SB2.2/SM4 Statistical Machine Learning
Lecture notes - Supervised Learning

François Caron

University of Oxford, Hilary Term 2021

Version of March 7, 2021
Please report typos to caron@stats.ox.ac.uk.

Aims and Structure of this course

The aims of this course are

- To learn a number of different machine learning methods and gain understanding about their statistical foundations;
- To learn to identify and use the appropriate method for a given dataset and a given task;
- To learn how to use the relevant Python libraries/modules to analyse data, interpret results and evaluate the methods.

The course is structured in two main parts. In the first part, we will cover unsupervised learning (dimensionality reduction, feature extraction and clustering). In the second part of the course, we will cover supervised learning (classification and regression).

References

These lecture notes are partially based on the slides from previous lecturers who have taught this course, and on the following books:

- L. Wasserman. All of Statistics. Springer, 2010.
- K. Murphy. Machine Learning. A probabilistic perspective. The MIT Press, 2012
- C. M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.
- T. Hastie, R. Tibshirani, J. Friedman. The Elements of Statistical Learning. Springer, 2009.
- I. Goodfellow, Y. Bengio and A. Courville. Deep Learning. The MIT Press, 2016.

Software

This document includes a number of examples written in the programming language Python. Python is certainly the most popular programming language for machine learning; it can be freely downloaded and installed. If you are new to Python, the following website contains useful information for beginners: `https://www.python.org/about/gettingstarted/`.

A number of Python code editors are freely available: PyCharm, Spyder, Atom, Jupyter. You may want to consider installing Anaconda, which is a distribution of Python and R programming languages for data science and machine learning. It comes with code editors such as Spyder and Jupyter for Python.

Python contains a very rich collection of libraries/packages for data science/machine learning. We will use the following libraries in this course:

- Numpy and Scipy: Core toolboxes for scientific computing (mathematical functions, linear algebra routines, random number generators, optimisation, etc.)
- Matplotlib: Visualisation
- Pandas: Data manipulation
- Seaborn: Statistical visualisation
- Scikitlearn: Machine Learning
- Pytorch: Deep Learning

These libraries can be loaded with the following commands.

