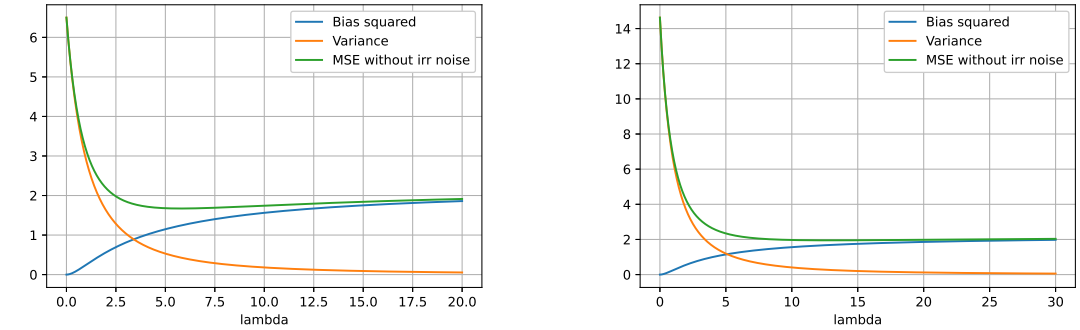
$$\mathrm{Var}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x})) = \tilde{\mathbf{x}}^\top\big((\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1} - \lambda(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-2}\big)\tilde{\mathbf{x}}\sigma^2. \tag{7.24}$$

Note that since $\mathrm{Bias}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x})) \neq 0$ in general, the regularized least squares solution is *biased*. We will see, however, that its variance is smaller than that of the unregularized solution. Indeed, by comparing (7.17) and (7.24), we see that for any $\sigma^2 > 0$, $\mathrm{Var}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x})) \leq \mathrm{Var}(\hat{f}_{\mathcal{D}}(\mathbf{x}))$ with equality if and only if $\lambda = 0$.

[2] More precisely, if $\mathbf{x}_i$ are sampled i.i.d. from a distribution $p(\mathbf{x})$, then as m grows, $\mathbf{X}^\top\mathbf{X}/m \to \mathbf{C}$ (in probability and almost surely) where $\mathbf{C} = \mathbb{E}[\mathbf{x}_i\mathbf{x}_i^\top]$ is the covariance matrix of $\mathbf{x}_i$ for any i ($\mathbf{C}$ does not depend on i because $\mathbf{x}_i$ are i.i.d.). Thus, roughly speaking, $(\mathbf{X}^\top\mathbf{X})^{-1}$ decays at a rate of $1/m$.



(a) $\sigma = 0.1$ (b) $\sigma = 1.0$



(c) $\sigma = 2.0$ (d) $\sigma = 3.0$

Figure 7.4: Plots of $\text{Bias}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x}))^2$ and $\text{Var}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x}))$ and their sum for various noise levels σ

λ	Fit on Training Data	Fit on Test Data	MSE	Remarks
0	Good	Bad	High	Overfitting
Moderate	Good	Good	Low	Correct Regularization
Too large	Bad	Bad	High	Underfitting

Table 7.2: Summary of effect of λ

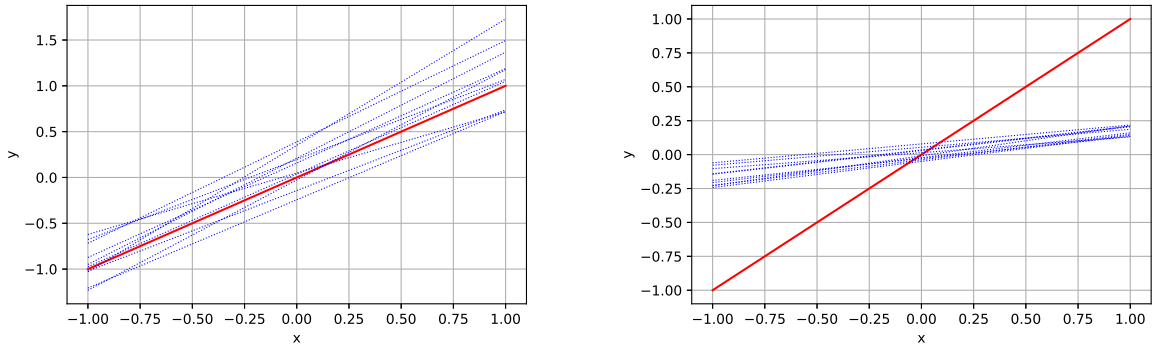
7.4 An Example of the Bias-Variance Tradeoff

We provide an example in which we can compute the bias and variance in (7.23) and (7.24) in closed form. Say $d = 1$ and we observe the two (scalar) training samples $\mathbf{x}_1 = -1$ and $\mathbf{x}_2 = 1$. Let $\mathbf{w}^* = [0, 1]^\top$ so the true model is nothing but the simple linear model $y = x + e$ (cf. Eqn. (7.3)). Then

$$\mathbf{X} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{X}^\top \mathbf{X} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} = \begin{bmatrix} 1/(2+\lambda) & 0 \\ 0 & 1/(2+\lambda) \end{bmatrix}. \tag{7.25}$$

Suppose our test sample is $\mathbf{x} = 1.5$, i.e., we are trying to predict the target of a point outside the range of values within our dataset. We plot $\text{Bias}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x}))^2$, $\text{Var}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x}))$ and their sum for different noise variances σ^2 in Fig. 7.4. A few remarks are in order:

- The bias is always zero when $\lambda = 0$ or when the least squares estimator in (7.5) is used; this is in line with (7.11). However, the squared bias grows as λ increases.
- The variance is initially large when $\lambda = 0$. However, it tends to zero as λ increases. This is in



(a) $\sigma = 0.3, \lambda = 0.1$ (b) $\sigma = 0.3, \lambda = 10$

Figure 7.5: Plots the predicted lines given 10 noise realizations

line with (7.24). In essence, with more regularization, the solution stabilizes and the variance of the prediction on a new test sample is reduced. It holds that $\mathrm{Var}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x})) \to 0^+$ as $\lambda \to 0^+$.

- The sum of the squared bias and variance, which is the mean-squared error minus the irreducible error (cf. Eqn. (7.19)), has a "sweet spot" at around $\lambda \approx 1.44$ for $\sigma = 1.0$.
- The effect of regularization is more pronounced for large noise, i.e., when σ is increased. Indeed, observe that the reduction of the sum of the squared bias and the variance from the case in which $\lambda = 0$ is more pronounced when the noise is larger. When the noise is large, there is severe overfitting and we are essentially fitting the curve to the noise. Hence, it is imperative to regularize the solution. For small σ, say $\sigma = 0.1$, the benefit of regularization is not obvious.

This discussion is summarized in Table 7.2. The code to generate Fig. 7.4 is provided in Appendix 7.D. You can play with it to generate other bias-variance plots.

In Fig. 7.5, we generate 10 datasets $\mathcal{D}$ as follows. Fixing $\mathbf{x}_1 = -1$ and $\mathbf{x}_2 = 1$, we generate 10 corresponding $\mathbf{y}$'s according to the true model in (7.3). Using the datasets, we learned the regression lines based on $\hat{\mathbf{w}}_\lambda$ in (7.21). Here, we fix the noise to be $\sigma = 0.3$ and consider two values of $\lambda \in \{0.1, 10\}$. For the small $\lambda = 0.1$, we notice that the regression lines have low bias on average; their averaged value is close to the ground truth red line $y = x$. However, they have large variability among themselves. For the large regularization parameter $\lambda = 10$, we see that the lines have high bias; they do not seem to approximate the ground truth very well indeed. However, among the 10 lines, they are close to one another and hence when λ is large, the variance of the solution is small. Finally, we see that for large λ, the intercept and slope of the regression lines are both small, which is intuitive as if λ is large $\hat{\mathbf{w}}_\lambda$ defined in (7.21) is small.

7.5 Practice Problems

Exercise 7.1. *Suppose we have m distinct training examples $\mathbf{x}_i$ in d dimensions. The target values y_i are related to the training examples as a polynomial of order p corrupted by some noise. What is the order of the polynomial we need to ensure zero training error? What happens if the points $\mathbf{x}_i$ are not distinct?*

Exercise 7.2. *In Section 7.3, we showed the bias-variance decomposition of the MSE. Show that in terms of the learned parameter $\hat{\mathbf{w}}$, we have*

$$\mathbb{E}\Big[\big\|\hat{\mathbf{w}} - \mathbf{w}^*\big\|^2\Big] = \big\|\mathbb{E}[\hat{\mathbf{w}}] - \mathbf{w}^*\big\|^2 + \mathbb{E}\Big[\big\|\hat{\mathbf{w}} - \mathbb{E}[\hat{\mathbf{w}}]\big\|^2\Big]. \tag{7.26}$$

The first term on the right-hand-side is squared bias in terms of estimating the parameter $\mathbf{w}$. It is the deviation of the expected value of the learned parameter $\hat{w}$ from the ground truth $\mathbf{w}^*$. The second term is the

variance. It represents the fluctuation of the learned parameter $\hat{\mathbf{w}}$ around its own expectation $\mathbb{E}[\hat{\mathbf{w}}]$. Why is there no irreducible error here?

Exercise 7.3. *In the bias-variance tradeoff formulas in* (7.19) *and* (7.26)*, the use of the squared ℓ_2 norm to measure the discrepancy between the estimated value $\hat{f}_{\mathcal{D}}(\mathbf{x})$ (or estimated parameter $\hat{\mathbf{w}}$) and target value $f(\mathbf{x}) + e$ (or target parameter $\mathbf{w}^*$) was important. If we use some other distance measure, the bias-variance formula would not hold in general. However, there is one other scenario in which the bias-variance formula would hold for a different discrepancy measure. This observation is due to Heskes [Hes98].*

The Kullback-Leibler (KL) divergence between two discrete distributions $\hat{p}$ and q is defined as

$$D_{\mathrm{KL}}(q \parallel \hat{p}) = \sum_{y \in \mathcal{Y}} q(y) \log \frac{q(y)}{\hat{p}(y)}. \tag{7.27}$$

Unlike the squared ℓ_2 norm, D_{KL} is not symmetric in its arguments. Nevertheless, we will prove a bias-variance formula. Suppose q in (7.27) *is a deterministic "target" distribution (analogous to $f(\mathbf{x}) + e$ and $\mathbf{w}^*$). Suppose $\hat{p}$ is a estimate of q (analogous to $\hat{f}_{\mathcal{D}}(\mathbf{x})$ and $\hat{\mathbf{w}}$), which is allowed to be random. Define $\bar{p}$ to be the following average distribution (analogous to $\mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]$ and $\mathbb{E}[\hat{\mathbf{w}}]$)*

$$\bar{p} = \operatorname*{arg\,min}_{r : r(y) \geq 0 \ \forall y \in \mathcal{Y}, \sum_{y \in \mathcal{Y}} r(y) = 1} \mathbb{E}\big[D_{\mathrm{KL}}(r \parallel \hat{p})\big]. \tag{7.28}$$

Prove the following bias-variance formula for the KL divergence

$$\mathbb{E}\big[D_{\mathrm{KL}}(q \parallel \hat{p})\big] = D_{\mathrm{KL}}(q \parallel \bar{p}) + \mathbb{E}\big[D_{\mathrm{KL}}(\bar{p} \parallel \hat{p})\big]. \tag{7.29}$$

Note that $D_{\mathrm{KL}}(q \parallel \bar{p})$ and $\mathbb{E}[D_{\mathrm{KL}}(\bar{p} \parallel \hat{p})]$ are analogous to the squared bias and variance respectively.

7.A Code to Generate the Learned Polynomials

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures

np.random.seed(0)

x = np.linspace(-2,2,10)
y = 4 *(x ** 2) - 2 * x + 3 + np.random.normal(0, 1, x.size)
plt.plot(x,y, 'bx', label='Original data points')

orders = np.array([1, 2, 9])
strs = ["1st", "2nd", "9th"]
x_vec = np.linspace(-2,2,1000)
i = 0

for j in orders:
    poly = PolynomialFeatures(degree=j)
    x = np.reshape(x, (x.size,1))
    P = poly.fit_transform(x)
    w = np.linalg.pinv(P).dot(y)
    plt.plot(x_vec, np.polyval(np.flip(w), x_vec), label=strs[i] + ' order polynomial')
    i = i+1

plt.grid()
```

```
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

7.B Proof of Bias-Variance Formula

We prove (7.19) here. We note that $f(\mathbf{x})$ is deterministic and $\hat{f}_{\mathcal{D}}(\mathbf{x})$ is random due to the randomness of $\mathcal{D}$. Clearly, e is also random. By expanding the right-hand-side of (7.18), we find that

$$\begin{aligned}
\mathrm{MSE}(\hat{f}_{\mathcal{D}}(\mathbf{x})) &= \mathbb{E}\left[\big(f(\mathbf{x}) + e - \hat{f}_{\mathcal{D}}(\mathbf{x}) + \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})] - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)^2\right] && (7.30)\\
&= \big(f(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)^2 + \mathbb{E}\left[\big(\hat{f}_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)^2\right] + \mathbb{E}[e^2] \\
&\quad + 2\big(f(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)\mathbb{E}[e] + 2\mathbb{E}\left[e\big(\hat{f}_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)\right] \\
&\quad + 2\big(f(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)\mathbb{E}\left[\hat{f}_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\right]. && (7.31)
\end{aligned}$$

We investigate the last three terms, which we will argue are all 0. First, because $\mathbb{E}[e] = 0$,

$$2\big(f(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)\mathbb{E}[e] = 0. \tag{7.32}$$

Second, because the noise on the test data point e is independent of the original dataset $\mathcal{D}$ and $\mathbb{E}[e] = 0$,

$$2\mathbb{E}\left[e\big(\hat{f}_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)\right] = 2\mathbb{E}[e]\mathbb{E}\left[\hat{f}_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\right] = 0. \tag{7.33}$$

Third, by linearity of expectation,

$$2\big(f(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)\mathbb{E}\left[\hat{f}_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\right] = 2\big(f(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)\left(\mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})] - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\right) = 0. \tag{7.34}$$

Thus, following (7.31), we have

$$\begin{aligned}
\mathrm{MSE}(\hat{f}_{\mathcal{D}}(\mathbf{x})) &= \big(f(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)^2 + \mathbb{E}\left[\big(\hat{f}_{\mathcal{D}}(\mathbf{x}) - \mathbb{E}[\hat{f}_{\mathcal{D}}(\mathbf{x})]\big)^2\right] + \mathbb{E}[e^2] && (7.35)\\
&= \big(\mathrm{Bias}(\hat{f}_{\mathcal{D}}(\mathbf{x}))\big)^2 + \mathrm{Var}(\hat{f}_{\mathcal{D}}(\mathbf{x})) + \sigma^2, && (7.36)
\end{aligned}$$

where $\mathbb{E}[e^2] = \mathrm{Var}(e) = \sigma^2$ because e has zero mean.

7.C Proofs of (7.23) and (7.24)

For the bias, we have

$$\begin{aligned}
\mathrm{Bias}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x})) &= \mathbb{E}\big[\hat{f}_{\mathcal{D},\lambda}(\mathbf{x}) - f(\mathbf{x})\big] && (7.37)\\
&\overset{(7.22)}{=} \mathbb{E}\big[\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y} - \tilde{\mathbf{x}}^\top\mathbf{w}^*\big] && (7.38)\\
&\overset{(7.4)}{=} \mathbb{E}\big[\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top(\mathbf{X}\mathbf{w}^* + \mathbf{e}) - \tilde{\mathbf{x}}^\top\mathbf{w}^*\big] && (7.39)\\
&= \mathbb{E}\big[\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^\top\mathbf{X}\mathbf{w}^* + \mathbf{X}^\top\mathbf{e}) - \tilde{\mathbf{x}}^\top\mathbf{w}^*\big] && (7.40)\\
&= \mathbb{E}\big[\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}((\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})\mathbf{w}^* - \lambda\mathbf{w}^* + \mathbf{X}^\top\mathbf{e}) - \tilde{\mathbf{x}}^\top\mathbf{w}^*\big] && (7.41)\\
&= \tilde{\mathbf{x}}^\top\mathbf{w}^* - \lambda\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{w}^* + \mathbb{E}\big[\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{e}\big] - \tilde{\mathbf{x}}^\top\mathbf{w}^* && (7.42)\\
&= -\lambda\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{w}^* && (7.43)
\end{aligned}$$

as desired. For the variance, we first note that the prediction can be simplified as follows:

$$\hat{f}_{\mathcal{D},\lambda}(\mathbf{x}) = \tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top(\mathbf{X}\mathbf{w}^* + \mathbf{e}) \tag{7.44}$$

$$= \tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{X}\mathbf{w}^* + \tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{e}. \tag{7.45}$$

The first part is non-random. So the variance of $\hat{f}_{\mathcal{D},\lambda}(\mathbf{x})$ is precisely that of the noise term $\tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{e}$. Let $\mathbf{c} = \mathbf{X}(\mathbf{X}^\top\mathbf{X}+\lambda\mathbf{I})^{-1}\tilde{\mathbf{x}}$. Then by the same steps as those for the variance of the unregularized case (see steps leading to (7.14)),

$$\mathrm{Var}(\hat{f}_{\mathcal{D},\lambda}(\mathbf{x})) = \|\mathbf{c}\|^2\sigma^2 \tag{7.46}$$

$$= \tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\tilde{\mathbf{x}}\sigma^2 \tag{7.47}$$

$$= \tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\big[(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}) - \lambda\mathbf{I}\big](\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\tilde{\mathbf{x}}\sigma^2 \tag{7.48}$$

$$= \tilde{\mathbf{x}}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\big(\mathbf{I} - \lambda(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\big)\tilde{\mathbf{x}}\sigma^2 \tag{7.49}$$

$$= \tilde{\mathbf{x}}^\top\big((\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1} - \lambda(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-2}\big)\tilde{\mathbf{x}}\sigma^2 \tag{7.50}$$

as desired.