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import sklearn as skl
import torch
```

Background material

0.1 Notations

Let $x_1, \ldots, x_p$ be scalar values. A p-dimensional row vector is noted as $(x_1, \ldots, x_p)$. A p-dimensional column vector is noted as $(x_1, \ldots, x_p)^\intercal$. By default, a vector means a column vector. The L2 norm of a vector $x = (x_1, \ldots, x_p)^\intercal \in \mathbb{R}^p$ is defined as

$$\|x\| = \sqrt{x^\intercal x} = \sqrt{\sum_{j=1}^{p} x_j^2}$$

Let $X \in \mathcal{X} \subseteq \mathbb{R}^p$ be a random variable with cumulative distribution function F. Throughout this course, we will use the notation

$$\mathbb{E}[h(X)] = \int_{\mathcal{X}} \phi(x) dF(x) = \begin{cases} \int_{\mathcal{X}} \phi(x) f(x) dx & \text{if } X \text{ is a continuous random variable} \\ \sum_{x \in \mathcal{X}} \phi(x) f(x) & \text{if } X \text{ is a discrete random variable} \end{cases}$$

where f denotes the probability density function of X if X is a continuous random variable, or the probability mass function of X if X is a discrete random variable.

Definition 1 (Statistical Functional)**.** *Let F be a cumulative distribution function. A **statistical functional** is a map T that maps a cdf F to a real number (or vector) $T(F)$.*

For example, for a random variable X with distribution F, its median m and mean μ are both statistical functionals of F, with

$$m = F^{-1}(1/2), \quad \mu = \int_0^\infty x dF(x).$$

Let $X = (X_1, \ldots, X_p)^\intercal \in \mathbb{R}^p$ be a random vector. We denote $\mathbb{E}[X]$ the mean of X and

$$\text{cov}(X) = \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^\intercal]$$

the p-by-p covariance matrix of X. If $p = 1$, we write $\text{var}(X)$ for its variance. For a vector $a \in \mathbb{R}^p$,

$$\mathbb{E}[a^\intercal X] = a^\intercal \mathbb{E}[X]$$
$$\text{var}(a^\intercal X) = a^\intercal \text{cov}(X) a.$$

The probability density function (pdf) of a Gaussian random vector $X \in \mathbb{R}^p$ with mean $\mu \in \mathbb{R}^p$ and p-by-p covariance matrix Σ is

$$\varphi(x; \mu, \Sigma) = (2\pi)^{-p/2} |\Sigma|^{-1/2} \exp\left(-\frac{1}{2}(x - \mu)^\intercal \Sigma^{-1} (x - \mu)\right). \tag{1}$$

0.2 Derivatives

Let $J : \mathbb{R}^p \to \mathbb{R}$ be a scalar function. For a vector $x = (x_1, \ldots, x_p)^\intercal \in \mathbb{R}^p$, the vector of partial derivatives, or gradient, is the p-dimensional vector given by

$$\nabla_x J(x) = \frac{\partial J(x)}{\partial x} = \begin{pmatrix} \frac{\partial J(x)}{\partial x_1} \\ \frac{\partial J(x)}{\partial x_2} \\ \cdots \\ \frac{\partial J(x)}{\partial x_p} \end{pmatrix}. \tag{2}$$

Let $x, a \in \mathbb{R}^p$ and B be a p-by-p symmetric matrix. We have

$$\frac{\partial (x^\intercal a)}{\partial x} = \frac{\partial (a^\intercal x)}{\partial x} = a \tag{3}$$
$$\frac{\partial ((x - a)^\intercal B (x - a))}{\partial x} = \frac{\partial ((a - x)^\intercal B (a - x))}{\partial x} = 2B(x - a) \tag{4}$$

For a p-by-p matrix Σ, let $|\Sigma|$ denote the determinant of Σ. Let $a, b \in \mathbb{R}^p$. For a full-rank symmetric matrix Σ,

$$\frac{\partial \log |\Sigma|}{\partial \Sigma} = \Sigma^{-1} \tag{5}$$
$$\frac{\partial (a^\intercal \Sigma^{-1} b)}{\partial \Sigma} = -\Sigma^{-1} a b^\intercal \Sigma^{-1} \tag{6}$$

1 — Basics of Statistical learning theory for supervised learning

1.1 Notations and assumptions

Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ be the target space. We will focus here on the cases where $\mathcal{Y} = \mathbb{R}$, which corresponds to a regression task, or $\mathcal{Y}$ is a finite set with $K \geq 2$ elements, which corresponds to a classification task. By convention, we will assume that $\mathcal{Y} = \{1, \ldots, K\}$ if $K \geq 3$ (multiclass classification) and $\mathcal{Y} = \{1, -1\}$ if $K = 2$ (binary classification).

Definition 2 (Prediction rule)**.** *Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ be the target space. A **prediction rule** is a function $h : \mathcal{X} \to \mathcal{Y}$. Let $\mathcal{F} = \{h : \mathcal{X} \to \mathcal{Y}\}$ the set of all prediction rules.*

In the classification case, the prediction rule is called a classification rule, or a **classifier**. Without loss of generality, classifiers can be (non-uniquely) specified via a collection of real functions $(f_k)_{k=1,\ldots,K}$:

$$h(x) = \underset{k=1,\ldots,K}{\arg\max}\, f_k(x) \tag{1.1}$$

where $f_k : \mathcal{X} \to \mathbb{R}$, $k = 1, \ldots, K$ are called **discriminant functions**. $f_k(x)$ may be interpreted as the level of confidence for choosing class k for an input x. The multivariate function $\widetilde{h} : x \to (f_1(x), \ldots, f_K(x))$ is sometimes called a **soft classifier** (by contrast, h is then called a **hard classifier**). For two classes k and k', the set of solutions to the equation $f_k(x) = f_{k'}(x)$ is called the **decision boundary** between these two classes. For a binary classifier h, one can consider a single discriminant function $f : \mathcal{X} \to \mathbb{R}$

$$h(x) = \begin{cases} 1 & \text{if } f(x) \geq 0 \\ -1 & \text{if } f(x) < 0. \end{cases} \tag{1.2}$$

The decision boundary between the two classes of the binary classifier is given by the solutions to the equation $f(x) = 0$.

Definition 3 (Linear prediction rule)**.** *A **linear prediction rule** is such that*