7.D Code to Generate the Bias-Variance Plots

```
import numpy as np
import matplotlib.pyplot as plt

X = np.array([[1, -1], [1, 1]])
XTX = X.T @ X
N = 1000
lam = np.linspace(0.0001,5,N)
x_test = 1.5
x = np.array([1, x_test])
w_s = np.array([0, 1]).T
sigma = 1
bias = np.zeros(N)
var = np.zeros(N)

for i in range(0, len(lam)):
    reg = lam[i]*np.identity(2)
    bias[i] = -reg @ x @ np.linalg.inv(XTX + reg) @ w_s
    var[i] = x @ ( np.linalg.inv(XTX + reg) - lam[i] * np.linalg.inv(XTX + reg)
                  @ np.linalg.inv(XTX + reg))@x.T*sigma**2

MSE = np.power(bias,2)+var
plt.plot(lam, np.power(bias,2), label = 'Bias squared')
plt.plot(lam, var, label = 'Variance')
plt.plot(lam, MSE, label = 'MSE without irr noise')
plt.legend()
plt.grid()
plt.xlabel('lambda')
plt.savefig('bias_variance.eps', format='eps')
print(lam[np.argmin(MSE)])
```

Chapter 8

Loss Functions, Regularizers and Gradient Descent

In this chapter, we provide several examples of loss functions and regularizers in Section 8.1. We also illustrate some interesting features of the gradient descent algorithm in Section 8.2. You do not need to know the convergence analysis in the latter half of Section 8.2 (i.e., from Eqn. (8.5) onwards).

8.1 Loss Functions and Regularizers

In this module thus far, we are given dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ and we wish to find a vector of weights $\mathbf{w}$ that minimizes the "distance" between $\mathbf{Xw}$ and $\mathbf{y}$ where $\mathbf{X}$ is the design matrix (with or without an offset) and $\mathbf{y}$ is the vector of targets. How did we measure the "distance"? We considered the criterion

$$\text{minimize} \quad \sum_{i=1}^m (\mathbf{x}_i^\top \mathbf{w} - y_i)^2. \tag{8.1}$$

We can write each term as $\text{Loss}(f(\mathbf{x}_i, \mathbf{w}), y_i)$ where $\text{Loss}(\hat{y}, y) = (\hat{y} - y)^2$ is the l_2 loss and $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{x}_i^\top \mathbf{w}$ is a function of the i-th training sample $\mathbf{x}_i$ and the weight vector $\mathbf{w}$. In addition, to stabilize the solution, we added a *regularization term* or *regularizer* $R(\mathbf{w}) = \|\mathbf{w}\|^2 = \mathbf{w}^\top \mathbf{w} = \sum_{i=1}^d w_i^2$ to the objective function in (8.1). This resulted in the *ridge regression* problem

$$\text{minimize} \quad \sum_{i=1}^m (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 + \lambda \|\mathbf{w}\|^2 \qquad \Longleftrightarrow \qquad \text{minimize} \quad \|\mathbf{Xw} - \mathbf{y}\|^2 + \lambda \|\mathbf{w}\|^2. \tag{8.2}$$

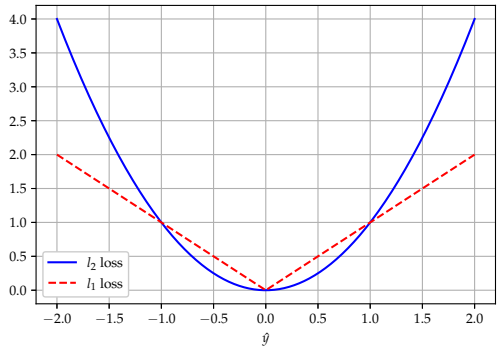
Recall that λ is known as the *regularization parameter*. Most machine learning problems can be stated as

$$\text{minimize} \quad \sum_{i=1}^m \text{Loss}(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w}), \tag{8.3}$$

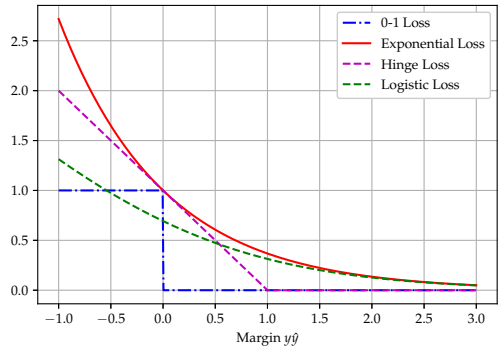
where $\text{Loss}(\hat{y}, y)$ is a loss function and $R(\mathbf{w})$ is a regularization term. Note that $f(\mathbf{x}_i, \mathbf{w})$ is the prediction of the target or class of $\mathbf{x}_i$ and so we will denote this as $\hat{y}_i$, or simply $\hat{y}$ if the sample is $\mathbf{x}$ (without an index).

Let us provide some examples of loss functions.

- The l_2 *loss* is $\text{Loss}(\hat{y}, y) = (\hat{y} - y)^2$. This loss function, used for regression, is perhaps the most common and is used for least squares problems as you have learned in EE2211.
- The l_1 *loss* is $\text{Loss}(\hat{y}, y) = |\hat{y} - y|$. This loss function, also used for regression, enforces robustness to outliers. To wit, the minimum of $\sum_{i=1}^m |w - y_i|$ over w is the *median* of the y_i's. The l_1 and l_2 losses are plotted in Fig. 8.1(a). Generally, for regression, loss functions measure the deviation of y and $\hat{y}$.



(a) l_2 and l_1 losses assuming $y = 0$



(b) 0-1, exponential, hinge, and logistic losses

Figure 8.1: Plots of various losses.

- The 0-1 *loss* is $\text{Loss}(\hat{y}, y) = \mathbf{1}\{y\hat{y} \leq 0\}$. This loss function is often a starting point in (binary) classification but is difficult to optimize analytically as it is non-convex and non-differentiable. Note that we incur a loss of 1 (resp. 0) if the signs of y and $\hat{y}$ do not coincide (resp. do coincide).
- The *exponential loss* is $\text{Loss}(\hat{y}, y) = \exp(-y\hat{y})$. This loss function is used in *boosting* [FS97], a popular algorithm for classification.
- The *hinge loss* is $\text{Loss}(\hat{y}, y) = \max\{0, 1 - y\hat{y}\}$. This loss function is used in *support vector machines* (SVMs) [Vap95], another popular algorithm for classification.
- The *logistic loss* is $\text{Loss}(\hat{y}, y) = \log[1 + \exp(-y\hat{y})]$. This loss function is used in *logistic regression*, which has a probabilistic interpretation.

The last four losses, used for classification, are plotted in Fig. 8.1(b). Notice that the losses are monotonically non-increasing in $y\hat{y}$, which can be interpreted as the *agreement* between the true target y and the estimated one $\hat{y}$. The more agreement there is, the smaller the loss. The agreement is also called *margin* in statistical learning theory. Observe from Fig. 8.1(b) that the exponential and hinge losses are *convex surrogates* of the 0-1 loss, i.e., convex functions that "approximate" the 0-1 loss.

Let us provide some examples of regularizers.

- The *ridge* or *Tikhonov* regularizer $R(\mathbf{w}) = \|\mathbf{w}\|^2 = \sum_{i=1}^{m} w_i^2$. This enforces that the solution's magnitude is small, enhances robustness and mitigates overfitting as you have seen in previous lectures. This is also used in SVMs.
- The *total variation* regularizer $R(\mathbf{w}) = \|\mathbf{w}\|_{\text{TV}}^2 = \sum_{i=1}^{m-1} (w_{i+1} - w_i)^2$ forces *adjacent* elements of $\mathbf{w}$ to be close to one another. It is particularly useful in signal smoothing or image denoising problems.
- The l_1 *regularizer* (or *Manhattan norm*) $R(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{i=1}^{m} |w_i|$ enforces sparsity in the elements of $\mathbf{w}$. This is useful in feature selection or extraction. It a cornerstone of *compressed sensing*, which used to be very popular. The l_2 loss coupled with the l_1 regularizer is called the *Lasso* [Tib96], which stands for *Least Absolute Shrinkage and Selection Operator*.

There is a lot more we can say about loss functions and regularizers (see [HTF09]) but we will not delve into these topics in any further detail in this course.

8.2 Gradient Descent Algorithm

In machine learning, besides the formulation of a suitable loss function with a regularization term, it is imperative to choose an optimization algorithm to optimize the chosen objective function. For the purposes

of this course, we will only discuss the ubiquitous *gradient descent* algorithm. It works as follows. We are given an objective function $C : \mathbb{R}^d \to \mathbb{R}$ and we would like to find

$$\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^d} C(\mathbf{w}). \tag{8.4}$$

We will assume that C is differentiable. Hence, if C includes the l_1 regularizer $\|\mathbf{w}\|_1$, it does not satisfy this differentiability assumption and we might have to use alternative methods such as subgradient or proximal gradient methods. The intuition for the gradient descent algorithm is the following. The gradient of C at $\mathbf{w}$, which is denoted as $\nabla_{\mathbf{w}} C(\mathbf{w})$ points in the direction that C *increases* the fastest; see Fig. 8.2. To minimize it, we want to walk in the direction in which f *decreases* the fastest, which is $-\nabla_{\mathbf{w}} C(\mathbf{w})$. Now that we have established a direction for us to walk, we need to decide how far to follow in this direction. This is dictated by our choice of the *step-size* or *learning rate*, which is usually denoted as η, a positive number. We walk in the direction $-\nabla_{\mathbf{w}} C(\mathbf{w})$ for a distance of η, stop and find a new direction to follow, walk along that direction for another possibly different length η', and so on. The iterative algorithm is summarized in Algorithm 1. The k-th iterate of the optimization variable is denoted as $\mathbf{w}^{(k)}$.

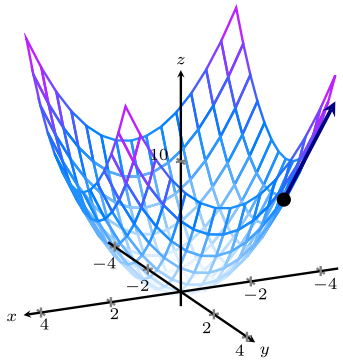


Figure 8.2: The gradient vector (indicated in the black arrow) points in the direction of steepest ascent.

There are many termination criteria one could use. For example, we could use the following criteria.

- Gradient becomes smaller than a threshold, i.e., $\|\nabla_{\mathbf{w}} C(\mathbf{w}^{(k)})\| \leq \varepsilon$;
- Difference in function values becomes smaller than a threshold, i.e., $|C(\mathbf{w}^{(k)}) - C(\mathbf{w}^{(k+1)})| \leq \varepsilon$;
- Difference in iterates becomes smaller than a threshold, i.e., $\|\mathbf{w}^{(k)} - \mathbf{w}^{(k+1)}\| \leq \varepsilon$;
- Iteration number exceeds a certain threshold.

In Algorithm 1, we adopt the first stopping criterion, which makes sense because at any optimal solution, the first-order optimality condition dictates that $\nabla_{\mathbf{w}} C(\mathbf{w}) = \mathbf{0}$.

An important design parameter here is the step-size η. If η is too small, we take too long to converge as the steps are too short. If η is too large, we may "overshoot" the minimum. Note that η can also depend on the iteration number k. In practice, on datasets with little structure, one makes "guesses" for an appropriate step-size but this is unsatisfactory from a theoretical standpoint. Can we do better? In this section, we show that indeed, at least for a small class of "nice" functions, we can analytically find the best step-size that minimizes the number of steps until which a certain convergence criterion is satisfied.

Let us consider the objective function given by the quadratic form

$$C(\mathbf{w}) = \frac{1}{2} \mathbf{w}^\top \mathbf{Q} \mathbf{w} \tag{8.5}$$

where $\mathbf{Q}$ is a positive definite (and symmetric) matrix. Clearly the optimal solution (why can we use "the" here?) is $\mathbf{w}^* = \mathbf{0}$. The gradient of C with respect to $\mathbf{w} = [w_1, w_2]^\top$ is $\nabla_{\mathbf{w}} C(\mathbf{w}) = \mathbf{Q}\mathbf{w}$. For concreteness in the subsequent discussion, say $\mathbf{Q}$ is the diagonal matrix

$$\mathbf{Q} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{8.6}$$

so the objective function is

$$C(\mathbf{w}) = \frac{1}{2}(\lambda_1 w_1^2 + \lambda_2 w_2^2) \tag{8.7}$$

Data: Differentiable function f; Step-size $\eta > 0$; Stopping threshold $\varepsilon > 0$
Result: An approximate solution to $\min_{\mathbf{w}} C(\mathbf{w})$
Initialize: Initial point $\mathbf{w}^{(0)}$; iteration count $k = 0$; flag notConverged = true
while notConverged **do**
  $\mathbf{w}^{(k+1)} \leftarrow \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} C(\mathbf{w}^{(k)})$;
  $k \leftarrow k + 1$;
  if $\|\nabla_{\mathbf{w}} C(\mathbf{w})\| < \varepsilon$ **then**
    notConverged = false;
  end
end

Algorithm 1: Gradient descent with constant step size

for some $\lambda_i > 0$ for $i = 1, 2$. Without loss of generality, assume $\lambda_2 \geq \lambda_1$. The iterative steps of the gradient descent algorithm are

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} C(\mathbf{w}^{(k)}) \tag{8.8}$$

$$= \mathbf{w}^{(k)} - \eta \mathbf{Q} \mathbf{w}^{(k)} \tag{8.9}$$

$$= (\mathbf{I} - \eta \mathbf{Q}) \mathbf{w}^{(k)}. \tag{8.10}$$

By noting that the matrix $\mathbf{I} - \eta \mathbf{Q}$ is also diagonal, we have

$$\mathbf{w}^{(k+1)} = \begin{bmatrix} 1 - \eta\lambda_1 & 0 \\ 0 & 1 - \eta\lambda_2 \end{bmatrix} \mathbf{w}^{(k)}. \tag{8.11}$$

If the algorithm is initialized at $\mathbf{w}^{(0)}$, then clearly (or by induction), the k-th iterate

$$\mathbf{w}^{(k)} = \begin{bmatrix} (1 - \eta\lambda_1)^k & 0 \\ 0 & (1 - \eta\lambda_2)^k \end{bmatrix} \mathbf{w}^{(0)}. \tag{8.12}$$

Hence, in order to maximize the the rate or speed of decay[1] of $\mathbf{w}^{(k)}$ to $\mathbf{0}$, we should solve the following optimization problem

$$\eta^* = \arg\min_{\eta > 0} \max \big\{ |1 - \eta\lambda_1|, |1 - \eta\lambda_2| \big\}. \tag{8.13}$$

This requires a moment's of thought.

The solution of (8.13) is $\eta^* = 2/(\lambda_1 + \lambda_2)$; see Fig. 8.3. This is the step-size that minimizes the number of iterations until $\|\mathbf{w}^{(k)}\|_2 < \varepsilon$ for any $\varepsilon > 0$. Substituting the optimal η^* into (8.12), we see that

$$\|\mathbf{w}^{(k)}\|_2 = \left(\frac{\lambda_2 - \lambda_1}{\lambda_2 + \lambda_1} \right)^k \|\mathbf{w}^{(0)}\|_2. \tag{8.14}$$

Observe that if $\lambda_1 = \lambda_2$ (isotropic), regardless of the initial point, gradient descent converges in one step using the optimal step-size. If $\lambda_1 \neq \lambda_2$, then to ensure that $\|\mathbf{w}^{(k)}\|_2 < \varepsilon$, we need

$$\left\lceil \frac{\log\left(\|\mathbf{w}^{(0)}\|_2 / \varepsilon \right)}{\log\left(\frac{\lambda_2/\lambda_1 + 1}{\lambda_2/\lambda_1 - 1} \right)} \right\rceil \quad \text{iterations.} \tag{8.15}$$

This is called *linear convergence* in the optimization literature because the dependence of the number of iterations on ε is $O(\log(1/\varepsilon))$.[2] The ratio $\lambda_2/\lambda_1 \in [1, \infty)$ is known as the *condition number*. The smaller it

[1] By "maximize the rate of decay", roughly speaking, we mean the minimizing the *contraction factor* $\rho \in (0, 1)$, i.e., the constant ρ satisfying $\|\mathbf{w}^{(k)}\| \leq \rho^k \|\mathbf{w}^{(0)}\|$ for k large enough. The smaller the ρ, the faster $\|\mathbf{w}^{(k)}\|$ converges to zero. For example, 0.99^k and 0.5^k require $k \geq 688$ and $k \geq 10$ to be less than 10^{-3}.

[2] This nomenclature appears strange initially but on log scales, the convergence as given by (8.14) is indeed linear.

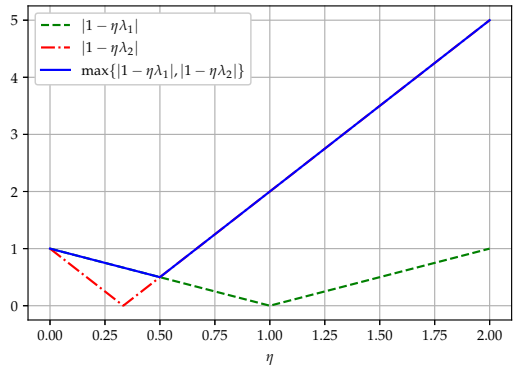
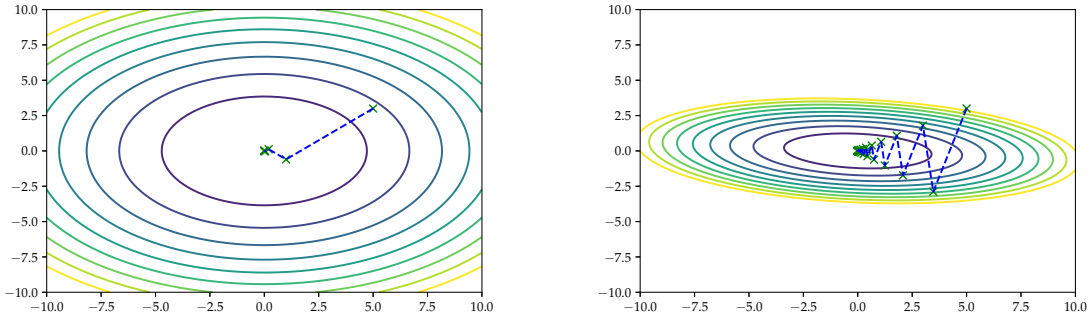


Figure 8.3: Illustration of (8.13) for $\lambda_1 = 1$ and $\lambda_2 = 3$. In this case, $\eta^* = 2/(1+3) = 0.5$.