$$h(x) = \beta_0 + \beta^\intercal x \tag{1.3}$$

for regression, where $\beta_0 \in \mathbb{R}$, $\beta = (\beta_1, \ldots, \beta_p) \in \mathbb{R}^p$, and

$$f_k(x) = \beta_{k0} + \beta_k^\intercal x \tag{1.4}$$

for classification, where $\beta_{k0} \in \mathbb{R}$ and $\beta_k = (\beta_{k1}, \ldots, \beta_{kp}) \in \mathbb{R}^p$ for $k = 1, \ldots, K$. The decision boundary between two classes k and k' is the hyperplane defined by the equation

$$(\beta_{k0} - \beta_{k'0}) + (\beta_k - \beta_{k'})^\intercal x = 0.$$

Consider a dataset $d = (x_i, y_i)_{i=1,\ldots,n}$ where $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ is the ith input/target observation. $n \geq 1$ is the number of observations. Our objective is, based on the dataset d, to **learn** (or **train**) a prediction rule $\widehat{h}^{(d)} \in \mathcal{F}$. This prediction rule is then used, for a set of new examples with known input values $(x_{n+1}, x_{n+2}, \ldots)$, to **predict** their target values $(\widehat{y}_{n+1}, \widehat{y}_{n+2}, \ldots)$ using the **learned prediction rule**

$$\widehat{y}_{n+i} = \widehat{h}^{(d)}(x_{n+i}).$$

We give below two simple examples of learned prediction rules $\widehat{h}^{(d)}$ for regression and classification.

Example 4 (Least square linear regression)**.** *Consider univariate regression ($\mathcal{X} = \mathcal{Y} = \mathbb{R}$), with the least square prediction rule, defined as*

$$\widehat{h}^{(d)}(x) = \widehat{\beta}_0 + x\widehat{\beta}_1 \tag{1.5}$$

where the least square estimates $\widehat{\beta}_0$ and $\widehat{\beta}_1$ are defined as

$$\widehat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})y_i}{\sum_{i=1}^n (x_i - \bar{x})^2}, \quad \widehat{\beta}_0 = \bar{y} - \widehat{\beta}_1 \bar{x} \tag{1.6}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$. The prediction rule $\widehat{h}^{(d)}(x)$ is linear.

Example 5 (Histogram binary classifier)**.** *For $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \{-1, 1\}$ consider the histogram classifier*

$$\widehat{h}^{(d)}(x) = \begin{cases} 1 & \text{if } \widehat{f}^{(d)}(x) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

where the discriminant function $\widehat{f}^{(d)}$ is defined as

$$\widehat{f}^{(d)}(x) = \sum_{i=1}^n \mathbb{1}_{\lfloor x_i \rfloor = \lfloor x \rfloor}(\mathbb{1}_{y_i=1} - \mathbb{1}_{y_i=-1}).$$

This simple classifier is piecewise constant, and assigns a point x in the interval $[\lfloor x \rfloor, \lfloor x \rfloor + 1)$ to the most frequent class in that interval in the dataset d.

We assume that the dataset $d = (x_i, y_i)_{i=1,\ldots,n}$ is a realisation from a random sample $D = (X_i, Y_i)_{i=1,\ldots,n}$ where $(X_1, Y_1), \ldots, (X_n, Y_n)$ are independent and identically distributed (iid) random variables with the same distribution as (X, Y). Let P_0 denote the distribution of (X, Y), called the true data distribution, or population distribution. Note that $\widehat{h}^{(D)}$ therefore denotes a **random** prediction rule, as it is parameterised by the random sample D; the prediction rule $\widehat{h}^{(d)}$ is a realisation of $\widehat{h}^{(D)}$.

Decomposition of the variance. For classification with K classes, using the law of total variance, we have the following decomposition of the covariance matrix of the input vector X

$$\operatorname{cov}(X) = \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^\intercal] = \underbrace{\mathbb{E}[\operatorname{cov}(X \mid Y)]}_{\text{within-class covariance}} + \underbrace{\operatorname{cov}(\mathbb{E}[X \mid Y])}_{\text{between-class covariance}}. \tag{1.7}$$

The first term in the right-handside of (1.7) corresponds to the part of the total covariance matrix $\operatorname{cov}(X)$ originating from the variance **within** each class. The second term corresponds to the variance originating from the variance **between** the different classes. Let $\mu_k = \mathbb{E}[X|Y = k]$, $\Sigma_k = \operatorname{cov}(X \mid Y = k)$, $\mu = \mathbb{E}[X]$, and $\pi_k = \Pr(Y = k)$. The two covariance terms can be expressed as

$$B = \operatorname{cov}(\mathbb{E}[X \mid Y]) = \sum_{k=1}^K \pi_k(\mu_k - \mu)(\mu_k - \mu)^\intercal$$

$$W = \mathbb{E}[\operatorname{cov}(X \mid Y)] = \sum_{k=1}^K \pi_k \Sigma_k$$

If $X \in \mathbb{R}$, the ratio

$$\frac{\operatorname{cov}(\mathbb{E}[X \mid Y])}{\mathbb{E}[\operatorname{cov}(X \mid Y)]} \geq 0$$

between the between-class and within-class variances is a measure of the **separability** between the classes. Larger values indicate more separated classes, while small values indicate a lot of overlap between the classes. For instance, if $\Pr(Y = 1) = \Pr(Y = -1) = 1/2$ and $X \mid Y = y \sim \mathcal{N}(y\mu, \sigma^2)$, for some $\mu \geq 0$ and $\sigma > 0$, we have

$$\mathbb{E}[\operatorname{cov}(X \mid Y)] = \sigma^2, \quad \operatorname{cov}(\mathbb{E}[X \mid Y]) = \mu^2.$$

σ^2 therefore controls the within-class variance, and μ^2 the between-class variance.

In order to formalise mathematically the supervised learning task, we need to answer the following questions:

- How do we quantify the accuracy of a prediction rule?
- If we were to know the true data distribution, what would be the associated (optimal) prediction rule?
- Based on a dataset of size n, how can we learn a prediction rule whose accuracy is close to the optimal one?

1.2 Loss function and risk

Consider a prediction rule $h : \mathcal{X} \to \mathcal{Y}$ where $h(x) \in \mathcal{Y}$ is the predicted target for an input x. We need a way to measure how "bad" it is to predict $h(x)$ when the true target is indeed y. This will be done via the introduction of a loss function.

Definition 6. *Let $\mathcal{Y}$ be the target space. A **loss function** is a non-negative function*

$$L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+.$$

The loss $L(y, h(x))$ represents the "cost" of predicting a target $h(x)$ when the true target is y. We typically have $L(y, h(x)) = 0$ when $h(x) = y$, and $L(y, h(x))$ increases as y and $h(x)$ are "further away". Typical loss functions are the squared error loss

$$L(y, h(x)) = (y - h(x))^2 \tag{1.8}$$

for regression, and the $0 - 1$ loss for classification

$$L(y, h(x)) = \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases} \tag{1.9}$$

The $0 - 1$ loss assigns a cost of 1 if the predicted class is different from the true target, and 0 otherwise.

Definition 7 (Risk - supervised learning)**.** *For a given loss function L, the (population) **risk** $R(h)$ of a prediction rule h is the expected loss*

$$R(h) = \mathbb{E}\left[L(Y, h(X))\right]$$

where the expectation is taken over the true (unknown) joint distribution of (X, Y).

The risk is therefore given by

$$R(h) = \begin{cases} \mathbb{E}\left[(Y - h(X))^2\right] & \text{under the squared loss (1.8)} \\ \Pr(Y \neq h(X)) & \text{under the } 0-1 \text{ loss (1.9).} \end{cases}$$

$R(h)$ is therefore the mean squared error under the squared loss, and the probability of misclassification for classification under the 0-1 loss. We assume for simplicity that $R(h)$ admits a unique minimum over the set of prediction rules $\mathcal{F}$. The optimal prediction rule, called **Bayes prediction rule**, is the function h^* that minimises the risk $R(h)$.

Definition 8 (Bayes prediction rule)**.** *The Bayes prediction rule is given by*

$$h^* = \arg\min_{h \in \mathcal{F}} R(h). \tag{1.10}$$

Proposition 9. *For any $x \in \mathcal{X}$, we have*

$$h^*(x) = \arg\min_{y^* \in \mathcal{Y}} \mathbb{E}[L(Y, y^*)|X = x], \tag{1.11}$$

where the expectation is taken over the true (unknown) conditional distribution of Y given $X = x$.

Proof. Note that for any function f, $\mathbb{E}[f(X, Y)] = \mathbb{E}_X[\mathbb{E}_Y[f(X, Y)|X]]$. It follows that, for any prediction rule h,

$$R(h) = \mathbb{E}_X[\mathbb{E}_Y[L(Y, h(X)) \mid X]].$$

As, for any x,

$$\begin{aligned} \mathbb{E}_Y[L(Y, h(X)) \mid X = x] &\geq \min_{y \in \mathcal{Y}} \mathbb{E}_Y[L(Y, y) \mid X = x] \\ &= \mathbb{E}_Y[L(Y, h^*(X)) \mid X = x] \end{aligned}$$

we therefore obtain, for any $h \in \mathcal{F}$, $R(h) \geq R(h^*)$. □

Let $F_x(y) = \Pr(Y \leq y \mid X = x)$ denote the conditional cdf of Y given $X = x$. For each x, the Bayes prediction $h^*(x)$ is a statistical functional of the conditional cdf F_x:

$$h^*(x) = T(F_x) \tag{1.12}$$

with

$$T(F_x) = \arg\min_{y^* \in \mathcal{Y}} \int_{\mathcal{Y}} L(y, y^*) dF_x(y). \tag{1.13}$$

The risk $R(h^*)$ of the Bayes prediction function h^* is called the **Bayes risk**; h^* is optimal in the sense that it achieves the lowest risk amongst all prediction rules. That is, for any function $h \in \mathcal{F}$, we have

$$R(h) \geq R(h^*).$$

The difference

$$R(h) - R(h^*) \geq 0 \tag{1.14}$$

is called the **excess risk** of the prediction rule h.

In the case of the squared loss function and of the $0-1$ loss, the Bayes prediction rule and the Bayes risk both admit a simple expression. For a squared loss function, we obtain (see Problem Sheet)

$$h^*(x) = \mathbb{E}[Y|X = x] \tag{1.15}$$

$$R(h^*) = \mathbb{E}_X[\mathrm{var}(Y \mid X)] \tag{1.16}$$

$$R(h) - R(h^*) = \mathbb{E}_X[(h(X) - h^*(X))^2]. \tag{1.17}$$

In the case of the $0-1$ loss, we obtain for classification problems

$$h^*(x) = \arg\max_{k \in \mathcal{Y}} \Pr(Y = k|X = x) \tag{1.18}$$

$$R(h^*) = 1 - \mathbb{E}_X \left[\max_{k \in \mathcal{Y}} \Pr(Y = k|X) \right]. \tag{1.19}$$

The discriminant functions of the optimal classifier are therefore $f_k(x) = \Pr(Y = k|X = x)$. For binary classification, this can be further simplified to

$$h^*(x) = \begin{cases} 1 & \text{if } \eta(x) \geq 1/2 \\ -1 & \text{otherwise} \end{cases} \tag{1.20}$$

$$R(h^*) = \frac{1}{2} - \mathbb{E}\left[|\eta(X) - 1/2|\right] \tag{1.21}$$

where $\eta(x) = \Pr(Y = 1 \mid X = x)$. The associated discrimination function is therefore $f(x) = \eta(x) - \frac{1}{2}$. Additionally, the excess risk takes the form

$$R(h) - R(h^*) = \mathbb{E}_X[(2\eta(X) - 1)(\mathbb{1}_{h^*(X)=1} - \mathbb{1}_{h(X)=1})] \tag{1.22}$$

1.3 Optimal classifier under Gaussian class distributions

We consider here the optimal classifier under Gaussian class distributions. Assume that the joint distribution of (X, Y), where $X \in \mathbb{R}^p$ and $Y \in \{1, \ldots, K\}$, is defined as follows.

$$\Pr(Y = k) = \pi_k \tag{1.23}$$

$$X \mid Y = k \sim \mathcal{N}(\mu_k, \Sigma_k) \tag{1.24}$$

where $\pi_k \geq 0$ are the class probabilities, with $\sum_{k=1}^K \pi_k = 1$, $\mu_k \in \mathbb{R}^p$ is the mean of observations in class k and Σ_k is the $p \times p$ covariance matrix of observations in class k. Denote

$$g_k(x) = (2\pi)^{-p/2} |\Sigma_k|^{-1/2} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) \right) \tag{1.25}$$

the conditional pdf of X given $Y = k$. An application of Bayes' theorem gives