(a) Well-conditioned example

(b) Ill-conditioned example

Figure 8.4: Illustration of gradient descent on two examples

is, the faster the convergence. Notice that $\lambda_i > 0$ for $i = 1, 2$ are the eigenvalues of the matrix $\mathbf{Q}$. In general (i.e., not necessarily $d = 2$ or diagonal or isotropic $\mathbf{Q}$), the formula one should use to minimize a quadratic form in (8.5) is

$$\eta^* = \frac{2}{\lambda_{\min}(\mathbf{Q}) + \lambda_{\max}(\mathbf{Q})}, \tag{8.16}$$

where $\lambda_{\min}(\mathbf{Q})$ and $\lambda_{\max}(\mathbf{Q})$ are respectively the minimum and maximum eigenvalues of $\mathbf{Q}$. This is true more generally (see Exercise 8.3) for so-called strongly convex functions with bounded Lipschitz gradients (strongly smooth); see recent work [TVT21] for a unified way of proving convergence rates of first-order optimization algorithms based on *sum-of-squares (polynomial) optimization*.

In Fig. 8.4, we show the iterates of gradient descent for two examples

$$\text{(a)}\ \ \mathbf{Q}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1.5 \end{bmatrix} \quad \text{and} \quad \text{(b)}\ \ \mathbf{Q}_2 = \begin{bmatrix} 2 & 1 \\ 1 & 15 \end{bmatrix}. \tag{8.17}$$

The contours (level sets) of $C_i(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{Q}_i \mathbf{w}$ for $i = 1, 2$ are also plotted. We use the optimal step size as given in (8.16). Our stopping criterion is $\|\mathbf{w}^{(k)}\|_2 < 10^{-5}$ and for both problems, we start from $\mathbf{w}^{(0)} = [5, 3]^\top$. We see that gradient descent on these problems "work" (in the sense of $\mathbf{w}^{(k)}$ converging to $\mathbf{0}$ as $k \to \infty$) but for the well-conditioned problem involving $\mathbf{Q}_1$, the convergence is much faster than the ill-conditioned problem involving $\mathbf{Q}_2$ (which is also non-isotropic). Indeed, Fig. 8.4(b) shows that the iterates exhibit inefficient zig-zag behavior. To get to $\|\mathbf{w}^{(k)}\|_2 < 10^{-5}$, the well-conditioned problem requires

only 9 iterations but the ill-conditioned problem requires 52 iterations. Optimization of ill-conditioned problems can be improved via pre-conditioning, conjugate gradient methods, or Newton's method (if C is twice differentiable and we have the computational resources). The code to generate Fig. 8.4 is provided in Appendix 8.A.

There is a lot more we can say about optimization theory and optimization algorithms [BV04] but unfortunately, the only algorithm that you will learn in EE2211 is gradient descent with constant step-size.

8.3 Practice Problems

Exercise 8.1. *Consider the univariate function $C(w) = 5w^2$ and the initial point $w_0 = 2$. Find the learning rate η so that gradient descent converges in one step. Find the set of learning rates such that gradient descent diverges. Finally, find the set of learning rates such that gradient descent converges.*

Exercise 8.2. *Show that the optimal step size for a general positive definite matrix $\mathbf{Q}$ is* (8.16).

Exercise 8.3. *One way to find the step size for a general function is to use use* exact line search

$$\eta = \arg\min_{s>0} f\left(\mathbf{w} - s\nabla_{\mathbf{w}} f(\mathbf{w})\right). \tag{8.18}$$

Show that if f is (μ, L)-strongly convex (i.e., $\mu\mathbf{I} \preceq \nabla^2_{\mathbf{w}} f(\mathbf{w}) \preceq L\mathbf{I}$ for $0 < \mu < L < \infty$), then gradient descent with exact line search converges linearly. Identify the rate of convergence.

8.A Code for Gradient Descent for Quadratic Functions

```
import numpy as np
import matplotlib.pyplot as plt

Q = np.array([[1, 0], [0, 1.5]])
[x1,x2] = np.meshgrid(np.linspace(-10,10,1001),np.linspace(-10,10,1001))
x_vals = np.linspace(-10, 10, 1001)
y_vals = np.linspace(-10, 10, 1001)
X, Y = np.meshgrid(x_vals, y_vals)
Z = Q[0,0]*X**2 + Q[1,1]*Y**2 + 2*Q[0,1]*X*Y
cp = plt.contour(X, Y, Z, np.linspace(0,200,10))

x_iter = np.array([[5], [3]]);
notConverged = 1;
lambdas, v = np.linalg.eig(Q)
eta = 2/(np.max(lambdas)+np.min(lambdas)) # optimal step size
iter = 0
x = np.zeros([2,1000])

while notConverged and iter < 1e3:
    plt.plot(x_iter[0],x_iter[1],'gx')
    x[:,iter] = x_iter.T
    x_iter = x_iter - eta*Q.dot(x_iter)
    if (np.linalg.norm(x_iter) < 1e-5):
        notConverged = 0
    iter = iter + 1

plt.plot(x[0,0:iter-1],x[1,0:iter-1],'b--')
```

Chapter 9

Decision Trees for Classification and Regression

In this chapter, we provide examples of the use of decision trees for classification and regression. We present all the steps slowly to aid understanding. It is my hope that this will bring some clarity to the algorithms, which are usually done using `tree.DecisionTreeClassifier()` and `tree.DecisionTreeRegressor()` in sklearn. It is good, and indeed essential, to know what exactly is going on in these powerful functions.

9.1 Decision Trees for Classification

Consider the dataset on whether we decide or not to play golf G. Here, there are four features or attributes—outlook O, temperature T, humidity H and wind W. There are a total of 15 labelled training samples.

	Outlook	**Temp**	**Humidity**	**Wind**	**Golf**
1	Rainy	Hot	High	False	No
2	Rainy	Hot	High	True	No
3	Overcast	Hot	High	False	Yes
4	Sunny	Mild	High	False	Yes
5	Sunny	Cool	Normal	False	Yes
6	Sunny	Cool	Normal	True	No
7	Overcast	Cool	Normal	True	Yes
8	Rainy	Mild	High	False	No
9	Rainy	Cool	Normal	False	Yes
10	Sunny	Mild	Normal	False	Yes
11	Rainy	Mild	Normal	True	Yes
12	Overcast	Mild	High	True	Yes
13	Overcast	Hot	Normal	False	Yes
14	Sunny	Mild	High	True	No
15	Sunny	Mild	Normal	True	Yes

We want to build a decision tree using the *(Shannon) entropy* impurity measure Q_m. For a node m,

$$Q_m = -\sum_{i=1}^{c} p_i \log p_i, \tag{9.1}$$

where c is number of classes, p_i is the fraction of samples in class i, and $\log(\,\cdot\,) = \log_2(\,\cdot\,)$.

1. Our first attribute to split is decided in the following way. First, we calculate the entropy of the class label G. We have

$$\begin{aligned} Q_G &= -p_{\text{Yes}} \log p_{\text{Yes}} - p_{\text{No}} \log p_{\text{No}} \\ &= -\frac{10}{15} \log \frac{10}{15} - \frac{5}{15} \log \frac{5}{15} = 0.9183. \end{aligned} \tag{9.2}$$

Now, we have to calculate the average conditional entropy of G given O, T, H and W. Let us show how to do this for the average conditional entropy of G given O. The outlook can take on 3 different values Rainy, Overcast and Sunny. We create the three sub-tables fixing the outlook to these three values. When we fix the outlook to Rainy, we get the following table.

	Outlook	**Temp**	**Humidity**	**Wind**	**Golf**
1	Rainy	Hot	High	False	No
2	Rainy	Hot	High	True	No
8	Rainy	Mild	High	False	No
9	Rainy	Cool	Normal	False	Yes
11	Rainy	Mild	Normal	True	Yes

Thus, the conditional entropy of G given that $O = \text{Rainy}$ is

$$\begin{aligned} Q_{G|\text{Rainy}} &= -p_{\text{Yes}|\text{Rainy}} \log p_{\text{Yes}|\text{Rainy}} - p_{\text{No}|\text{Rainy}} \log p_{\text{No}|\text{Rainy}} \\ &= -\frac{2}{5} \log \frac{2}{5} - \frac{3}{5} \log \frac{3}{5} = 0.971. \end{aligned} \tag{9.3}$$

When we fix the outlook to Overcast, we get the following table.

	Outlook	**Temp**	**Humidity**	**Wind**	**Golf**
3	Overcast	Hot	High	False	Yes
7	Overcast	Cool	Normal	True	Yes
12	Overcast	Mild	High	True	Yes
13	Overcast	Hot	Normal	False	Yes

Thus, the conditional entropy of G given that $O = \text{Overcast}$ is

$$\begin{aligned} Q_{G|\text{Overcast}} &= -p_{\text{Yes}|\text{Overcast}} \log p_{\text{Yes}|\text{Overcast}} - p_{\text{No}|\text{Overcast}} \log p_{\text{No}|\text{Overcast}} \\ &= -\frac{4}{4} \log \frac{4}{4} - \frac{0}{4} \log \frac{0}{4} = 0. \end{aligned} \tag{9.4}$$

Since all the labels here are the same (Yes), this is a *pure* node.

Finally when we fix the outlook to Sunny, we get the following table.

	Outlook	**Temp**	**Humidity**	**Wind**	**Golf**
4	Sunny	Mild	High	False	Yes
5	Sunny	Cool	Normal	False	Yes
6	Sunny	Cool	Normal	True	No
10	Sunny	Mild	Normal	False	Yes
14	Sunny	Mild	High	True	No
15	Sunny	Mild	Normal	True	Yes

Thus, the conditional entropy of G given that $O = \text{Sunny}$ is

$$\begin{aligned} Q_{G|\text{Sunny}} &= -p_{\text{Yes}|\text{Sunny}} \log p_{\text{Yes}|\text{Sunny}} - p_{\text{No}|\text{Sunny}} \log p_{\text{No}|\text{Sunny}} \\ &= -\frac{4}{6} \log \frac{4}{6} - \frac{2}{6} \log \frac{2}{6} = 0.9183. \end{aligned} \tag{9.5}$$

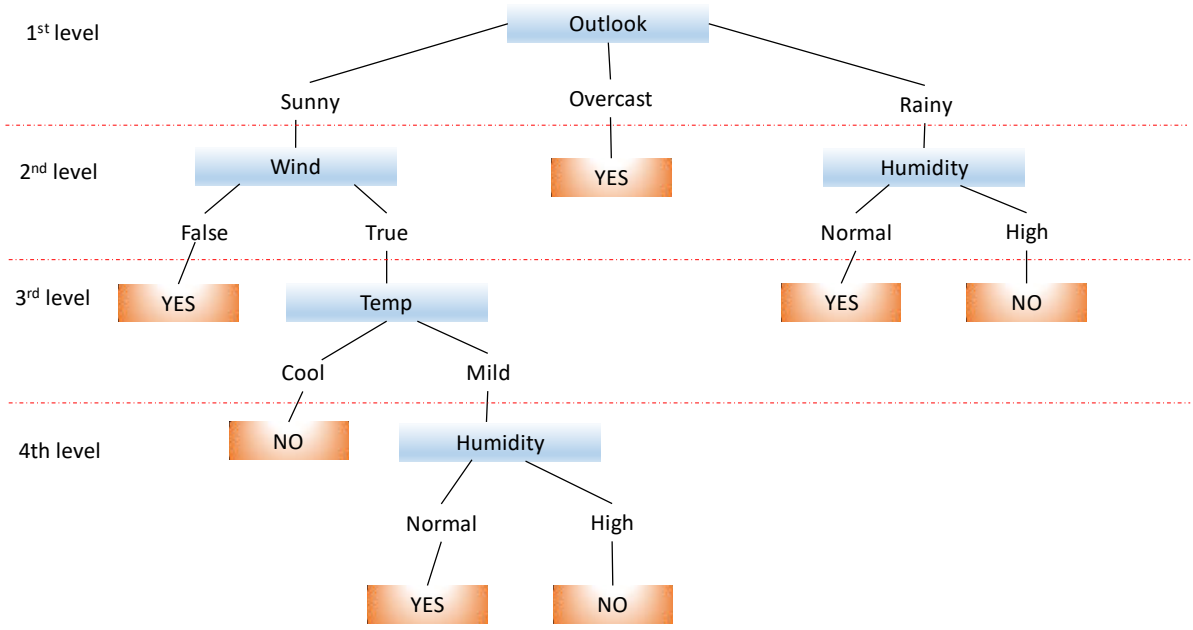


Figure 9.1: Decision Tree for the golf example. Terminal (leaf) nodes where decisions are made are indicated in orange.

The conditional entropy of G given O is then

$$\begin{aligned} Q_{G|O} &= p_{\text{Rainy}} Q_{G|O=\text{Rainy}} + p_{\text{Overcast}} Q_{G|O=\text{Overcast}} + p_{\text{Sunny}} Q_{G|O=\text{Sunny}} \\ &= \frac{5}{15} \times 0.9710 + \frac{4}{15} \times 0 + \frac{6}{15} \times 0.9183 = 0.6910. \end{aligned} \tag{9.6}$$

Thus, the gain of the attribute O over class G is

$$\text{Gain}(O; G) = Q_G - Q_{G|O} = 0.9183 - 0.6910 = 0.2273. \tag{9.7}$$

In a similar way, we can compute the gains of various other attributes over class G as

$$\text{Gain}(T; G) = 0.0325, \qquad \text{Gain}(H; G) = 0.1686, \qquad \text{Gain}(W; G) = 0.0258. \tag{9.8}$$

Make sure you know how to compute these based on the calculation above. Note that these gains are always *non-negative*, or equivalently, $Q_{G|a} \leq Q_G$ for attributes $a \in \{O, T, H, W\}$. In other words, the impurity can only go down as we split nodes. We provide a proof of this in Appendix 9.A.

Since $\text{Gain}(O; G)$ yields the largest gain over all features, the first split will be based on attribute O. So at this point, we have the tree shown in Fig. 9.1 but up to the 1st level only. Note that since overcast is a pure node, if it is an overcast day, G is definitely "Yes" and we will definitely play golf.

2. Now, we have to continue splitting the tree. First, we look at the left branch where it is definitely Sunny. Removing the Sunny variable, we have the following table.

   The entropy of G for this new table is

$$\begin{aligned} Q_G &= -p_{\text{Yes}} \log p_{\text{Yes}} - p_{\text{No}} \log p_{\text{No}} \\ &= -\frac{4}{6} \log \frac{4}{6} - \frac{2}{6} \log \frac{2}{6} = 0.9183. \end{aligned} \tag{9.9}$$

	Temp	Humidity	Wind	Golf
4	Mild	High	False	Yes
5	Cool	Normal	False	Yes
6	Cool	Normal	True	No
10	Mild	Normal	False	Yes
14	Mild	High	True	No
15	Mild	Normal	True	Yes

For the Temp variable,

$$Q_{G|\text{Mild}} = 0.8113 \quad \text{and} \quad Q_{G|\text{Cool}} = 1 \quad \Longrightarrow \quad Q_{G|T} = \frac{4}{6}Q_{G|\text{Mild}} + \frac{2}{6}Q_{G|\text{Cool}} = 0.8742. \tag{9.10}$$

For the Humidity variable,

$$Q_{G|\text{High}} = 1 \quad \text{and} \quad Q_{G|\text{Normal}} = 0.8113 \quad \Longrightarrow \quad Q_{G|H} = \frac{2}{6}Q_{G|\text{High}} + \frac{4}{6}Q_{G|\text{Normal}} = 0.8742. \tag{9.11}$$

For the Wind variable

$$Q_{G|\text{False}} = 0 \quad \text{and} \quad Q_{G|\text{True}} = 0.9183 \quad \Longrightarrow \quad Q_{G|W} = \frac{3}{6}Q_{G|\text{False}} + \frac{3}{6}Q_{G|\text{True}} = 0.4592. \tag{9.12}$$

The gains are

$$\text{Gain}(T; G) = 0.0441, \qquad \text{Gain}(H; G) = 0.0441, \qquad \text{Gain}(W; G) = 0.4592. \tag{9.13}$$

Hence, here we split according to the Wind variable, resulting in the left branch of the 2nd level.

3. Now, we have to split the right branch where it is definitely Rainy. Removing the Rainy variable, we have the following table.

	Temp	Humidity	Wind	Golf
1	Hot	High	False	No
2	Hot	High	True	No
8	Mild	High	False	No
9	Cool	Normal	False	Yes
11	Mild	Normal	True	Yes

Here, it is obvious that we should split according to the Humidity variable, resulting in the right branch of the 2nd level.

4. Observe that when it is sunny and Windy = False, the node is pure and we declare that we will definitely play golf. Otherwise, we have to consider the following table. In this case, both attributes will result in the same gain (verify) and it does not matter which attribute we choose to split. Say we choose Temp. Then the resultant tree is given the tree truncated to the 3rd level. Finally, we examine the final attribute Humidity resulting in the tree up to the 4th level.

	Temp	Humidity	Wind	Golf
6	Cool	Normal	True	No
14	Mild	High	True	No
15	Mild	Normal	True	Yes

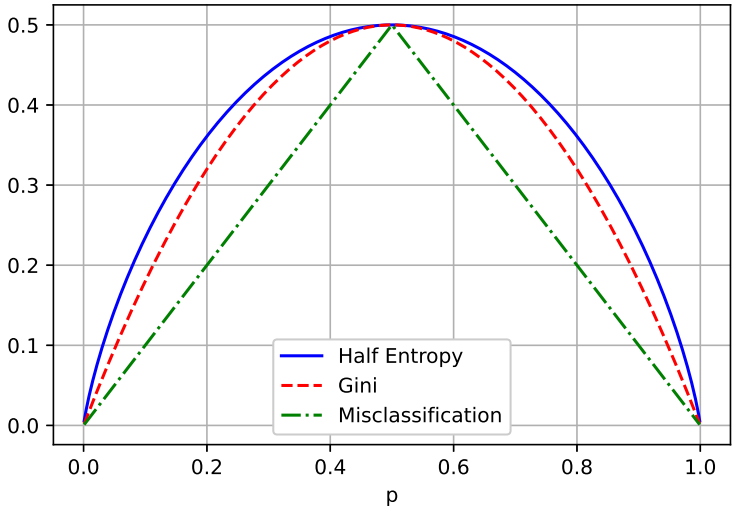


Figure 9.2: Comparison of impurity measures for the two-class case

Some remarks are in order. First, in practice, we may want to limit the maximum depth of the tree and not mandate that all leaf nodes are pure. We can also prune the tree after learning it. These methods serve to prevent overfitting. Second, observe that small changes in the dataset will result in dramatically different trees; this is often undesirable as we want our solutions in machine learning tasks to be stable and robust. Thus, one typically improves the robustness of decision trees by *bootstrapping* or using *random forests*.