$$\Pr(Y = k|X = x) = \frac{\pi_k g_k(x)}{\sum_{\ell=1}^K \pi_\ell g_\ell(x)}. \tag{1.26}$$

Note that the denominator in Equation (2.1) does not depend on k. It follows that, under a $0-1$ loss, the optimal classifier is

$$\begin{aligned} h^*(x) &= \underset{k\in\{1,\dots,K\}}{\arg\max}\ \Pr(Y=k|X=x) \\ &= \underset{k\in\{1,\dots,K\}}{\arg\max}\ \pi_k g_k(x) \\ &= \underset{k\in\{1,\dots,K\}}{\arg\max}\ \log(\pi_k)+\log(g_k(x)) \\ &= \underset{k\in\{1,\dots,K\}}{\arg\max}\ f_k(x) \end{aligned}$$

where the discriminant functions f_k, $k=1,\dots,K$, are defined as

$$\begin{aligned} f_k(x) &= \log(\pi_k) - \frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^\intercal\Sigma_k^{-1}(x-\mu_k) \\ &= \log(\pi_k) - \frac{1}{2}(\log|\Sigma_k| + \mu_k^T\Sigma_k^{-1}\mu_k) + \mu_k^\intercal\Sigma_k^{-1}x - \frac{1}{2}x^\intercal\Sigma_k^{-1}x \\ &= a_k + b_k^\intercal x + x^\intercal c_k x \end{aligned}$$

where $a_k = \log(\pi_k) - \frac{1}{2}(\log|\Sigma_k| + \mu_k^T\Sigma_k^{-1}\mu_k)$, $b_k = \Sigma_k^{-1}\mu_k$ and $c_k = -\frac{1}{2}\Sigma_k^{-1}$. The discriminant function is therefore a quadratic function of x. The decision boundary between class k and class k' is obtained by looking at the solutions to the equation $f_k(x) - f_{k'}(x) = 0$. This gives

$$a_k + b_k^\intercal x + x^\intercal c_k x - (a_{k'} + b_{k'}^\intercal x + x^\intercal c_{k'} x) = a_* + b_*^\intercal x + x^\intercal c_* x = 0.$$

where $a_* = a_k - a_{k'}$, $b_* = b_k - b_{k'}$ and $c_* = c_k - c_{k'}$. The decision boundary is therefore given by the roots of a quadratic function of x.

Equal covariance and linear classifier.

If the covariance matrices are all equal, $\Sigma_k = \Sigma$ for all k, the discriminant functions simplify to

$$\begin{aligned} f_k(x) &= \log(\pi_k) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\underbrace{(x-\mu_k)^\intercal\Sigma^{-1}(x-\mu_k)}_{\text{Mahalanobis distance}} \\ &= -\frac{1}{2}(x^T\Sigma^{-1}x + \log|\Sigma|) + \log(\pi_k) - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \mu_k^T\Sigma^{-1}x. \end{aligned}$$

As the first term $-\frac{1}{2}(x^T\Sigma^{-1}x + \log|\Sigma|)$ does not depend on k,

$$\underset{k\in\{1,\dots,K\}}{\arg\max}\ f_k(x) = \underset{k\in\{1,\dots,K\}}{\arg\max}\ \widetilde{f}_k(x)$$

with discriminant functions

$$\begin{aligned} \widetilde{f}_k(x) &= \log(\pi_k) - \frac{1}{2}\mu_k^\intercal\Sigma^{-1}\mu_k + \mu_k^\intercal\Sigma^{-1}x \\ &= a_k + b_k^\intercal x \end{aligned}$$

where $a_k = \log(\pi_k) - \frac{1}{2}(\mu_k^\intercal\Sigma^{-1}\mu_k)$, $b_k = \Sigma^{-1}\mu_k$. The discriminant functions are therefore linear, and the decision boundary between two classes is an hyperplane, solution to the equation

$$a_* + b_*^\intercal x = 0$$

where $a_* = a_k - a_{k'}$ and $b_* = b_k - b_{k'}$. The resulting optimal classifier is therefore a **linear classifier**. Figure 1.1 provides illustrations of the Bayes classifier and decision boundaries when X is two-dimensional, and Y has two classes, when the covariance matrices or different or equal.

Equal covariance and discriminant coordinates.

In the case of equal covariances, the predicted class $h^*(x)$ only depends on x through lower-dimensional projections of x, called its **discriminant coordinates**. In order to gain some intuition, consider first the binary

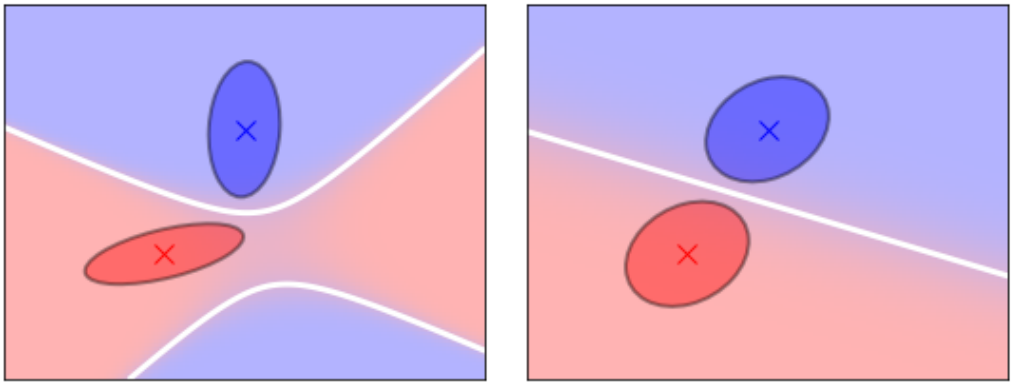


Figure 1.1: Optimal Bayes classifiers (background color) and associated decision boundaries (white line) under Gaussian class distributions, with (left) different covariances within each class and (right) the same covariance. Crosses correspond to the class means, and ellipses represent the contour of constant pdf of the Gaussian class distributions.

case, with a discriminative function $\eta(x) = \widetilde{f}_2(x) - \widetilde{f}_1(x)$

$$\begin{aligned}\eta(x) &= \log(\pi_2) - \log(\pi_1) - \frac{1}{2}\left(\mu_2^\intercal \Sigma^{-1}\mu_2 - \mu_1^\intercal \Sigma^{-1}\mu_1\right) + (\mu_2 - \mu_1)^\intercal \Sigma^{-1} x \\ &= \log\frac{\pi_2}{\pi_1} - \frac{1}{2}(\mu_2 - \mu_1)^\intercal \Sigma^{-1}(\mu_2 + \mu_1) + (\mu_2 - \mu_1)^\intercal \Sigma^{-1} x\end{aligned}$$

Letting

$$x_0 = \frac{1}{2}(\mu_1 + \mu_2) - (\mu_2 - \mu_1)\frac{\log(\pi_2/\pi_1)}{(\mu_2 - \mu_1)^\intercal \Sigma^{-1}(\mu_2 - \mu_1)}$$

we obtain

$$\eta(x) = (\mu_2 - \mu_1)^\intercal \Sigma^{-1}(x - x_0).$$

Note that x_0 lies on the line passing through the class means μ_1 and μ_2, and $x_0 = \frac{1}{2}(\mu_1 + \mu_2)$ is half-way if $\pi_2 = \pi_1 = 1/2$.

For a vector $x \in \mathbb{R}^p$, writing $x^\bullet = \Sigma^{-1/2}x$, we obtain

$$\eta(x) = (\mu_2^\bullet - \mu_1^\bullet)^\intercal (x^\bullet - x_0^\bullet).$$

The decision boundary therefore only depends on x through the projection of $x^\bullet = \Sigma^{-1/2}x$ onto the vector of the difference of the transformed class means $(\mu_2^\bullet - \mu_1^\bullet)$. Hence, for the purpose of classification, only the one-dimensional projection $(\mu_2^\bullet - \mu_1^\bullet)\Sigma^{-1/2}x$ of the p-dimensional vector x is relevant.

For general K, the classifier relies on the Mahalanobis distances

$$(x - \mu_k)^\intercal \Sigma^{-1}(x - \mu_k)$$

between the vector x to the class means or, equivalently, on the squared Euclidian distances

$$(x^\bullet - \mu_k^\bullet)^\intercal (x^\bullet - \mu_k^\bullet)$$

between the transformed $x^\bullet$ and $\mu_k^\bullet$. The K sphered class means $\mu_k^\bullet$ lie in an affine manifold of dimension lower or equal to $K-1$. When looking at the closest class mean, one can ignore distances orthogonal to this subspace, as they contribute equally for each class. We can therefore project the $x^\bullet$ onto this class-spanning subspace H_{K-1}. This corresponds to finding the principal components subspace of the transformed means, or similarly, to find the eigendecomposition of the sphered between-class covariance

$$B^\bullet = \operatorname{cov}(\mathbb{E}[\Sigma^{-1/2}X|Y]) = \Sigma^{-1/2}\operatorname{cov}(\mathbb{E}[X|Y])\Sigma^{-1/2} = \sum_{k=1}^{K}\pi_k(\mu_k^\bullet - \mu^\bullet)(\mu_k^\bullet - \mu^\bullet)^\intercal$$

where $\mu^\bullet = \sum_{k=1}^{K}\pi_k\mu_k^\bullet$. Denote

$$B^\bullet = U^\bullet D^\bullet U^{\bullet\intercal}$$

the eigendecomposition of $B^\bullet$. The columns $u_\ell^\bullet$ of $U^\bullet$ in sequence define the coordinates of the optimal subspaces. We therefore define

$$z_\ell = (u_\ell^\bullet)^\intercal x^\bullet = (u_\ell^\bullet)^\intercal \Sigma^{-1/2} x$$

the ℓth **discriminant coordinate** of the vector x.

Fisher arrived at the same decomposition with a different route, without referring to Gaussian distributions. Assume we want to find a linear combination $Z = a^\intercal X$ such that the between-class variance is maximised relative to the within-class variance. That is, we want to maximise

$$\frac{\text{var}(\mathbb{E}[a^\intercal X|Y])}{\mathbb{E}[\text{var}(a^\intercal X|Y)]} = \frac{a^\intercal B a}{a^\intercal \Sigma a}.$$

Setting $u = \Sigma^{1/2} a$ yields

$$\frac{u^\intercal B^\bullet u}{u^\intercal u}.$$

Maximisation over u is achieved by the first eigenvector of $B^\bullet$, which corresponds to $u_1^\bullet$ as above. The next eigenvector $u_2^\bullet$ is obtained by finding the vector orthogonal to $u_1^\bullet$ that maximises $\frac{u_2^\intercal B^\bullet u_2}{u_2^\intercal u_2}$, etc.

1.4 Statistical Learning Approaches

For a prediction rule h and a dataset $d = (x_i, y_i)_{i=1,\dots,n}$, the **empirical risk**, or **training error**, is defined as

$$\widehat{R}_n(h) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i)). \tag{1.27}$$

This represents the average error over the dataset d used to train/learn the classifier. The population risk $R(h)$, defined in Definition 7, is also called the **generalisation error** or **out-of-sample** error. It represents the average error for a new observation out of the training set d, and is the quantity we aim to minimise. The **generalisation gap** is defined as the difference between the generalisation error and the training error

$$R(h) - \widehat{R}_n(h).$$

Our objective is to find a prediction rule h with small generalisation error $R(h)$. The conditional distribution of $Y|X = x$ being unknown, we cannot calculate the optimal prediction rule h^*. For a given function h, we cannot either compute its generalisation error $R(h)$ as this involves an expectation with respect to the true unknown distribution of (X, Y). All we have is a realisation $d = (x_i, y_i)_{i=1,\dots,n}$ from a random sample $D = (X_i, Y_i)_{i=1,\dots,n}$.

Considering either the representation (1.10) or (1.11) for the optimal prediction rule, there are two broad families of methods that we might consider.

1. **Empirical Risk Minimisation** (ERM): Replace the population risk $R(h)$ by the empirical risk $\widehat{R}_n(h)$, and minimise this function over some set of functions $\mathcal{H} \subset \mathcal{F}$

$$\widehat{h}^{(d)} = \arg\min_{h \in \mathcal{H}} \widehat{R}_n(h). \tag{1.28}$$

2. **Plug-in methods**: Obtain some estimate $\widehat{F}_x$ of the conditional distribution of Y given $X = x$, and set

$$\widehat{h}^{(d)}(x) = T(\widehat{F}_x) = \arg\min_{y^* \in \mathcal{Y}} \int_{\mathcal{Y}} L(y, y^*) d\widehat{F}_x(y). \tag{1.29}$$

The two different approaches address the prediction problem by considering tasks of increasing difficulty: ERM learns directly the map from $\mathcal{X}$ to $\mathcal{Y}$, without trying to estimate the conditional distribution of $Y|X = x$. The conditional approach aims at estimating the conditional distribution of $Y|X = x$ in order to derive the prediction rule. Plug-in methods can themselves be separated between the conditional approach and the generative approach:

2a. **The conditional plug-in** method estimates directly the conditional distribution of Y given $X = x$

2b. **The generative plug-in** method first estimates the joint distribution of (X, Y), then derives an estimate of the conditional distribution of Y given $X = x$ via Bayes' theorem.

While the conditional approach leaves the marginal distribution of X unspecified, the generative approach aims at estimating the joint distribution, and hence tackles a more difficult problem.

The ERM and conditional plug-in approaches are commonly referred as **discriminative learning**, as one tries to directly learn the relation from X to Y. The generative plug-in approach is known as **generative learning**, as one tries to learn the full joint distribution of (X, Y), hence tries to learn the full generative model of the data.

1.4.1 Empirical Risk Minimisation: Definition