Besides the entropy in (9.1), there are many other impurity measures we can use such as the *Gini impurity measure*

$$Q_m := 1 - \sum_{i=1}^{c} p_i^2 \tag{9.14}$$

or the *misclassification rate*

$$Q_m := 1 - \max_{i=1,\dots,c} p_i. \tag{9.15}$$

Here, c again denotes the number of classes. These impurity measures are plotted in Fig. 9.2 for binary (Bernoulli) distributions parameterized by a single value $p \in [0, 1]$, which represents the proportion of samples in any one of the classes. Note that when the distribution is pure (i.e., $p = 0$ or $p = 1$) the impurity measure is equal to 0. When the distribution has its most uncertainty or is most impure $p = 1/2$, and we see that the impurity measures also attain their maximal values.

We note that all these measures are simple functions of a broader family of entropy measures known as *Rényi entropies*

$$H_\alpha(X) = \frac{1}{1-\alpha} \log \sum_{i=1}^{c} p_i^\alpha \tag{9.16}$$

where $\alpha \geq 0$ and $\alpha \neq 1$ is the parameter of the Rényi entropy. Note that when $\alpha \to 1$, we recover the Shannon entropy $H(X) = H_1(X) = -\sum_{i=1}^{c} p_i \log p_i$ by invoking L'Hospital's rule. When $\alpha = 2$, we obtain the *collision entropy* $H_2(X) = -\log \sum_{i=1}^{c} p_i^2$. When $\alpha \to \infty$, we get the *min-entropy* $H_\infty(X) = -\log \max_{i=1,\dots,c} p_i$. These quantities measure uncertainty and so its not surprising that they are used for assessing impurity in our context. They have many other applications in other areas, not least in the study of information theory and cryptography.

9.2 Decision Trees for Regression

Now we consider using decision trees for regression. Consider the following (very) hypothetical dataset of house prices in Singapore. The target variable is Price P and the attributes are House Size S and Number of Rooms R.

	House Size ('000 sq ft)	**Num of Rooms**	**Price ('000,000 SGD)**
1	0.5	2	0.19
2	0.6	1	0.23
3	1.0	3	0.28
4	2.0	5	0.42
5	3.0	4	0.53
6	3.2	6	0.75
7	3.8	7	0.80

Note that I have arranged the data points in increasing order of P, which so happens to be increasing order of S as well. However, this is not the same order as that of R.

In regression, we use the mean squared error (MSE) for each node. The MSE for a node m with samples $\{y_i : 1 \leq i \leq J_m\}$ is

$$Q_m = \frac{1}{J_m} \sum_{i=1}^{J_m} (y_i - \hat{\mu}_m)^2 \quad \text{where} \quad \hat{\mu}_m = \frac{1}{J_m} \sum_{i=1}^{J_m} y_i. \tag{9.17}$$

The overall MSE is $Q_P = 0.0520$. For regression problems, we also have to determine the threshold to split each attribute into two. Let's focus on the House Size attribute S. If we set the threshold at $\tau = 0.75$, then the targets of the two classes are $\{0.19, 0.23\}$ and $\{0.28, 0.42, 0.53, 0.75, 0.80\}$. The individual conditional MSEs are

$$Q_{P|S<0.75} = 4 \times 10^{-4} \quad \text{and} \quad Q_{P|S\geq 0.75} = 0.0385 \tag{9.18}$$

and thus, the averaged conditional MSE with a split of S at 0.75 is

$$Q_{P|S(0.75)} = \frac{2}{7} Q_{P|S<0.75} + \frac{5}{7} Q_{P|S\geq 0.75} = 0.0276. \tag{9.19}$$

We need to sweep through all possible thresholds τ to determine the best threshold for attribute S. Since P is monotonically increasing and in the same order as S, we can just use the order as presented in the table above. Doing so, we get the following results:

$Q_{P\|S(0.55)}$	$Q_{P\|S(0.75)}$	$Q_{P\|S(1.5)}$	$Q_{P\|S(2.5)}$	$Q_{P\|S(3.1)}$	$Q_{P\|S(3.5)}$
0.0402	0.0276	0.0145	0.0102	0.0116	0.0325

To deal with the attribute Number of Rooms R, we need to rearrange the target variables in order of the house sizes. Doing so we get $(0.23, 0.19, 0.28, 0.53, 0.42, 0.75, 0.80)$. Now we sweep through all possible thresholds τ for R to get the following averaged conditional MSEs

$Q_{P\|R(1.5)}$	$Q_{P\|R(2.5)}$	$Q_{P\|R(3.5)}$	$Q_{P\|R(4.5)}$	$Q_{P\|R(5.5)}$	$Q_{P\|R(6.5)}$
0.0435	0.0276	0.0145	0.0222	0.0116	0.0325

The minima of the split of the S and R variables at different thresholds τ are shaded. We choose the minimum MSE as doing so and keeping in mind that Q_P is the same throughout, the gain which is $\text{Gain}(S(\tau); P) = Q_P - Q_{P|S(\tau)}$ or $\text{Gain}(R(\tau); P) = Q_P - Q_{P|R(\tau)}$ for various τ is maximized (just like for classification). We see that the minimum MSE is attained for the split of the S attribute at $\tau = 2.5$.

Thus, for the first split, we should split the dataset into two branches, the left branch indicating $S < 2.5$ and the right with $S \geq 2.5$. We then split the dataset into two sub-datasets and we may decide to stop or split

the R feature. If we decide to stop, then for any new/test house with a house size of < 2.5, we will predict that its price is the average of the houses in our training set whose size is < 2.5, i.e., $(0.19+0.23+0.28+0.42)/4 = 0.28$. Similarly, for a new/test house with a house size of ≥ 2.5, we will predict that its price is the average of the houses in our training set whose size is ≥ 2.5, i.e., $(0.53 + 0.75 + 0.80)/3 = 0.6933$.

9.3 Practice Problems

Exercise 9.1. *Using the Shannon entropy measure, learn the decision tree with the dataset given in Section 9.1 but with the* 15^{th} *row (data sample) removed.*

Exercise 9.2. *Convince yourself from Fig. 9.2 that in the binary classification case, using the entropy, Gini impurity measure and the misclassification rate result in the same learned decision tree. Can we say the same for multiclass classification?*

Exercise 9.3. *In Section 9.2, we first discretized each feature by selecting thresholds and then found the split, together with the threshold, that minimizes the average conditional MSE. A more direct way is to employ variance reduction, defined as*

$$V_m := \frac{1}{|\mathcal{J}_m|^2} \sum_{(i,j)\in\mathcal{J}_m^2} \frac{1}{2}(y_i - y_j)^2 - \left[\frac{|\mathcal{J}_{m,1}|^2}{|\mathcal{J}_m|^2} \cdot \frac{1}{|\mathcal{J}_{m,1}|^2} \sum_{(i,j)\in\mathcal{J}_{m,1}^2} \frac{1}{2}(y_i - y_j)^2 + \frac{|\mathcal{J}_{m,2}|^2}{|\mathcal{J}_m|^2} \cdot \frac{1}{|\mathcal{J}_{m,2}|^2} \sum_{(i,j)\in\mathcal{J}_{m,2}^2} \frac{1}{2}(y_i - y_j)^2 \right]. \tag{9.20}$$

Here $\mathcal{J}_m$ is the set of points at node m and $\mathcal{J}_{m,i}, i = 1, 2$ are the sets of points after node m has been split. Note that each of the sums is an estimate of the variance of the points in each set. Prove an analogue of Proposition 9.2, i.e., that $V_m \geq 0$ for any split of $\mathcal{J}_m$ into $\mathcal{J}_{m,1}$ and $\mathcal{J}_{m,2}$.

9.A Proof of Non-Increase of Impurity (Optional)

In this appendix, we show that the impurity "can only go down". We need a basic definition.

Definition 9.1. *A function $f : [a, b] \to \mathbb{R}$ is* convex *if $f(\lambda x+(1-\lambda)y) \leq \lambda f(x)+(1-\lambda)f(y)$ for all $\lambda \in [0, 1]$ and $x, y \in [a, b]$. A function f is* concave *if $-f$ is convex.*

We only consider two classes here. In this case, the impurity $h : [0, 1] \to \mathbb{R}$ is a function of a single parameter p which can be interpreted as the fraction of samples in class 1.

Proposition 9.1. *Suppose we have m_1 samples in class 1 and m_2 samples in class 2. Let $m = m_1+m_2$. For any split such that $m_1 = m_{11} + m_{12}$ and $m_2 = m_{21} + m_{22}$, for any concave impurity measure $h : [0, 1] \to \mathbb{R}$,*

$$\frac{m_{11} + m_{21}}{m} h\left(\frac{m_{11}}{m_{11} + m_{21}}\right) + \frac{m_{12} + m_{22}}{m} h\left(\frac{m_{12}}{m_{12} + m_{22}}\right) \leq h\left(\frac{m_1}{m}\right). \tag{9.21}$$

We remark that the LHS represents the average impurity after we have split the original m samples into two groups. The first group has $m_{11} + m_{21}$ samples with m_{11} of those samples from class 1 and m_{21} samples from class 2. The second group has $m_{21} + m_{22}$ samples with m_{21} of those samples from class 1 and m_{22} samples from class 2. See Fig. 9.3 for the notation used. Equation (9.21) basically says that the impurity can only go down; it cannot increase. Notice from Fig. 9.2 that all the impurity measures are concave so for the entropy, Gini, and misclassification impurities, these impurities can only decrease if we do any arbitrary split. Of course, we want to do the best split, but if you notice that after your split, the impurity increases, you must have calculated something wrongly!

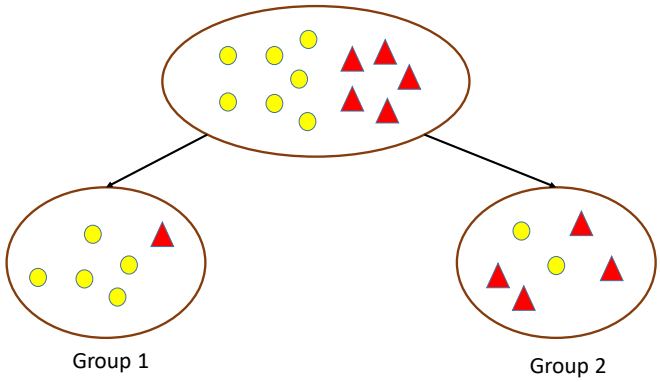


Figure 9.3: Here, $m_1 = 7$ (yellow circles) and $m_2 = 5$ (red triangles). They are split into two groups as shown. In this split, which may not be optimal, $m_{11} = 5$, $m_{12} = 1$, $m_{21} = 2$ and $m_{22} = 4$.

Proof of Proposition 9.1. Note that if we set $\lambda = (m_{11} + m_{21})/m$, then $1 - \lambda = (m_{12} + m_{22})/m$. In other words, the coefficients on the LHS of (9.21) are non-negative and add up to one. Since we assumed that $h : [0, 1] \to \mathbb{R}$ is concave, the LHS of (9.21) simplifies to

$$\begin{aligned} &\underbrace{\frac{m_{11} + m_{21}}{m}}_{=\lambda} h\left(\frac{m_{11}}{m_{11} + m_{21}}\right) + \underbrace{\frac{m_{12} + m_{22}}{m}}_{=1-\lambda} h\left(\frac{m_{12}}{m_{12} + m_{22}}\right) \\ &\overset{\text{Def. 9.1}}{\leq} h\left(\frac{m_{11} + m_{21}}{m} \cdot \frac{m_{11}}{m_{11} + m_{21}} + \frac{m_{12} + m_{22}}{m} \cdot \frac{m_{12}}{m_{12} + m_{22}}\right) \qquad (9.22) \\ &= h\left(\frac{m_{11}}{m} + \frac{m_{12}}{m}\right) = h\left(\frac{m_1}{m}\right) \qquad (9.23) \end{aligned}$$

as desired. □

A natural question: Can we do the same for the MSE for "regression using decision trees" as in Section 9.2?

Proposition 9.2. *Suppose we have m (unordered) target values $\{y_1, y_2, \ldots, y_m\} \subset \mathbb{R}$ in our original dataset and the MSE is Q_1. We partition (or split) the dataset into two mutually disjoint parts $\{y_1, y_2, \ldots, y_{m_1}\}$ and $\{y_{m_1+1}, y_{m_1+2}, \ldots, y_m\}$. The MSEs of each node at level 2 are Q_{21} and Q_{22} respectively. Then, it holds that*

$$\frac{m_1}{m} Q_{21} + \frac{m - m_1}{m} Q_{22} \leq Q_1. \qquad (9.24)$$

In other words, the MSE (just like the impurity measures) can only go down after a split.

Proof of Proposition 9.2. By the definition of MSE in (9.17), we have

$$Q_1 := \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{\mu}_1)^2, \quad Q_{21} := \frac{1}{m_1} \sum_{i=1}^{m_1} (y_i - \hat{\mu}_{21})^2, \quad \text{and} \quad Q_{22} := \frac{1}{m - m_1} \sum_{i=m_1+1}^{m} (y_i - \hat{\mu}_{22})^2, \qquad (9.25)$$

where the means are

$$\hat{\mu}_1 = \frac{1}{m} \sum_{i=1}^{m} y_i, \quad \hat{\mu}_{21} = \frac{1}{m_1} \sum_{i=1}^{m_1} y_i, \quad \text{and} \quad \hat{\mu}_{22} = \frac{1}{m - m_1} \sum_{i=m_1+1}^{m} y_i. \qquad (9.26)$$

Hence, the LHS of (9.24), can be written as

$$\frac{1}{m} \left[\sum_{i=1}^{m_1} (y_i - \hat{\mu}_{21})^2 + \sum_{i=m_1+1}^{m} (y_i - \hat{\mu}_{22})^2 \right]. \qquad (9.27)$$

Now, let us consider the function

$$f(a,b) := \sum_{i=1}^{m_1}(y_i - a)^2 + \sum_{i=m_1+1}^{m}(y_i - b)^2. \tag{9.28}$$

We would like to minimize it with respect to a and b. By differentiating f with respect to a and setting the partial derivative to zero, we get

$$\frac{\partial f(a,b)}{\partial a} = -2\sum_{i=1}^{m_1}(y_i - a) = 0 \quad \Longrightarrow \quad a = \frac{1}{m_1}\sum_{i=1}^{m_1} y_i. \tag{9.29}$$

and so the optimal a is $\hat{\mu}_{21}$, the average of the points in $\{y_1, \ldots, y_{m_1}\}$. Similarly, the optimal b is the average of the points in $\{y_{m_1+1}, \ldots, y_m\}$, namely $\hat{\mu}_{22}$. Thus,

$$\min_{a,b} f(a,b) = f(\hat{\mu}_{21}, \hat{\mu}_{22}). \tag{9.30}$$

In other words,

$$\begin{aligned}(9.27) = \frac{f\left(\hat{\mu}_{21}, \hat{\mu}_{22}\right)}{m} &\overset{(9.30)}{\leq} \frac{f(\hat{\mu}_1, \hat{\mu}_1)}{m} = \frac{1}{m}\left[\sum_{i=1}^{m_1}(y_i - \hat{\mu}_1)^2 + \sum_{i=m_1+1}^{m}(y_i - \hat{\mu}_1)^2\right] \\ &= \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{\mu}_1)^2 = Q_1, \end{aligned} \tag{9.31}$$

as was to be shown. □

This technique will be useful when we discuss the K-means algorithm in Chapter 11.

Chapter 10

Cross-Validation and Performance Metrics

In this chapter, we provide a description of cross-validation for hyperparameter tuning. I describe as carefully as I can the distinctions between the training, validation and test sets. We perform some basic statistical analyses for leave-one-out cross-validation. Finally, we discuss performance metrics for binary classification including the Receiver Operating Characteristic (ROC) and Area Under Curve (AUC).

10.1 Cross-Validation for Tuning Hyperparameters

As we have seen in machine learning, to learn a good model, we not only need to have a good cost function, regularizer and optimization algorithm, it is also imperative to choose the hyperparameters appropriately. What exactly are *hyperparameters*? They are, for example, the λ parameter when we do ridge-regression with linear features. For a fixed $\lambda \geq 0$, this could be the degree p of the polynomial used to enhance the feature space in a linear model. A hyperparameter could also be the number of trees we include in a random forest, the number of levels of a decision tree, or the minimum number of samples at a leaf/terminal node. How can we use our dataset to learn these hyperparameters to find the best model (out of a given finite set of them)?

For the sake of simplicity, say we only have a single hyperparamter λ in the context of ridge regression with a fixed polynomial order p. We do not know what is the best λ we should choose to minimize the test error. What can we do, at least on a heuristic (no proof) level to learn λ? Cross-validation is one way to do it. Let us refer to Fig. 10.1.

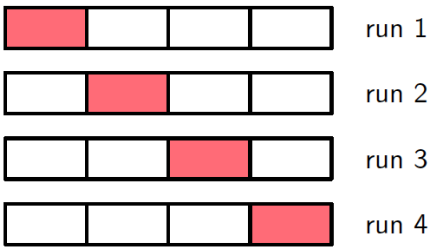


Figure 10.1: Training and validation sets

What we would do is to split the available training dataset into two parts—the training set and the validation set (indicated in red above). Consider just the first row. Let us call the training set and validation set $\mathcal{T}$ and $\mathcal{V}$ respectively. Note that these sets are mutually disjoint and $\mathcal{D} = \mathcal{T} \cup \mathcal{V}$.

We would presume that λ takes on values in some finite set Λ. In practice, what we would do is to consider the set Λ to contain pretty "spread out" values of λ, say $\Lambda = \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$. Here, I chose the values of $\lambda \in \Lambda$ to be on a logarithmic scale. In this example, the size of Λ is 7. For this split of $\mathcal{D}$ into $\mathcal{T} \cup \mathcal{V}$, we would train the model using each value of $\lambda \in \Lambda$ on $\mathcal{T}$, the training set. We would compute the mean-squared errors (or some form of loss) on the validation set $\mathcal{V}$ for each $\lambda \in \Lambda$. This we can do because we have the labels or target values of each training example $\mathbf{x}_i$ belonging to $\mathcal{V}$. Say the mean-squared errors are $\text{MSE}^{(1)}(\lambda), \lambda \in \Lambda$ where the superscript (1) represents the current 1st run.