As the true data distribution of the data is unknown (as well as the conditional distribution of $Y|X = x$), neither the true risk nor the optimal prediction function can be computed. However, we have a training set of data $(x_i, y_i)_{i=1,\dots,n}$ which are realisations from the true data distribution P_0. As a proxy for the true risk, we use the empirical risk, which can be seen as a Monte Carlo estimate of the true risk $R(h)$ as $(x_i, y_i)_{i=1,\dots,n}$ are realisations of an iid random sample $(X_i, Y_i)_{i=1,\dots,n}$ from P_0. As such, it is an unbiased and consistent estimator of the true risk:[1] For any fixed h, $\mathbb{E}[\widehat{R}_n(h)] = R(h)$ and $\widehat{R}_n(h) \to R(h)$ almost surely as $n \to \infty$ (See Problem Sheet).

We cannot minimise the empirical risk over the whole set of functions $\mathcal{F}$. If all the x_i take different values, the minimal empirical risk $\widehat{R}_n(h) = 0$ is achieved for an infinite number of prediction rules:

$$\widehat{h}^{(d)}(x) = \begin{cases} y_i & \text{if } x \in \{x_1, \dots, x_n\} \\ h_0(x) & \text{otherwise} \end{cases}$$

where $h_0 : \mathcal{X} \to \mathcal{Y}$ is an arbitrary function. We therefore restrict the set of prediction functions to a specified function space $\mathcal{H} \subset \mathcal{F}$ (e.g. the set of linear classifiers).

Definition 10. *For a function space $\mathcal{H}$, a loss function L, and a training set $d = (x_i, y_i)_{i=1,\dots,n}$, the Empirical Risk Minimiser $\widehat{h}^{(d)}$ is defined as*

$$\widehat{h}^{(d)} = \arg\min_{h \in \mathcal{H}} \widehat{R}_n(h) \tag{1.30}$$

where $\widehat{R}_n(h)$ is the empirical risk defined in Equation (1.27)*.*

In practice, a function $h \in \mathcal{H}$ will often be parameterise by a vector $\theta \in \Theta$, and we write h_θ. For instance, if $\mathcal{H}$ is the class of linear functions on $\mathbb{R}$, then $h_\theta(x) = \beta_0 + \beta_1 x$ where $\theta = (\beta_0, \beta_1) \in \mathbb{R}^2$. The ERM is therefore given by $\widehat{h}^{(d)} = h_{\widehat{\theta}}$ where

$$\widehat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(y_i, h_\theta(x_i)).$$

1.4.2 Plug-in methods

Plug-in methods obtain some estimate $\widehat{F}_x$ of the conditional distribution of Y given $X = x$, and set

$$\widehat{h}^{(d)}(x) = T(\widehat{F}_x) = \arg\min_{y^* \in \mathcal{Y}} \int_{\mathcal{Y}} L(y, y^*) d\widehat{F}_x(y). \tag{1.31}$$

We will focus here on the case where the estimate is obtaining via maximum likelihood. Assume that the conditional probability mass/density function of Y given $X = x$ is $f_\theta(y|x)$ where the parametric form f_θ is known, but $\theta \in \Theta$ is some unknown parameter. In the generative case, $f_\theta(y|x) \propto \pi_\theta(x, y)$, where π_θ is the known parametric form of the joint distribution. The maximum likelihood estimate of θ based on the training dataset d is

$$\widehat{\theta} = \arg\max_{\theta \in \Theta} \sum_{i=1}^{n} \log(f_\theta(y_i|x_i)) \quad \text{[Conditional Approach]}$$

$$\widehat{\theta} = \arg\max_{\theta \in \Theta} \sum_{i=1}^{n} \log(\pi_\theta(x_i, y_i)) \quad \text{[Generative Approach]}$$

The associated estimate of the pmf/pdf is therefore $f_{\widehat{\theta}}$, leading to the plug-in estimators

$$\widehat{h}^{(d)}(x) = \int_{\mathbb{R}} y f_{\widehat{\theta}}(y|x) dy \text{ for regression}$$

$$\widehat{h}^{(d)}(x) = \arg\max_{k=1,\dots,K} f_{\widehat{\theta}}(k|x) \text{ for classification}$$

[1] We use the same notation $\widehat{R}_n(h)$ for the random variable $\frac{1}{n}\sum_{i=1}^n L(Y_i, h(X_i))$ and its realisation $\frac{1}{n}\sum_{i=1}^n L(y_i, h(x_i))$.

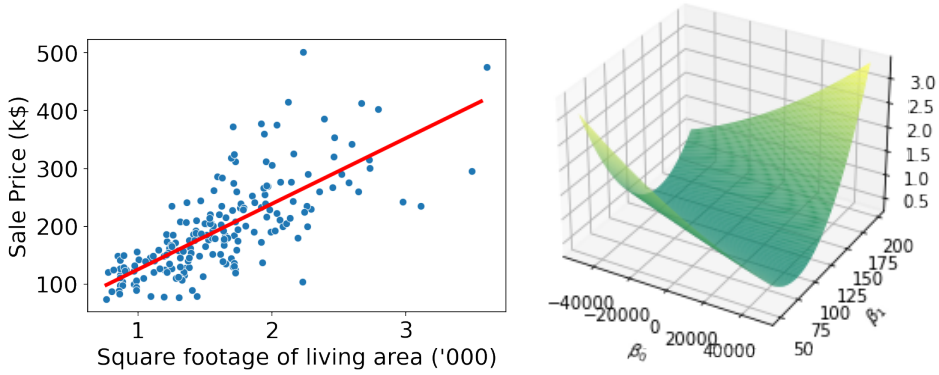


Figure 1.2: (Left) House price (dollars) versus living area (square feet) (blue dots) and estimated prediction rule $\widehat{h}^{(d)}(x)$ under a linear model (red). (Right) Empirical Risk as a function of the model parameters β_0 and β_1.

1.4.3 Example: ordinary linear regression

As an illustration we will consider the familiar example of univariate linear regression.

We consider a dataset consisting of house sale prices together with various attributes (square footage, number of rooms, etc)[2]. The objective is to predict the sale price of an house based on its attributes. We will only consider here predicting the sale price $y \in \mathbb{R}$ based on the square footage $x \in \mathbb{R}$ of the living area. We take a training set of $n = 200$ observations (x_i, y_i), $i = 1, \ldots, n$. The data are represented in Figure 1.2.

Empirical Risk minimisation. Consider the set of linear prediction rules

$$\mathcal{H}_1 = \{h : \mathcal{X} \to \mathcal{Y} \mid h(x) = \beta_0 + \beta_1 x, \text{ with } \beta_0 \in \mathbb{R}, \beta_1 \in \mathbb{R}\}.$$

For a prediction rule $h \in \mathcal{H}_1$, with $h(x) = \beta_0 + \beta_1 x$, the empirical risk under the squared error loss is given by

$$\widehat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x)^2.$$

The empirical risk is plotted in Figure 1.2 (right) as a function of the two parameters β_0 and β_1.

The ERM under the squared error loss, in the hypothesis set $\mathcal{H}_1$, is therefore given by

$$\widehat{h}^{(d)}(x) = \widehat{\beta}_0 + \widehat{\beta}_1 x \tag{1.32}$$

where

$$(\widehat{\beta}_0, \widehat{\beta}_1) = \underset{(\beta_0,\beta_1)\in\mathbb{R}^2}{\arg\min} \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x)^2$$

are the ordinary least squares estimates, whose expression is given by Equation (1.6).