There is no reason to bias the whole procedure to the validation set being at the start of the dataset as in the first row of Fig. 10.1. Thus, we consider several other runs—also called folds—in order to place the validation set in various parts of the dataset. We then can obtain $\text{MSE}^{(k)}(\lambda)$ where $\lambda \in \Lambda$ and $k = 1, \ldots, K$ where K is the number of folds. In Fig. 10.1, there are $K = 4$ folds. The best λ to choose is then

$$\hat{\lambda} = \arg\min_{\lambda \in \Lambda} \frac{1}{K} \sum_{k=1}^{K} \text{MSE}^{(k)}(\lambda). \tag{10.1}$$

The problem with this procedure is that it is often computationally intensive because we have to train the models K times; in each time for a different value of hyperparameter. Worse, we often have multiple hyperparameters in our system (e.g., maximum depth of trees, number of trees, minimum number of samples in each leaf within a random forest). This creates a multi-dimensional grid of hyperparameters, which results in an uncontrollable growth of number of models we have to train. Nevertheless, there are some smart ways to implement cross-validation. Please see this nice article `https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74`.

10.2 On Training, Validation, and Test Sets

The validation sets are not to be confused with the test set in which we do not have the labels or targets. The test set is for us to apply our model with the "best" hyperparameters we got through training the model using validation sets (e.g., $\hat{\lambda}$ in (10.1)) to get a good estimate of the test error. To be clear, the definitions of the training, validation and test sets are as follows.

- Training Dataset: The sample of data used to *fit a certain model*.
- Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset *while tuning the model hyperparameters*.
- Test Dataset: The sample of data used to provide an unbiased evaluation of the *final* model fit on the training dataset.

Please refer to the pseudocode below just for the workflow based on one split. In practice, we need to do $K > 1$ splits and find the best model based on the average performance over the K splits.

```
1. data = load_data()
2. # split data (only one split)
3. train, validation, test = split(data)
4.
5. # tune model hyperparameters
6. set_of_parameters = ... # a pre-determined finite set
7. for params in set_of_parameters:
8.         model = fit(train, params)
9.         performance = evaluate(model, validation)
10.
11. # find best model hyperparameters
12. best_params = argmax(performance)
```

13.
14. # evaluate final "best" model for comparison to other competitors
15. best_model = fit(train, best_params)
16. best_performance = evaluate(best_model, test)

In line 12 above, the performance is computed to be the average over the performances of splits if we have multiple splits; see Eqn. (10.1).

10.3 Properties of Leave-One-Out Cross-Validation (Optional)

As we mentioned in the previous section, cross-validation is a generic tool that can be used to select appropriate features, values for regularization parameters such as λ in ridge regression or the number of trees in a random forest model. It can even be used to choose among competing models. We will consider here some basic properties of *leave-one-out cross-validation* (LOOCV) in which we train m models (where m is the number of training samples) with one sample left out of the original dataset. This is also known as m-fold cross-validation.

More formally, let $\mathcal{D}_m = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ denote our training set with m training samples and $\mathcal{D}_m^{-i}$ be the corresponding set with the i-th training example and label removed. That is, $\mathcal{D}_m^{-i} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{i-1}, y_{i-1}), (\mathbf{x}_{i+1}, y_{i+1}), \ldots, (\mathbf{x}_m, y_m)\}$. LOOCV is performed as follows: For each $(\mathbf{x}_i, y_i)$ in the training set, we train the classifier on the remaining $m-1$ points $\mathcal{D}_m^{-i}$ and test our prediction on the left-out pair $(\mathbf{x}_i, y_i)$. More precisely, when using the squared loss, we define $\mathrm{MSE}_{\mathrm{LOOCV}}$ as

$$\mathrm{MSE}_{\mathrm{LOOCV}}(\mathcal{D}_m) = \frac{1}{m} \sum_{i=1}^{m} \left(y_i - \hat{f}_{-i}(\mathbf{x}_i) \right)^2 \tag{10.2}$$

where $\hat{f}_{-i}(\cdot)$ is the estimator trained on $\mathcal{D}_m^{-i}$. We would like to understand the theoretical properties of $\mathrm{MSE}_{\mathrm{LOOCV}}(\mathcal{D}_m)$

Let's start with a simpler strategy. We only leave out the first point, i.e., training with $\mathcal{D}_m^{-1}$, and test on $(\mathbf{x}_1, y_1)$. The mean-squared error is now

$$\mathrm{MSE}_1(\mathcal{D}_m) = \left(y_1 - \hat{f}_{-1}(\mathbf{x}_1) \right)^2. \tag{10.3}$$

Assuming each training example and label is sampled independently from some underlying distribution $P(\mathbf{x}, y)$, let us convince ourselves that

$$\mathbb{E}[\mathrm{MSE}_1(\mathcal{D}_m)] = \mathbb{E}\left[\left(y - \hat{f}_{\mathcal{D}_{m-1}}(\mathbf{x}) \right)^2 \right] \tag{10.4}$$

where the expectation on the left is over all random quantities and, on the right hand side, it is over both $(\mathbf{x}, y)$ (test example) as well as a dataset $\mathcal{D}_{m-1}$ of size $m-1$ sampled from the same distribution. In other words, on average, $\mathrm{MSE}_1(\mathcal{D}_m)$ gives the best test error! To show (10.4), let us note that if A and B are random variables (RVs) with the same probability distribution or density (say $f_A = f_B$), then $\mathbb{E}[g(A)] = \mathbb{E}[g(B)]$ for any function g. This may be clearer when we write out the corresponding integral:

$$\mathbb{E}[g(A)] = \int g(x) f_A(x)\, \mathrm{d}x = \int g(x) f_B(x)\, \mathrm{d}x = \mathbb{E}[g(B)]. \tag{10.5}$$

This holds also when A and B are sets of RVs. In particular, $A = \{\mathcal{D}_m^{-1}, (\mathbf{x}_1, y_1)\}$, where we would train on $\mathcal{D}_m^{-1}$ and test on $(\mathbf{x}_1, y_1)$, has the same distribution as $B = \{\mathcal{D}_{m-1}, (\mathbf{x}, y)\}$, where we would train on another set of $m-1$ samples $\mathcal{D}_{m-1}$ and test on $(\mathbf{x}, y)$, also sampled from the same distribution. Hence, (10.4) holds.

Now, using the above result, we would like to establish that $\mathrm{MSE}_{\mathrm{LOOCV}}(\mathcal{D}_m)$ also has this property, i.e.,

$$\mathbb{E}[\mathrm{MSE}_{\mathrm{LOOCV}}(\mathcal{D}_m)] = \mathbb{E}\left[\left(y - \hat{f}_{\mathcal{D}_{m-1}}(\mathbf{x}) \right)^2 \right]. \tag{10.6}$$

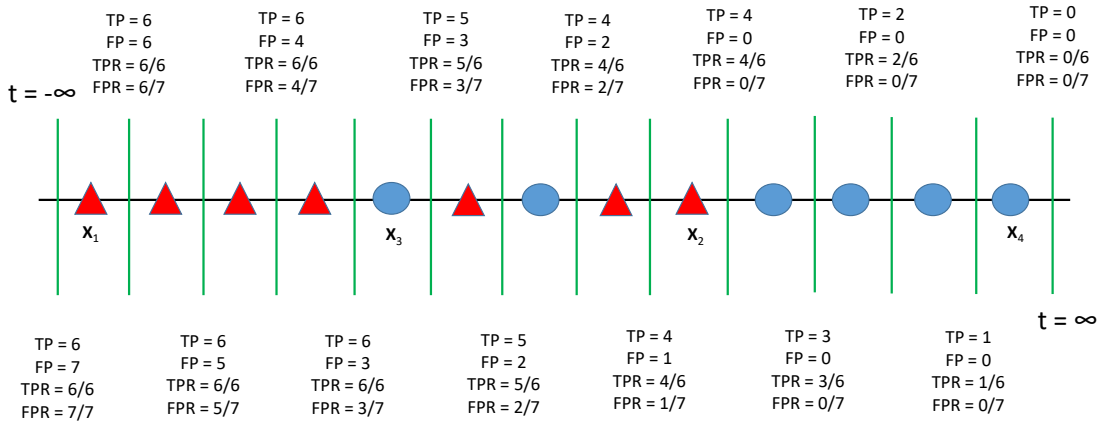


Figure 10.2: The positively and negatively labelled samples arranged in increasing order based on the score function g. The thresholds are indicated in green lines. The TPRs and FPRs are also indicated.

By the linearity of expectation, and Eqn. (10.4), we have

$$\mathbb{E}\left[\mathrm{MSE}_{\mathrm{LOOCV}}(\mathcal{D}_m)\right] = \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}\left[(y_i - \hat{f}_{-i}(\mathbf{x}_i))^2\right] = \frac{1}{m}\sum_{i=1}^{m}\mathrm{MSE}_i(\mathcal{D}_m) = \mathrm{MSE}_1(\mathcal{D}_m). \tag{10.7}$$

The results in (10.4) and (10.6) seem to indicate that both LOOCV and the single test set approximation are unbiased estimates of the test error based on the $m-1$ training examples. Are the variances of $\mathrm{MSE}_{\mathrm{LOOCV}}$ and MSE_1 the same as well? Why can we not naïvely assume that LOOCV is an average of m tests so that its variance would do down asymptotically as $1/m$? Well, alas the variances will not be equal! $\mathrm{MSE}_{\mathrm{LOOCV}}$ is an estimate based on averaging the error over m trials while MSE_1 is based on a single trial. Recall that if r random variables $V_1, V_2, \ldots, V_r$ are distributed i.i.d., the variance of the sample mean $\bar{V} = \frac{1}{r}\sum_{j=1}^{r} V_j$ is $1/r$ times the variance of the V_i's. While the trials in $\mathrm{MSE}_{\mathrm{LOOCV}}$ are far from independent because there is a significant amount of "overlap" in the training sets, $\mathrm{MSE}_{\mathrm{LOOCV}}$ will nevertheless have lower variance (compared to MSE_1).

10.4 Receiver Operating Characteristic (ROC) and Area Under Curve (AUC)

Now, we are in a position to evaluate the goodness of our classifiers. In a binary classification problem, we have two sets of samples—the positively labelled samples $\mathcal{P} = \{\mathbf{x}_i : y_i = +1\}$ and the negatively labelled samples $\mathcal{N} = \{\mathbf{x}_i : y_i = -1\}$. We can design a *score function* $g : \mathbb{R}^d \to \mathbb{R}$ that evaluates how positive each training or test sample is. For example, if we have learned a linear classifier with no offset, we have in fact learned a set of weights $\mathbf{w} \in \mathbb{R}^d$. Thus, the score function takes the form $g(\mathbf{x}) = \mathbf{x}^\top\mathbf{w}$ and we typically declare that $\mathbf{x}$ is positively labelled if $g(\mathbf{x}) \geq 0$ and negatively labelled otherwise. In other words, the predicted class of $\mathbf{x}$ is $y = \mathrm{sgn}(g(\mathbf{x})) = \mathrm{sgn}(\mathbf{x}^\top\mathbf{w}) \in \{\pm 1\}$. We are implicitly placing the threshold for deciding whether or not a class is positively labelled at 0. However, the number 0 is not sacrosanct and there may be better thresholds $t \neq 0$ that suit the task at hand better.

Let us say that we have fixed threshold $t \in \mathbb{R}$ and a score function g. That is, the predicted label is

$$y = \mathrm{sgn}(g(\mathbf{x}) - t) \in \{\pm 1\}. \tag{10.8}$$

Then we can compute the number of True Positives TP, False Positives FP, True Negatives TN, False Negatives FN. The true positives are the positively labelled samples that have been (correctly) classified

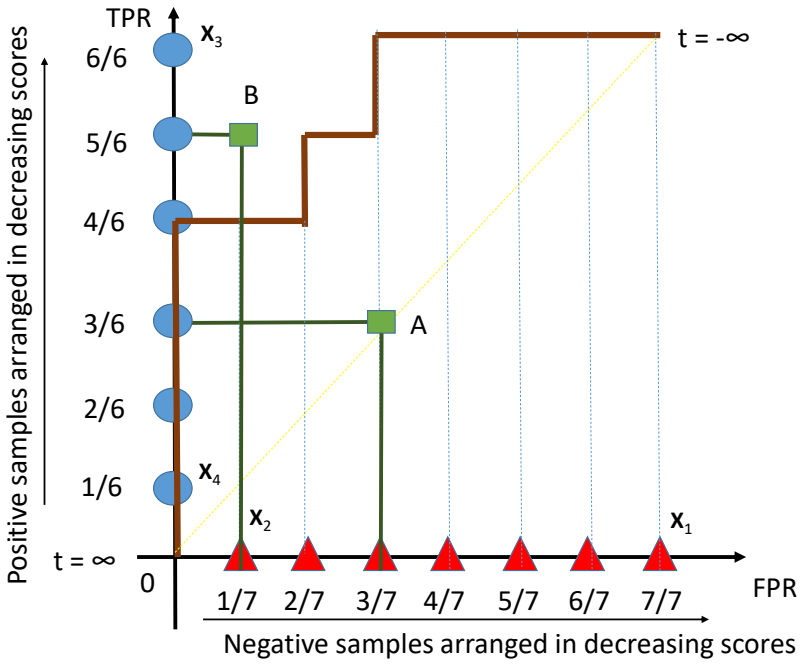


Figure 10.3: The Receiver Operating Characteristic (ROC) corresponding to the dataset and score function in Fig. 10.2

as positives; the false positives are the negatively labelled samples that have been (incorrectly) classified as positives. For example, if we using IgE levels to determine a child's propensity to a certain allergy, a true positive is the result of a test that declares a child has the allergy when he in fact has the allergy; a false positive is a test that declares that a child has the allergy when in fact he does not have it. As we vary the threshold t from $-\infty$ to ∞, the number of true positives decreases and the number of false positives increases. As an example, look at Fig. 10.2. We have also tabulated the True Positive Rates and False Positive Rates defined together with the False Negative Rates and False Positive Rates as follows:

$$\text{TPR} = \frac{\text{TP}}{\text{TP}+\text{FN}}, \quad \text{FNR} = \frac{\text{FN}}{\text{TP}+\text{FN}}, \quad \text{TNR} = \frac{\text{TN}}{\text{FP}+\text{TN}}, \quad \text{FPR} = \frac{\text{FP}}{\text{FP}+\text{TN}}. \tag{10.9}$$

The thresholds t are the green bars in Fig. 10.2. Observe that when $t = -\infty$, TPR = FPR = 1 because all points are classified as being positively labelled. When $t = +\infty$, TPR = FPR = 0 because all points are classified as being negatively labelled. The TPR and FPR do not increase (and sometimes decrease) as t increases. Verify that all the numbers in Fig. 10.2 are correct.

We can now place the points in Fig. 10.2 on the axes of Fig. 10.3, which is constructed as follows. On the vertical axis, we place the positively labelled points in decreasing order. So the *topmost* blue point on the vertical axis $\mathbf{x}_3$ corresponds to the positively labelled sample with the *least* value of g. Similarly, on the horizontal axis, we place the negatively labelled points in decreasing order. So the *rightmost* red point on the horizontal axis $\mathbf{x}_1$ corresponds to the negatively labelled sample with the *least* value of g. By varying the threshold t, we can sketch out a curve, indicated in brown. Check that the piecewise linear brown curve is indeed correctly drawn based in Fig. 10.2. This is the so-called *Receiver Operating Characteristic* or ROC. This term originated from World War II in which the British were designing radar systems to detect if the German Luftwaffe (air force) was in the air. The ROC used as follows. Suppose one can tolerate a FPR to be some small number say 1/7. What is the best TPR that can be attained? From the curve, you see that the best TPR is 4/6.

To compare between different score functions g, one typically compares the Area Under Curve (AUC), which is a function of the dataset $\mathcal{D}$ (or $\mathcal{P}$ and $\mathcal{N}$) and the score function g.

Proposition 10.1. *The AUC is given by the formula*

$$\text{AUC} = \frac{1}{|\mathcal{P}||\mathcal{N}|} \sum_{\mathbf{x}^+ \in \mathcal{P}, \mathbf{x}^- \in \mathcal{N}} u\big(g(\mathbf{x}^+) - g(\mathbf{x}^-)\big), \tag{10.10}$$

where

$$u(e) = \begin{cases} 1 & e > 0 \\ 1/2 & e = 0 \\ 0 & e < 0 \end{cases}. \tag{10.11}$$

Why is the formula in (10.10) the way it is? Let us prove that the AUC is indeed the formula given in (10.10) when the $g(\mathbf{x})$'s are distinct for all training samples $\mathbf{x}$ in the dataset. In Exercise 10.2, you will provide the complete proof of Proposition 10.1 for the (rare) case in which we have points $\mathbf{x}^+$ and $\mathbf{x}^-$ such that $g(\mathbf{x}^+) = g(\mathbf{x}^-)$.

Remember the area under the curve is the integral of the curve over its domain. Let us see that (10.10) indeed gives the area under the brown curve in Fig. 10.3. Randomly and uniformly take a point on the vertical axis and a point on the horizontal axis. Call them $\mathbf{X}^+$ and $\mathbf{X}^-$ respectively.[1] Note that the probability that $\mathbf{X}^+$ (resp. $\mathbf{X}^-$) equals any of the positively (resp. negatively) labelled points is 1/6 (resp. 1/7). What is the chance that $(\mathbf{X}^-, \mathbf{X}^+)$ lies under the curve? Because the points on the horizontal and vertical axes are arranged in decreasing orders, $(\mathbf{X}^-, \mathbf{X}^+)$ lies under ROC if and only if the score of the blue point is greater than the score of the red point, i.e., $g(\mathbf{X}^+) > g(\mathbf{X}^-)$. This is indicated by point A (under the ROC) on Fig. 10.3. Also refer to Fig. 10.2. On the other hand, $(\mathbf{X}^-, \mathbf{X}^+)$ is above the ROC if the scores of $(\mathbf{X}^-, \mathbf{X}^+)$ satisfy $g(\mathbf{X}^+) < g(\mathbf{X}^-)$. This is indicated by point B (above the ROC) on Fig. 10.3. This says that

$$\text{AUC} = \Pr\big(g(\mathbf{X}^+) > g(\mathbf{X}^-)\big), \tag{10.12}$$

where $\mathbf{X}^+$ and $\mathbf{X}^-$ are respectively uniform over the negatively and positively labelled samples respectively. Note that[2]

$$\Pr\big(g(\mathbf{X}^+) > g(\mathbf{X}^-)\big) = \sum_{\mathbf{x}^+ \in \mathcal{P}, \mathbf{x}^- \in \mathcal{N}} \Pr(\mathbf{X}^+ = \mathbf{x}^+, \mathbf{X}^- = \mathbf{x}^-) \mathbb{1}\big\{g(\mathbf{x}^+) > g(\mathbf{x}^-)\big\} \tag{10.13}$$

$$= \frac{1}{|\mathcal{P}||\mathcal{N}|} \sum_{\mathbf{x}^+ \in \mathcal{P}, \mathbf{x}^- \in \mathcal{N}} \mathbb{1}\big\{g(\mathbf{x}^+) > g(\mathbf{x}^-)\big\}, \tag{10.14}$$

which is exactly equal to the formulae in (10.10) and (10.11) since we assumed that there are no positively labelled points that have the same g value as negatively labelled points (so the second clause of u is never activated). The formula (10.10) is proved.

10.5 Practice Problems

Exercise 10.1. *Suppose we would like to use K-fold cross-validation to learn the best polynomial order in the set $\mathcal{P} = \{1, \ldots, p\}$ and the best regularization parameter in the set Λ where $|\Lambda| = \ell$. How many models have to be trained?*

Exercise 10.2. *Verify the TP, FP, TPR, and FPR values in Fig. 10.2. Thus verify that the ROC is indeed drawn correctly in Fig. 10.3.*

Exercise 10.3. *Generalize the proof of Proposition 10.1 to the case in which there are points $\mathbf{x}^+$ and $\mathbf{x}^-$ such that $g(\mathbf{x}^+) = g(\mathbf{x}^-)$ so $u(\,\cdot\,)$ does take on the value $1/2$. Some conditions or assumptions may be needed here.*

[1] We use the upper case letters $\mathbf{X}^+$ and $\mathbf{X}^-$ because these quantities are random variables.

[2] We use the notation $\mathbb{1}\{\text{clause}\}$ to denote the indicator function which returns 1 if the clause is true and 0 otherwise.

Chapter 11

K-Means Clustering

In this chapter, we provide a description of the ubiquitous K-means clustering algorithm and prove an important property of the algorithm. We provide an example to show that initialization affects the solution. Next, we show how to initialize in a smart way that has some theoretical guarantees. Finally, we provide a few practice problems for you to familiarize yourself with K-means.

A word about notation. As usual, we have m data samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ that are unlabelled since this is an unsupervised learning task. The dataset is thus $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^m \subset \mathbb{R}^d$. Often this will be abbreviated as $\{\mathbf{x}_i\}$ when the underlying number of samples m is suppressed from the notation (since there is usually no cause of confusion). We do this for other sets as well. The total number of clusters is K. The cluster that the i-th sample $\mathbf{x}_i$ is assigned to at a certain iteration is denoted as k_i, an integer in $\{1, \ldots, K\}$. The cardinality or size of a finite set $\mathcal{A}$ is denoted as $|\mathcal{A}|$. Iteration count is indicated by a superscript (l). As usual, the l_2 norm of a vector $\mathbf{a}$ is defined as $\|\mathbf{a}\| = \sqrt{\sum_i a_i^2}$.

11.1 The K-means Clustering Algorithm

Given a set of unlabelled data points $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^m$, and a fixed number of clusters $2 \leq K \leq m$, we execute the following procedure.

1. Step 0: Initialize the centroids $\mathbf{c}_k^{(1)}$ for each $1 \leq k \leq K$ randomly (and far apart).
2. Step $l \in \mathbb{N}$ (Assignment step) : Assign each point $\mathbf{x}_i$ to its closest mean, i.e.,
$$k_i^{(l)} = \underset{k=1,\ldots,K}{\arg\min} \left\| \mathbf{x}_i - \mathbf{c}_k^{(l)} \right\|^2 \tag{11.1}$$
3. Step $l \in \mathbb{N}$ (Re-computation step) : Recompute centroids (also called means) by averaging the points within each cluster
$$\mathbf{c}_k^{(l+1)} = \frac{1}{|\{i : k_i^{(l)} = k\}|} \sum_{i:k_i^{(l)}=k} \mathbf{x}_i. \tag{11.2}$$
We note that $|\{i : k_i^{(l)} = k\}|$ denotes the number of points in cluster k.
4. Terminate when cluster assignments $k_i^{(l)}$ for all $i \in \{1, 2, \ldots, m\}$ no longer change as we increment the iteration number from l to $l + 1$.

We can write the cost (objective) function for K-means clustering as

$$J(\{w_{ik}\}, \{\mathbf{c}_k\}) = \sum_{i=1}^{m} \sum_{k=1}^{K} w_{ik} \|\mathbf{x}_i - \mathbf{c}_k\|^2 \tag{11.3}$$

where $w_{ik} = 1$ if $\mathbf{x}_i$ belongs to cluster k (i.e., $k = \arg\min_j \|\mathbf{x}_i - \mathbf{c}_j\|$); otherwise $w_{ik} = 0$. The arguments of this cost function are the binary matrix $\{w_{ik} : 1 \le i \le m, 1 \le k \le K\}$ and the centroids $\{\mathbf{c}_k : 1 \le k \le K\}$. Let us see that optimizing this function over $\{w_{ik}\}$ and $\{\mathbf{c}_k\}$ yields the updates in (11.1) and (11.2).

Now suppose the centroids $\{\mathbf{c}_k\}$ are fixed. We would like to update $\{w_{ik}\} \in \{0,1\}^{m \times K}$. To do so, notice that for fixed $i \in \{1, \dots, m\}$, the inner sum is $\sum_{k=1}^{K} w_{ik}\|\mathbf{x}_i - \mathbf{c}_k\|^2$. The w_{ik} can only be 1 if for one particular $k \in \{1, \dots, K\}$; the rest of the entries are equal to 0. Clearly, to minimize $\sum_{k=1}^{K} w_{ik}\|\mathbf{x}_i - \mathbf{c}_k\|^2$ (over $\{w_{ik}\}$), we should assign $w_{ik} = 1$ when $\mathbf{x}_i$ is closest to $\mathbf{c}_k$. This is precisely the rule in (11.1). Now for fixed $\{w_{ik}\}$, we want to find the best centroids. We can take the derivative of $J(\{w_{ik}\}, \{\mathbf{c}_k\})$ with respect to one particular $\mathbf{c}_j$ (for $1 \le j \le K$) as follows:

$$\nabla_{\mathbf{c}_j} J(\{w_{ik}\}, \{\mathbf{c}_k\}) = -2 \sum_{i=1}^{m} w_{ij}(\mathbf{x}_i - \mathbf{c}_j) = \mathbf{0}. \tag{11.4}$$

This implies that the optimal j-th centroid is the average of the points in the j-th cluster, i.e.,

$$\mathbf{c}_j = \frac{\sum_{i=1}^{m} w_{ij}\mathbf{x}_i}{\sum_{i=1}^{m} w_{ij}} = \frac{\sum_{i:k_i=j} \mathbf{x}_i}{|\{i : k_i = j\}|} \tag{11.5}$$

which serendipitously coincides with (11.1). Notice that $\sum_{i=1}^{m} w_{ij} = |\{i : k_i = j\}|$ since both sides are different ways of expressing the number of points that reside in cluster j. Furthermore, $w_{ij} = 1$ if and only of $k_i = j$, explaining the numerator of the rightmost term in (11.5).

11.2 Non-Increase of the Cost Function of K-means

The purpose of the next result is to show that the cost function in K-means in (11.3) cannot increase. It can only decrease until it "gets stuck" at a local minimum; or if you are lucky, a global minimum. Also *a priori*, since the cluster assignments and the centroids continually change, it is not clear whether the K-means algorithm will terminate after a finite number of steps, or like gradient descent, will only *converge* to a local minimum. The following proposition shows that we will definitely stop at some point and cannot make any improvement, no matter how small.

Proposition 11.1. *Let $J^{(l)} := J(\{w_{ik}^{(l)}\}, \{\mathbf{c}_k^{(l)}\})$ be the cost function at the l-th iteration based on the cluster assignments $w_{ik}^{(l)}$ and the centroids $\mathbf{c}_k^{(l)}$. Then, for any unlabelled dataset, the K-means algorithm results in*

$$J^{(l+1)} \le J^{(l)} \quad \textit{for all } l \in \mathbb{N}. \tag{11.6}$$

Furthermore, the K-means algorithm terminates in a finite number of steps.

Proof of Proposition 11.1. Once all samples have been assigned to their initial cluster in (11.1), compute the total distance between each sample $\mathbf{x}_i$ and its nearest centroid $\mathbf{c}_{k_i}$. The sum of squares of these distances is $J(\{w_{ik}^{(1)}\}, \{\mathbf{c}_k^{(1)}\}) = \sum_i \|\mathbf{x}_i - \mathbf{c}_{k_i}\|^2$ (compare this to (11.3)). When the centroids are recalculated per the Re-computation step in (11.2), we are minimizing the total distance of each cluster's samples from the respective centroid, so the new distance $J(\{w_{ik}^{(1)}\}, \{\mathbf{c}_k^{(2)}\})$ either decreases or remains the same. Note that the Re-computation step is equivalent to optimizing the functions

$$f_k(\mathbf{c}) = \sum_{i:k_i=k} \|\mathbf{x}_i - \mathbf{c}\|^2. \tag{11.7}$$

The optimum (minimum) of the above function is, of course, attained at

$$\hat{\mathbf{c}}_k = \frac{1}{|\{i : k_i = k\}|} \sum_{i:k_i=k} \mathbf{x}_i, \tag{11.8}$$

which is another way of stating (11.2). Consequently,

$$J(\{w_{ik}^{(1)}\}, \{\mathbf{c}_k^{(2)}\}) \leq J(\{w_{ik}^{(1)}\}, \{\mathbf{c}_k^{(1)}\}) = J^{(1)}. \tag{11.9}$$

In the subsequent assignment step, if a sample is reassigned to a different centroid then it must be that this sample is closer to its new centroid than to the previously one. If a sample is not reassigned, then there is obviously no change in its distance (cost). Therefore, for this new distance $J(\{w_{ik}^{(2)}\}, \{\mathbf{c}_k^{(2)}\})$, we have

$$J^{(2)} = J(\{w_{ik}^{(2)}\}, \{\mathbf{c}_k^{(2)}\}) \leq J(\{w_{ik}^{(1)}\}, \{\mathbf{c}_k^{(2)}\}). \tag{11.10}$$

Combining the above two relations, we see that

$$J^{(2)} \leq J^{(1)}. \tag{11.11}$$

By the same logic, for each subsequent recalculation or assignment step, we have (11.6). So we have shown that the cost function is monotonically non-increasing. Note that if equality holds in either (11.9) or (11.10), then there is either no change to any centroid (equality in (11.9)), or no sample is reassigned to a different cluster (equality in (11.10)).

Now we show that the procedure terminates in a finite number of steps. Since there are a finite number of samples m and a finite number of labellings, changes to cost function are discrete with a finite set of possibilities. Indeed, there are only K^m possible clusterings. Why? Note that for each clustering determined by $\{w_{ik}\}$, we can compute the optimal centroids $\hat{\mathbf{c}}_k$'s in (11.8). This leads to the cost $J(\{w_{ik}\}, \{\hat{\mathbf{c}}_k\})$. We note that $J(\{w_{ik}\}, \{\hat{\mathbf{c}}_k\})$ can only take on finitely many values; more precisely at most K^m values. Because of this, and because the cost function is lower bounded by 0 then it must be that the algorithm converges to some $J^{(l^*)}$ with $J^{(l^*)} = J^{(l^*-1)}$ and $0 \leq J^* \leq J^{(l^*)}$ where J^* is the optimal cost function over all partitions, which is hard (more precisely, NP-hard) in the computational complexity sense to compute. □

The message I want to get across here is that we need check that when we do K-means, the cost function *never increases*. If it does, you have done something wrong, because Proposition 11.1 says that $J^{(l+1)} \leq J^{(l)}$ for all $l \in \mathbb{N}$. Furthermore, if it seems like you are not converging (for a finite dataset), you must surely be doing something wrong as well because Proposition 11.1 says that we must stop; no convergence criterion is needed (unlike gradient descent).

11.3 An Example of K-means

Consider the following unlabelled one-dimensional dataset (so that the samples are all scalar)

$$x_1 = -2, \quad x_2 = 0, \quad x_3 = x_4 = 2. \tag{11.12}$$

Consider the first initialization

$$c_1^{(1)} = -3, \quad c_2^{(1)} = 3.5 \tag{11.13}$$

Then, once we run the Assignment step, we see that

$$k_1 = k_2 = 1, \quad k_3 = k_4 = 2. \tag{11.14}$$

This means that samples 1 and 2 are in group one and samples 3 and 4 are in group two. Thus,

$$c_1^{(2)} = -1, \quad c_2^{(2)} = 2. \tag{11.15}$$

The total cost function is

$$J = 1^2 + 1^2 + 0^2 + 0^2 = 2, \tag{11.16}$$

which turns out to be the optimum partitioning.

Now, instead consider the second initialization

$$c_1^{(1)} = -3, \quad c_2^{(1)} = 2.5 \tag{11.17}$$

Then, once we run the Assignment step, we see that

$$k_1 = 1, \quad k_2 = k_3 = k_4 = 2. \tag{11.18}$$

Thus,

$$c_1^{(2)} = -2, \quad c_2^{(2)} = 4/3 \tag{11.19}$$

The total cost function is

$$J' = 0^2 + (4/3)^2 + 2(2 - 4/3)^2 = 24/9 \tag{11.20}$$

which is suboptimal and there is no way of improving the cost anymore, i.e., we are stuck. The moral of the story is that initialization is important.

11.4 The K-means++ Algorithm (Optional)

The vanilla K-means algorithm has no theoretical guarantee (apart from Proposition 11.1) and as we have seen from Section 11.3, initialization is very important, otherwise one can get an arbitrarily bad solution. In an amazing breakthrough, Arthur and Vassilvitskii (2007) [AV07] suggested the following slight modification to K-means and proved an amazing guarantee. To state the algorithm called K-means++, let $D(\mathbf{x})$ be the distance from a data point $\mathbf{x} \in \mathcal{D}$ to to the closest centroid we have already chosen, i.e., $D(\mathbf{x}) = \min_{k \in \mathcal{K}} \|\mathbf{x} - \mathbf{c}_k\|$ where $\mathcal{K}$ denotes the set of centroids already chosen.

1a. Take one centroid $\mathbf{c}_1$ chosen uniformly at random from the dataset $\mathcal{D}$.

1b. Take one new centroid $\mathbf{c}_k$ choosing $\mathbf{x} \in \mathcal{D}$ with probability $D(\mathbf{x})^2 / \sum_{\mathbf{x}' \in \mathcal{D}} D(\mathbf{x}')^2$.

1c. Repeat Step 1b. until we have taken K centroids altogether.

2-4. Proceed as in the standard K-means algorithm above using the random chosen centroids in Step 1.

The intuition here is that we would like to select initial centroids that are as far as possible; this is precisely the rule (with randomization) in Step 1b. The following result holds if we run K-means++ on any dataset. Note that the output of K-means++ is random and so its cost function is random.

Theorem 11.2. *If the set of centroids is constructed with K-means++, then the corresponding cost function after convergence satisfies*

$$\mathbb{E}[J] \leq 8(\ln K + 2) \cdot J^*, \tag{11.21}$$

where J^* is the optimal cost function (which is NP-hard to compute).

In other words, K-means++ is $O(\log K)$-competitive with respect to the optimal clustering resulting in J^* which is NP-hard to find. The punchline is that if you do not know how to initialize, use Arthur and Vassilvitskii's initialization strategy [AV07]! It will give you something decent. A former student of mine, Davin Choo, who took my NUS Math Machine Learning class MA4270 in 2015/6 recently improved on this seminal result [CGPR20]. Together with his co-authors, he showed that with εK ($\varepsilon > 0$ arbitrarily small) local search steps, one obtains an $O(1)$-approximation to J^*. Amazing!

11.5 Practice Problems

The problems below are mostly taken from my MA4270 exams but they have been carefully recalibrated to be accessible to you. No computers or programming is required.

Exercise 11.1 (Final Exam 2017/8)**.** *We are given the following $m = 4$ two-dimensional data samples:*

$$\mathbf{x}_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \qquad \mathbf{x}_2 = \begin{bmatrix} 6 \\ 8 \end{bmatrix}, \qquad \mathbf{x}_3 = \begin{bmatrix} 8 \\ 6 \end{bmatrix}, \qquad \mathbf{x}_4 = \begin{bmatrix} 2 \\ 4 \end{bmatrix}.$$

(i) Using the K-means algorithm, cluster this dataset into two *clusters. To initialize the algorithm, put samples 1 and 2 in one cluster, and samples 3 and 4 in the other cluster. Show the steps of the algorithm clearly, i.e., calculate the cluster centroids and the assignments of the points to the cluster centroids. Give the value of the K-means (sum-of-squares) error function after convergence.*

(ii) What is the value of the error function in the optimal solution for $K = 4$? No calculations needed.

Exercise 11.2 (Final Exam 2015/6)**.** *Consider a set of $m = 2n + 1$ of one-dimensional samples where $n \in \mathbb{N}$. The dataset is such that n samples coincide at $x = -2$, n at $x = 0$, and one at $x = a > 0$, i.e.,*

$$\mathcal{D} = \left\{ \underbrace{-2, -2, \ldots, -2}_{n \text{ times}}, \ \underbrace{0, 0, \ldots, 0}_{n \text{ times}}, \ a \right\}.$$

(In fact $\mathcal{D}$ is a so-called multi-set *since it allows for multiple instances of the same element.)*

(i) Let $\mathcal{D}_1, \mathcal{D}_2 \subset \mathcal{D}$ be a two-cluster partitioning of the dataset $\mathcal{D}$ so $\mathcal{D}_2 = \mathcal{D} \setminus \mathcal{D}_1$. It is known that the two-cluster partitioning that minimizes the sum-of-squared errors (used in the 2-means algorithm)

$$J(\mathcal{D}_1, \mathcal{D}_2, c_1, c_2) = \sum_{k=1}^{2} \sum_{i \in \mathcal{D}_k} (x_i - c_k)^2 \tag{11.22}$$

groups the n samples at $x = 0$ with the one at $x = a$ (i.e., $\mathcal{D}_2 = \{0, \ldots, 0, a\}$) if

$$a^2 < f(n)$$

where $f : \mathbb{N} \to \mathbb{R}$ is a function of n. Find this function.

(ii) What happens to the clustering result if $a^2 > f(n)$? What if $a^2 = f(n)$?

Exercise 11.3. *We are given a dataset $\mathcal{D}$ with a finite number of points. We initialize the K-means clustering algorithm by first partitioning the points into K disjoint (non-overlapping) clusters $\mathcal{D}_1, \ldots, \mathcal{D}_K$. New centroids are then computed using* (11.5)*. The points are then re-assigned to* different *clusters resulting in a new partition $\mathcal{D}'_1, \ldots, \mathcal{D}'_K$ which is* different *from the original one $\mathcal{D}_1, \ldots, \mathcal{D}_K$. The process continues* ad infinitum*. Will the algorithm ever revisit $\mathcal{D}_1, \ldots, \mathcal{D}_K$? Why or why not?*

Exercise 11.4. *Consider a problem in which the samples are one-dimensional.*

(i) Derive an analogue of the K-means algorithm when the sum-of-squares cost function in (11.3) *is replaced by*

$$\widetilde{J}(\{w_{ik}\}, \{c_k\}) = \sum_{i=1}^{m} \sum_{k=1}^{K} w_{ik} |x_i - c_k|. \tag{11.23}$$

Note that the squared difference $(x_i - c_k)^2$ has been replaced by the absolute difference $|x_i - c_k|$.

(ii) Find a one-dimensional dataset for which the final groups of points that are produced using the usual K-means algorithm are different from that using the criterion in (11.23) *given the same initialization.*

The final two questions are a bit challenging, but they are good mathematical character-building exercises. The following problem is modified with permission from Prof. Jonathan Scarlett (SoC, Maths, NUS).

Exercise 11.5 (Modified from Final Exam 2018/9)**.** *Suppose that we apply the K-means clustering algorithm with $K = 2$ to a one-dimensional data set with four points: $x_1 = 1$, $x_2 = 2$, $x_3 = 4$, and $x_4 = 8$. Assume that we initialize the $K = 2$ centroids by choosing them to be* distinct *points in* $[1, 8]$.

(i) How many pairs of cluster centroids (c_1, c_2) can the algorithm possibly terminate with? Justify (prove) your answer carefully.

(ii) Which are the pairs?

(iii) Why does the algorithm terminate when any such pair identified in Part (ii) is encountered?

Exercise 11.6 (Final Exam 2016/7)**.** *Suppose a set of m distinct samples $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ is partitioned into K disjoint (non-overlapping) clusters $\mathcal{D}_1, \ldots, \mathcal{D}_K$. The sum-of-squared errors criterion used in K-means is the following expression:*

$$J(\mathcal{D}_1, \ldots, \mathcal{D}_K, \mathbf{c}_1, \ldots, \mathbf{c}_K) = \sum_{\substack{k=1: \\ \mathcal{D}_k \neq \emptyset}}^{K} \sum_{i \in \mathcal{D}_k} \|\mathbf{x}_i - \mathbf{c}_k\|^2. \tag{11.24}$$

Observe that the outer sum involves summing only over the nonempty subsets. Here, as usual, $\mathbf{c}_k$ is the average of the points in $\mathcal{D}_k$; see (11.5)*. Assume that $m \geq K$. Prove that there are no empty subsets in a partition that minimizes J as defined in* (11.24)*. You may want to do this by contradiction.*

Chapter 12

Deep Neural Networks and Backpropagation

In this final chapter (how sad!), we provide a brief description of deep neural networks in the form of feedforward neural networks and multi-layer perceptrons. Before we get started though, we remind the reader of Jacobians and the matrix version of the chain rule. We provide a toy example to show the utility of nonlinearities in neural networks. We also describe the backpropagation algorithm for multi-layer perceptrons in detail. Finally, we provide a few practice problems for you to familiaize yourself with the concepts. The exposition here is based partly on slides by Shubhendu Trivedi & Risi Kondor of the University of Chicago. Section 12.5 is based on notes by Prof. Adrian Roellin of NUS' Department of Statistics and Applied Probability (DSAP). For more information on this fascinating topic, please refer to Bishop's books [Bis98, Bis08] and Goodfellow, Bengio, and Courville's book [GBC16].

12.1 Preliminaries

For a multivariate function with multiple outputs $f : \mathbb{R}^n \to \mathbb{R}^m$, we use $Df : \mathbb{R}^n \to \mathbb{R}^{m\times n}$ to denote its *Jacobian matrix*, the $m \times n$ matrix of all its first-order partial derivatives. That is, for every $\mathbf{x} \in \mathbb{R}^n$,

$$Df(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{12.1}$$

Note that the entries are functions of $\mathbf{x}$. When we want to emphasize that the Jacobian is taken with respect to a certain parameter $\boldsymbol{\theta}$ (keeping other parameters fixed), we denote it as $D_{\boldsymbol{\theta}} f$.

When we talk about the backpropagation algorithm to train neural networks in Section 12.5, we will have to use the *matrix version of the chain rule*, which you may not seen before. We restate it here for ease of reference. Suppose we have functions $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^m \to \mathbb{R}^k$ such that the range of f is the domain of g. We are then allowed to construct a new function $h : \mathbb{R}^n \to \mathbb{R}^k$ prescribed by the recipe $h(\mathbf{x}) = (g \circ f)(\mathbf{x}) = g(f(\mathbf{x}))$ for each $\mathbf{x} \in \mathbb{R}^n$. The function h is known as the *composition* of g and f. The Jacobian of h (evaluated at $\mathbf{x}$) is

$$Dh(\mathbf{x}) = \big(Dg(f(\mathbf{x}))\big) \cdot (Df(\mathbf{x})), \tag{12.2}$$

where $\cdot$ means usual matrix multiplication. Note that this makes sense as for each $\mathbf{x}$, $Df(\mathbf{x})$ is a matrix of size $m \times n$ and $Dg(f(\mathbf{x}))$ is a matrix of size $k \times m$ so the matrices involved in the product in (12.2) are compatible. The matrix chain rule involving Jacobians is completely analogous to the usual chain rule $h'(x) = g'(f(x))f'(x)$ for $h = g \circ f$.

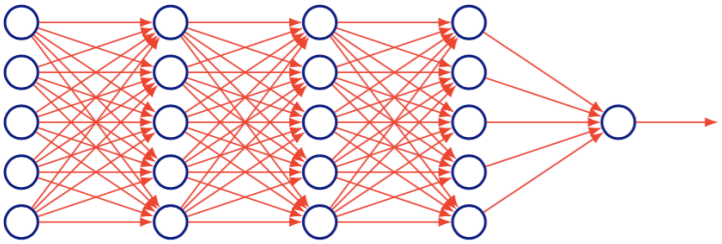


Figure 12.1: A feedforward network

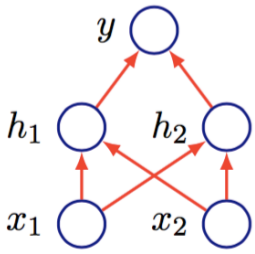


Figure 12.2: Architecture for the XOR example

12.2 Feedforward Networks With Multiple Neurons

As we have seen throughout the course, the overarching goal in machine learning is to approximate some unknown and ideal function $f^* : \mathbb{R}^d \to \mathcal{Y} \subset \mathbb{R}$ using training samples. For example, if $\mathcal{Y} = \{-1, +1\}$, then we want to construction an ideal classifier $y = f^*(\mathbf{x})$ that takes in an example $\mathbf{x}$ and outputs a label y. In a feedforward neural network, our functions are parametrized as $y = f(\mathbf{x}; \boldsymbol{\theta})$ and we would like to learn $\boldsymbol{\theta}$ to get a good approximation to f^* from a bunch of training examples $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$. The reason why we use the term "feedforward" is because usually the target function f is a composition of many different functions

$$f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}))). \tag{12.3}$$

See Fig. 12.1. Here, $f^{(1)}$ represents the operation undertaken by the first layer; $f^{(2)}$ represents the operation undertaken by the second layer and so on. The choices of $f^{(i)}$ are usually inspired by neuroscience and are colloquially called *neurons*. We will see how to construct $f^{(i)}$ carefully in the following. In particular, we argue through an example that incorporating *nonlinearities* within $f^{(i)}$ is beneficial.

12.3 The XOR Example Revisited

In this section, we provide a simple toy example to demonstrate the power of using nonlinearities in a multi-layer network. Consider learning the XOR function which is given by the following training dataset

$$\begin{aligned}
\mathbf{x}_1 &= \begin{bmatrix} -1 & -1 \end{bmatrix}^\top, \qquad y_1 = -1, \\
\mathbf{x}_2 &= \begin{bmatrix} -1 & +1 \end{bmatrix}^\top, \qquad y_2 = +1, \\
\mathbf{x}_3 &= \begin{bmatrix} +1 & -1 \end{bmatrix}^\top, \qquad y_3 = +1, \\
\mathbf{x}_4 &= \begin{bmatrix} +1 & +1 \end{bmatrix}^\top, \qquad y_4 = -1.
\end{aligned} \tag{12.4}$$

We have already seen (in Section 6.4) that we can use polynomial kernels (of order 2) to find a decision boundary or surface that classifies these points with zero training error. Here, we consider another approach

based on feedforward networks. Consider the squared loss function

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{4} \big(f^*(\mathbf{x}_i) - f(\mathbf{x}_i;\boldsymbol{\theta})\big)^2. \tag{12.5}$$

We need to choose a particular form for the parametric function $f(\mathbf{x};\boldsymbol{\theta})$. We have already seen that if $f(\mathbf{x};\mathbf{w},b) = \mathbf{x}^\top\mathbf{w} + b$, i.e., f is an affine function (with parameters $\boldsymbol{\theta} = (\mathbf{w},b)$), this will not work as the dataset is not linearly separable. Let's look at another architecture given by Fig. 12.2. In particular, choose

$$f^{(1)}(\mathbf{x};\mathbf{W},\mathbf{c}) = \mathbf{h} = (h_1,h_2) \tag{12.6}$$

giving the outputs for the hidden layer and

$$y = f^{(2)}(\mathbf{h};\mathbf{w},b). \tag{12.7}$$

Thus the complete model is given by

$$y = f(\mathbf{x};\mathbf{W},\mathbf{c},\mathbf{w},b) = f^{(2)}(f^{(1)}(\mathbf{x})) \tag{12.8}$$

with parameters collated in $\boldsymbol{\theta} = (\mathbf{W},\mathbf{c},\mathbf{w},b)$. So what should the function $f^{(1)}$ be? We will see that some *nonlinearity* is required to drive the training error to zero. Let's consider $f^{(1)}$ to be a composition of an affine function and a nonlinear activation function. We set the nonlinear activation function to be $\sigma(z) = \max\{0,z\}$. This is commonly known as the Rectified Linear Unit or ReLU. Thus, $f^{(1)}$ takes the form $f^{(1)}(\mathbf{x}) = \sigma(\mathbf{W}^\top\mathbf{x} + \mathbf{c})$ and the complete network model is thus given by

$$y = f(\mathbf{x};\mathbf{W},\mathbf{c},\mathbf{w},b) = \mathbf{w}^\top\sigma(\mathbf{W}^\top\mathbf{x} + \mathbf{c}) + b. \tag{12.9}$$

The activation function in (12.9) is applied element-wise. Consider the following choices of the parameters:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 1 & -1 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} 2 \\ -6 \end{bmatrix}, \quad b = -1. \tag{12.10}$$

The design matrix based on the data points in (12.4) (no offset) is

$$\mathbf{X} = \begin{bmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{bmatrix}. \tag{12.11}$$

Let us compute the intermediate operations in (12.9). We have

$$\mathbf{K} := \mathbf{X}\mathbf{W} + \begin{bmatrix} \mathbf{c} \\ \mathbf{c} \\ \mathbf{c} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} -1 & -3 \\ 1 & -1 \\ 1 & -1 \\ 3 & 1 \end{bmatrix} \qquad \text{and} \qquad \mathbf{H} = \sigma(\mathbf{K}) = \max\{\mathbf{K},0\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 3 & 1 \end{bmatrix}. \tag{12.12}$$

Each row corresponds to the output of the first layer for each sample, for which there are 4 of them. Now we compute our predictions at the second layer as

$$\mathbf{H}\mathbf{w} + \begin{bmatrix} b \\ b \\ b \\ b \end{bmatrix} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ -1 \end{bmatrix}. \tag{12.13}$$

Observe that the network has predicted the labels of the four samples exactly. Hence, the cost function $J(\boldsymbol{\theta})$ (defined in (12.5)) with parameters $\boldsymbol{\theta}$ equal to those in (12.10), equals to 0. Voila! We have achieved zero training error with a two-layer neural network with the ReLU activation function. Note the importance of the nonlinearity in the network; this nonlinearity is induced by σ. Finally, we remark that the choice of parameters in (12.10) to achieve the desired result in (12.13) is not unique.

12.4 On Universality and Depth

As mentioned, the first and second layers of the feedforward network admit the following expressions:

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \tag{12.14}$$

$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{W}_2\mathbf{h}^{(1)} + \mathbf{b}_2). \tag{12.15}$$

How to we decide on the depth and width of neural networks? How many layers suffice? What is the quality of the approximation of functions within a certain class if we use a certain number of layers and a certain number of nodes per layer. The answers to these questions constitute a very active area of research. Let us describe two results qualitatively.

A seminal theoretical result by Cybenko [Cyb89] states that a 2-layer neural network with linear output with some squashing nonlinearity in hidden units can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!). The implication of this is that regardless of function we are trying to learn, we know a large MLP can represent this function. But it is not guaranteed that our training algorithm will be able to learn that function It also gives no guidance on how large the network will be (exponential size in worst case).

Montúfar et al. [MPCB14] showed that the number of linear regions carved out by a deep rectifier network with d inputs, depth l and n units per hidden layer is $O\Big(\binom{n}{d}^{d(l-1)} n^d\Big)$. This is exponential in depth. They thus showed functions representable with a deep rectifier network can require an exponential number of hidden units with a shallow network. Research in this area is also growing exponentially by the day.

12.5 Backpropagation to Train Feedforward Networks

This part of the lecture notes is based on handwritten notes by Prof. Adrian Roellin of NUS DSAP. I sincerely thank Prof. Roellin for allowing me to reproduce his clear derivations.

12.5.1 Training a Single-Layer Feedforward Network

Now, we discuss how to train a feedforward network. Once again, we have a set of training data $\mathcal{D} := \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$ and each $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{y}_i$ can be a vector, say in $\mathbb{R}^{c\times 1}$; this is akin to multi-class classification where we have multiple outputs. But here, we focus on multiple regression. We have a certain per-sample loss function $\mathrm{Loss}(\mathbf{y}, \mathbf{y}')$, e.g., $\mathrm{Loss}(\mathbf{y}, \mathbf{y}') = \|\mathbf{y} - \mathbf{y}'\|^2$. Our aim is to minimize the total loss

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{m} \mathrm{Loss}(f(\mathbf{x}_i; \boldsymbol{\theta}), \mathbf{y}_i) \tag{12.16}$$

Let's first consider a single-layer feedforward network where neuron indexed by $j \in \{1, \ldots, c\}$ has output $\sigma\big(\sum_{i=1}^d w_{j,i}x_i + b_j\big)$, Here $\mathbf{W} = [w_{j,i}] \in \mathbb{R}^{c\times d}$ is the weight matrix, $\mathbf{b} = (b_1, b_2, \ldots, b_c) \in \mathbb{R}^{c\times 1}$ is the bias vector and $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function (e.g., the ReLU). Thus, the overall prediction from this single layer is

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^{c\times 1}, \tag{12.17}$$

where $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$ and we let $q \in \mathbb{N}$ the dimension of $\boldsymbol{\theta}$. Verify that $q = c(d + 1)$. In order to implement stochastic gradient descent,[1] we need to calculate

$$\nabla J(\boldsymbol{\theta}; i) = D_{\boldsymbol{\theta}}\big(\mathrm{Loss}(f(\mathbf{x}_i; \boldsymbol{\theta}), \mathbf{y}_i)\big), \tag{12.18}$$

[1]Stochastic gradient descent is just like gradient descent we have learned in Chapter 8 except that we move in the direction of a gradient vector evaluated at a single data point in (12.16), namely $\nabla J(\boldsymbol{\theta}; i)$ in (12.18). Such a gradient is unbiased but has high variance (why?). Nevertheless, since only one data point is involved, as opposed to all $m \gg 1$, the computational complexity is manageable. There are other optimization algorithms that are more advanced than stochastic gradient descent. One such variant is the limited-memory Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm which is used to optimize the MLPClassifier in scikit-learn in Python. An improvement of the limited memory BFGS algorithm can be found here [ZHT18].

where $D_{\boldsymbol{\theta}}$ is the *Jacobian operator* and we are evaluating it at the i-th sample. By the chain rule as stated in in (12.2), this becomes

$$\nabla J(\boldsymbol{\theta}; i) = D_{\mathbf{y}}\big(\text{Loss}(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i)\big) \cdot D_{\boldsymbol{\theta}}\big(f(\mathbf{x}_i;\boldsymbol{\theta})\big). \tag{12.19}$$

Note that $D_{\mathbf{y}}\big(\text{Loss}(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i)\big) \in \mathbb{R}^{1\times c}$ and $D_{\boldsymbol{\theta}}\big(f(\mathbf{x}_i;\boldsymbol{\theta})\big) \in \mathbb{R}^{c\times q}$. If $\text{Loss}(\mathbf{y}, \mathbf{y}') = \|\mathbf{y} - \mathbf{y}'\|^2$, the squared l_2 loss, the Jacobian can be calculated to be

$$D_{\mathbf{y}}\big(\text{Loss}(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i)\big) = 2\begin{bmatrix} f_1(\mathbf{x}_i, \boldsymbol{\theta}) - y_{i1} & \dots & f_c(\mathbf{x}_i, \boldsymbol{\theta}) - y_{ic}\end{bmatrix} \in \mathbb{R}^{1\times c}. \tag{12.20}$$

Now we deal with the second term using the chain rule as stated in in (12.2) as follows:

$$D_{\boldsymbol{\theta}}\big(f(\mathbf{x}_i;\boldsymbol{\theta})\big) = D_{\boldsymbol{\theta}}\big(\sigma(\mathbf{W}\mathbf{x}_i + \mathbf{b})\big) \tag{12.21}$$

$$= D_{\mathbf{x}}\big(\sigma(\mathbf{W}\mathbf{x}_i + \mathbf{b})\big) \cdot D_{\boldsymbol{\theta}}\big(\mathbf{W}\mathbf{x}_i + \mathbf{b}\big). \tag{12.22}$$

We again note that $D_{\mathbf{x}}\big(\sigma(\mathbf{W}\mathbf{x}_i + \mathbf{b})\big) \in \mathbb{R}^{c\times c}$ and $D_{\boldsymbol{\theta}}\big(\mathbf{W}\mathbf{x}_i + \mathbf{b}\big) \in \mathbb{R}^{c\times q}$. Now we have

$$D_{\mathbf{x}}\big(\sigma(\mathbf{W}\mathbf{x}_i + \mathbf{b})\big) = \begin{bmatrix} \sigma'\big((\mathbf{W}\mathbf{x}_i + \mathbf{b})_1\big) & 0 & \dots & 0 \\ 0 & \sigma'\big((\mathbf{W}\mathbf{x}_i + \mathbf{b})_2\big) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \sigma'\big((\mathbf{W}\mathbf{x}_i + \mathbf{b})_c\big) \end{bmatrix} \in \mathbb{R}^{c\times c}. \tag{12.23}$$

Note that for the ReLU activation function, $\sigma'(z) = 1$ if $z \geq 0$ and 0 otherwise.[2] Vectorizing the weight matrix as $\mathbf{W} = (w_{1,1}, \dots, w_{1,d}, w_{2,1}, \dots, w_{2,d}, \dots, w_{c,1}, \dots, w_{c,d}) \in \mathbb{R}^{cd}$, we obtain

$$D_{\mathbf{W}}\big(\mathbf{W}\mathbf{x}_i + \mathbf{b}\big) = \begin{bmatrix} x_{i,1} & \dots & x_{i,d} & 0 & \dots & 0 & \dots & 0 & \dots & 0 \\ 0 & \dots & 0 & x_{i,1} & \dots & x_{i,d} & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & \dots & x_{i,1} & \dots & x_{i,d} \end{bmatrix} \in \mathbb{R}^{c\times cd} \tag{12.24}$$

and

$$D_{\mathbf{b}}\big(\mathbf{W}\mathbf{x}_i + \mathbf{b}\big) = \mathbf{I}_c \in \mathbb{R}^{c\times c}, \tag{12.25}$$

where $\mathbf{I}_c$ is the $c \times c$ identity matrix. Hence,

$$D_{\boldsymbol{\theta}}\big(\mathbf{W}\mathbf{x}_i + \mathbf{b}\big) = [D_{\mathbf{W}}\big(\mathbf{W}\mathbf{x}_i + \mathbf{b}\big), D_{\mathbf{b}}\big(\mathbf{W}\mathbf{x}_i + \mathbf{b}\big)] \in \mathbb{R}^{c\times(d+1)c} = \mathbb{R}^{c\times q}. \tag{12.26}$$

In summary, we can compute the sample-wise gradient explicitly

$$\nabla J(\boldsymbol{\theta}; i) = D_{\mathbf{y}}\big(\text{Loss}(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i)\big) \cdot D_{\mathbf{x}}\big(\sigma(\mathbf{W}\mathbf{x}_i + \mathbf{b})\big) \cdot D_{\boldsymbol{\theta}}\big(\mathbf{W}\mathbf{x}_i + \mathbf{b}\big) \in \mathbb{R}^{1\times q}. \tag{12.27}$$

This gradient can then be used within a gradient descent or stochastic gradient descent framework to optimize the training loss in (12.16).

12.5.2 Training a Multi-Layer Feedforward Network

Now, we derive the equations to train a multi-layer feedforward network, also known as a multi-layer perceptron (MLP). See Fig. 12.3. We denote the number of layers or depth as l, the dimension of the input feature vectors as d_0, the length of the vectors at each hidden layer as d_k where $1 \leq k \leq l$. Note first that the total number of parameters is $(d_0+1)d_1 + (d_1+1)d_2 + \ldots + (d_{l-1}+1)d_l \approx ld^2$ if $d_k \approx d$ for all $1 \leq k \leq l$. Note that

$$\mathbf{z}_{k+1} = \sigma(\mathbf{W}_{k+1}\mathbf{z}_k + \mathbf{b}_{k+1}). \tag{12.28}$$

[2]Here, we have been a bit casual with what happens at $z = 0$ because $\sigma(\,\cdot\,) = \max\{\,\cdot\,, 0\}$ is not differentiable at 0 so $\sigma'(\,\cdot\,)$ does not really make sense at 0.

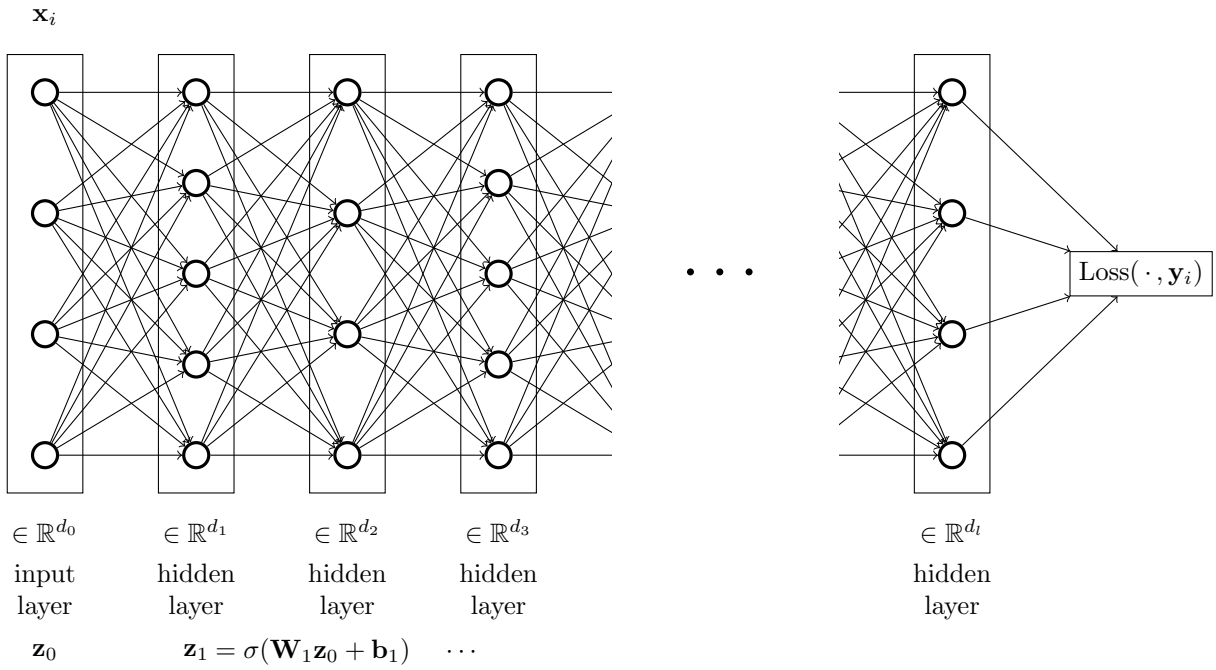


Figure 12.3: Multi-Layer Feedforward Network

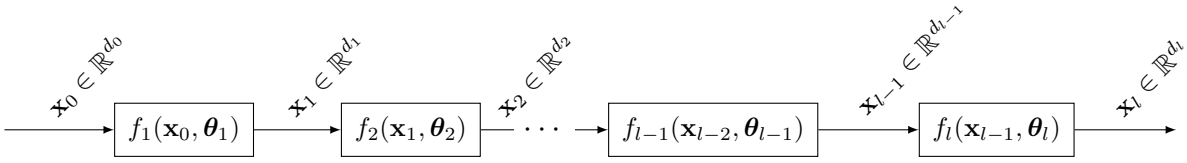


Figure 12.4: Backpropagation

First, let us calculate $\nabla J(\boldsymbol{\theta}, i)$, the gradient with respect to a single sample indexed by i. By repeated applications of the chain rule in (12.2), we obtain

$$\begin{aligned} D_{\mathbf{W}_k}\left[J(\boldsymbol{\theta}, i)\right] = D_{\mathbf{y}}(\mathrm{Loss}(\mathbf{z}_l, \mathbf{y}_i)) \cdot D_{\mathbf{x}}(\mathbf{W}_l\mathbf{z}_{l-1} + \mathbf{b}_l) \cdot D_{\mathbf{z}}(\mathbf{W}_l\mathbf{z}_{l-1} + \mathbf{b}_l) \\ \cdot D_{\mathbf{x}}(\mathbf{W}_{l-1}\mathbf{z}_{l-2} + \mathbf{b}_{l-1}) \cdot D_{\mathbf{z}}(\mathbf{W}_{l-1}\mathbf{z}_{l-2} + \mathbf{b}_{l-1}) \\ \vdots \\ \cdot D_{\mathbf{x}}(\mathbf{W}_k\mathbf{z}_{k-1} + \mathbf{b}_k) \cdot D_{\mathbf{W}_k}(\mathbf{W}_k\mathbf{z}_{k-1} + \mathbf{b}_k), \end{aligned} \tag{12.29}$$

where the individual Jacobians are $D_{\mathbf{y}}(\mathrm{Loss}(\mathbf{z}_l, \mathbf{y}_i)) \in \mathbb{R}^{1\times d_l}$, $D_{\mathbf{x}}(\mathbf{W}_k\mathbf{z}_{k-1}+\mathbf{b}_k) \in \mathbb{R}^{d_k\times d_k}$, $D_{\mathbf{z}}(\mathbf{W}_l\mathbf{z}_{l-1}+\mathbf{b}_l) \in \mathbb{R}^{d_l\times d_{l-1}}$, and $D_{\mathbf{W}_k}(\mathbf{W}_k\mathbf{z}_{k-1} + \mathbf{b}_k) \in \mathbb{R}^{d_k\times(d_k d_{k-1})}$.

Equipped with $D_{\mathbf{W}_k}\left[J(\boldsymbol{\theta}, i)\right]$, we now implement *backpropagation*. See Fig. 12.4 for the notations. Our aim is to minimize the output (denoted here as $\mathbf{x}_l$) over $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_l)$ where $\boldsymbol{\theta}_k$ (for $1 \le k \le l$) denotes the parameter vector at the k-th layer. In the forward pass, we recursively evaluate

$$f_k(\mathbf{x}_{k-1}, \boldsymbol{\theta}_k), \quad D_{\mathbf{x}_{k-1}}\left[f_k(\mathbf{x}_{k-1}, \boldsymbol{\theta}_k)\right], \quad \text{and} \quad D_{\boldsymbol{\theta}_k}\left[f_k(\mathbf{x}_{k-1}, \boldsymbol{\theta}_k)\right]. \tag{12.30}$$

In the backward pass, we recursively update the $\boldsymbol{\theta}_k$'s with some learning rate $\lambda > 0$:

- Update $\boldsymbol{\theta}_l$ as follows:

$$\boldsymbol{\theta}_l \longleftarrow \boldsymbol{\theta}_l - \lambda D_{\boldsymbol{\theta}_l}\left[f_l(\mathbf{x}_{l-1}, \boldsymbol{\theta}_l)\right]^\top, \tag{12.31}$$

where $D_{\boldsymbol{\theta}_l}\left[f_l(\mathbf{x}_{l-1}, \boldsymbol{\theta}_l)\right]^\top \in \mathbb{R}^{d_l \times \dim(\boldsymbol{\theta}_l)}$;

- Pass $\mathbf{z}_l = D_{\mathbf{x}_{l-1}}\left[f_l(\mathbf{x}_{l-1}, \boldsymbol{\theta}_l)\right] \in \mathbb{R}^{d_l \times d_{l-1}}$ to the previous layer;
- Update $\boldsymbol{\theta}_{l-1}$ as follows:

$$\boldsymbol{\theta}_{l-1} \longleftarrow \boldsymbol{\theta}_{l-1} - \lambda \left(\mathbf{z}_l \cdot D_{\boldsymbol{\theta}_{l-1}}\left[f_{l-1}(\mathbf{x}_{l-2}, \boldsymbol{\theta}_{l-1})\right]\right)^\top, \tag{12.32}$$

  where $\mathbf{z}_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $D_{\boldsymbol{\theta}_{l-1}}\left[f_{l-1}(\mathbf{x}_{l-2}, \boldsymbol{\theta}_{l-1})\right] \in \mathbb{R}^{d_{l-1} \times \dim(\boldsymbol{\theta}_l)}$;
- Pass $\mathbf{z}_{l-1} = \mathbf{z}_l \cdot D_{\mathbf{x}_{l-2}}\left[f_{l-1}(\mathbf{x}_{l-2}, \boldsymbol{\theta}_{l-1})\right] \in \mathbb{R}^{d_{l-1} \times d_{l-2}}$ to the previous layer;
- Repeat until

$$\boldsymbol{\theta}_1 \longleftarrow \boldsymbol{\theta}_1 - \lambda \left(\mathbf{z}_2 \cdot D_{\boldsymbol{\theta}_1}[f_1(\mathbf{x}_0, \boldsymbol{\theta}_1)]\right)^\top. \tag{12.33}$$

Because we start by updating $\boldsymbol{\theta}_l$ all the way back to $\boldsymbol{\theta}_1$ (from the right to the left of Fig. 12.4), this is known as *backpropagation*.

Observe that if we use the sigmoid $\sigma(a) = 1/(1+\exp(-a))$ as the activation function there will be a problem in (12.23) and (12.31)–(12.33) because its gradient is very small for $|a|$ large (graph the sigmoid function to understand why). Hence, after many backpropagation steps, small values outside the "active" range of σ (i.e., the actives range is when $|a|$ small) will accumulate so that the gradient for weights in earlier layers vanishes (the vanishing gradients problem). This is partly ameliorated by the ReLU ($\sigma(a) = \max\{a, 0\}$) or the leaky ReLU ($\sigma(a) = a$ for $a \geq 0$ and $\sigma(a) = 0.01a$ for $a < 0$), hence their popularity in recent years.

12.6 Practice Problems

No computers and programming are required to do these problems.

Exercise 12.1. *Let $f : \mathbb{R}^2 \to \mathbb{R}^3$ and $g : \mathbb{R}^3 \to \mathbb{R}$ be defined as*

$$f(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix}, \quad \text{and} \quad g(u, v, w) = e^w \sin(u)\cos(v). \tag{12.34}$$

Define the vector-input, scalar-output function $h(\mathbf{x}) = (g \circ f)(\mathbf{x})$. Use the matrix version of the chain rule to find $Dh(\mathbf{x})$ for each $\mathbf{x} \in \mathbb{R}^2$.

The following problem is taken from Bishop's book [Bis08, Exercise 5.1].

Exercise 12.2. *Consider any l-layer network with activation functions given by the sigmoid*

$$\sigma(a) = \frac{1}{1+\exp(-a)}. \tag{12.35}$$

Given any inputs, prove that there exists an equivalent network, which computes exactly the same outputs, but with activation functions given by the hypertangent

$$\tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}. \tag{12.36}$$

Exercise 12.3. *This problem reinforces concepts learned in Section 12.3. We would like to design a multi-layer perceptron (MLP) with 2 layers whose inputs are three binary-valued (i.e., 0 or 1) variables x_1, x_2 and x_3. The first layer has two hidden nodes h_1 and h_2. So the MLP looks like that in Fig. 12.2 except that there are three (instead of two) inputs. The MLP should output 1 if exactly one of the three inputs is equal 1, and outputs 0 otherwise. All of the units use the following hard threshold activation function:*

$$\sigma(a) = \begin{cases} 1 & a \geq 0 \\ 0 & a < 0 \end{cases}. \tag{12.37}$$

Note that there is one weight matrix $\mathbf{W} \in \mathbb{R}^{2\times 3}$ *and one bias (row) vector* $\mathbf{c} \in \mathbb{R}^{1\times 2}$ *that connects the input and the first layer. There is another weight matrix* $\mathbf{w} \in \mathbb{R}^{1\times 2}$ *and one bias scalar* $b \in \mathbb{R}$ *that connects the first layer and and the second layer. Specify feasible weights* $(\mathbf{W}, \mathbf{w})$ *and biases* $(\mathbf{c}, b)$ *to implement this function.*

Exercise 12.4. *Consider an MLP with* 2 *layers. The input variables are continuous, the first layer uses a hard threshold holding function as defined in* (12.37)*. The second layer uses the sigmoid as defined in* (12.35) *and the loss function to be minimized is the squared* l_2 *loss. Discuss what will go wrong if you train this network using the backpropagation algorithm described in Section 12.5.2.*

Bibliography

[AV07] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1027–1035, 2007.

[Bis98] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1998.

[Bis08] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2008.

[BT02] D. P. Bertsekas and J. N. Tsitsiklis. *Introduction to Probability*. Athena Scientific, 1st edition, 2002.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CGPR20] D. Choo, C. Grunau, J. Portmann, and V. Rozhoň. k-means++: Few more steps yield constant approximation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–313, 1989.

[FS97] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119– 139, 1997.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press;, 2016.

[Hes98] T. Heskes. Bias/variance decompositions for likelihood-based estimators. *Neural Computation*, 10(6):1425–1433, Aug 1998.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction,*. Springer Series in Statistics, 3rd edition, 2009.

[KW70] G. Kimeldorf and G. Wahba. A correspondence between Bayesian estimation of stochastic processes and smoothing by splines. *Ann. Math. Statist.*, 41:495–502, 1970.

[LSST+02] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, Feb 2002.

[MPCB14] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2924–2932, 2014.

[PP12] K. B. Petersen and M. S. Pedersen. *The Matrix Cookbook*. 2012. `https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf`.

[Ros12] S. Ross. *A First Course in Probability*. Pearson, 9th edition, 2012.

[SHS01] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In *International Conference on Computational Learning Theory*, pages 416–426. Lecture Notes in Computer Science, 2001.

[Sil12] N. Silver. *The Signal and the Noise: The Art and Science of Prediction*. Penguin Books Ltd, 2012.

[SSB18] B. Schölkopf, A. J. Smola, and F. Bach. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2018.

[Sta16] B. Stacey. Fukushima: The failure of predictive models. *MPRA Paper No. 69383*, 2016. Online at https://mpra.ub.uni-muenchen.de/69383/.

[Str16] G. Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 5th edition, 2016.

[Tib96] R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (methodological)*, 58(1):267–288, 1996.

[TVT21] S. S. Y. Tan, A. Varvitsiotis, and V. Y. F. Tan. Analysis of optimization algorithms via sum-of-squares. *Journal of Optimization Theory and Applications*, 2021.

[Vap95] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Berlin: Springer-Verlag, 1995.

[XFTF19] R. Xia, L. Filstroff, V. Y. F. Tan, and C. Févotte. A ranking model motivated by nonnegative matrix factorization with applications to tennis tournaments. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 187–203, 2019.

[ZHT18] R. Zhao, W. B. Haskell, and V. Y. F. Tan. Stochastic L-BFGS revisited: Improved convergence rates and practical acceleration strategies. *IEEE Transactions on Signal Processing*, 66(5):1155–1169, May 2018.