Plug-in approach. In this particular case, the estimated prediction rule (1.32) may also be interpreted as a plug-in estimator, under additional assumptions. Assume now that the conditional distribution of Y given $X = x$ is normal with mean $\beta_0 + \beta_1 x$ and variance σ^2, where σ^2 is assumed fixed. The conditional pdf is therefore given by

$$f_\theta(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\beta_0-\beta_1 x)^2}{2\sigma^2}}$$

where $\theta = (\beta_0, \beta_1) \in \mathbb{R}^2$ are unknown parameters. The log-likelihood takes the form

$$\ell(\beta_0, \beta_1) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2.$$

[2] https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

The maximum likelihood estimator is given by

$$\begin{aligned}(\widehat{\beta}_0, \widehat{\beta}_1) &= \arg\max_{(\beta_0,\beta_1)} -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2 \\ &= \arg\min_{(\beta_0,\beta_1)} \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2\end{aligned}$$

which is the ordinary least square estimator. The learned prediction rule is therefore

$$\widehat{h}^{(d)}(x) = \int_{\mathbb{R}} y f_{(\widehat{\beta}_0,\widehat{\beta}_1)}(y|x)dy = \widehat{\beta}_0 + \widehat{\beta}_1 x$$

which is the same as the ERM.

2 — Generative classifiers

This chapter is concerned with generative approaches for multiclass classification. We will present three generative classifiers: Linear Discriminant Analysis, Quadratic Discriminant Analysis and Naive Bayes. We start by a general definition of a generative classifier.

2.1 General definition

As mentioned in the previous chapter, generative classifiers attempt to obtain some estimate of the joint distribution of the pair $(X, Y) \in \mathcal{X} \times \{1, \dots, K\}$. Let

- $\pi_k = \Pr(Y = k)$ be the class probability and
- $g_k(x)$ be the conditional pdf or pmf of X, given $Y = k$.

An application of Bayes' theorem gives

$$\Pr(Y = k | X = x) = \frac{\pi_k g_k(x)}{\sum_{\ell=1}^{K} \pi_\ell g_\ell(x)}. \tag{2.1}$$

Under a $0-1$ loss, the optimal classifier is

$$\begin{aligned} h^*(x) &= \underset{k \in \{1,\dots,K\}}{\arg\max} \Pr(Y = k | X = x) \\ &= \underset{k \in \{1,\dots,K\}}{\arg\max} \pi_k g_k(x) \end{aligned}$$

as the denominator in Equation (2.1) does not depend on k.

Definition 11 (Generative classifier)**.** *For $k = 1, \dots, K$ and $x \in \mathcal{X}$, let $\widehat{\pi}_k$ and $\widehat{g}_k(x)$ be some estimates, using the training set d, of the class probabilities and of the conditional densities. The generative classifier is then given by*

$$\begin{aligned} \widehat{h}^{(d)}(x) &= \underset{k \in \{1,\dots,K\}}{\arg\max} \widehat{\pi}_k \widehat{g}_k(x) \\ &= \underset{k \in \{1,\dots,K\}}{\arg\max} \log(\widehat{\pi}_k) + \log(\widehat{g}_k(x)) \end{aligned}$$

How do we obtain these estimates? For the class probabilities, it is rather straightforward. The class variables $Y_1, \dots, Y_n$ are iid from a discrete distribution on $\{1, \dots, K\}$ with probability mass function $(\pi_1, \dots, \pi_K)$. We can estimate π_k by maximum likelihood. The log-likelihood is given by

$$\ell(\pi_1, \dots, \pi_K) = \sum_{k=1}^{K} m_k \log(\pi_k) \tag{2.2}$$

where $m_k = \#\{j : y_j = k\}$ is the number of observations in the class k. We aim at maximising the log-likelihood in (2.2) under the constraints $\pi_k \geq 0$ for all $k \geq 0$ and $\sum_{k'} \pi_{k'} = 1$. The Lagrangian is

$$\mathcal{L}(\pi_1, \dots, \pi_K, \gamma) = \sum_{k=1}^{K} m_k \log(\pi_k) - \gamma \left(\sum_{k=1}^{K} \pi_k - 1 \right).$$

Differentiating $\mathcal{L}$ with respect to π_k and setting to 0 gives $\pi_k = m_k / \gamma$ for all k. Combining this with the constraint $\sum_{k=1}^{K} \pi_k = 1$, we obtain the maximum likelihood estimate

$$\widehat{\pi}_k = \frac{m_k}{n}.$$

We now describe three different methods for obtaining estimates of $g_k(x)$, leading to different classifiers.

2.2 Linear Discriminant Analysis

Let $\mathcal{X} = \mathbb{R}^p$. Linear discriminant analysis (LDA) assumes the class conditional densities $g_k(x)$ are normal pdfs, with a **shared** covariance matrix

$$g_k(x) = \varphi(x; \mu_k, \Sigma)$$

where μ_k, $k = 1, \ldots, p$ is a vector of size p, called the centroid of the class, and Σ is a $p \times p$ covariance matrix. Note that under this assumption, Σ is the within-class covariance matrix of the random vector X.

The parameters μ_k and Σ can be estimated via maximum likelihood, as for the class frequencies π_k. Assuming that the (x_i, y_i) are iid realisations of a random variable (X, Y) with $\Pr(Y = k) = \pi_k$ and conditional class densities $g_k(x)$ given by 2.3, the log-likelihood takes the form

$$\ell(\pi_1, \ldots, \pi_K, \mu_1, \ldots, \mu_K, \Sigma) = \sum_{i=1}^{n} \left(\log(\pi_{y_i}) + \log \varphi(x_i; \mu_{y_i}, \Sigma)\right) \tag{2.3}$$

$$= \sum_{k=1}^{K} \left(m_k \log(\pi_k) + \sum_{i|y_i=k} \log \varphi(x_i; \mu_k, \Sigma) \right) \tag{2.4}$$

$$= \left(\sum_{k=1}^{K} m_k \log(\pi_k) \right) - \frac{1}{2} \sum_{k=1}^{K} \sum_{i|y_i=k} (x_i - \mu_k)^\intercal \Sigma^{-1} (x_i - \mu_k) - \frac{n}{2} \log|\Sigma| + \text{const} \tag{2.5}$$

Maximising the log-likelihood under the constraint $\sum_{k=1}^{K} \pi_k = 1$ gives the maximum likelihood estimates

$$\widehat{\pi}_k = \frac{m_k}{n} \tag{2.6}$$

$$\widehat{\mu}_k = \frac{1}{m_k} \sum_{i:y_i=k} x_i \tag{2.7}$$

$$\widehat{\Sigma} = \frac{1}{n} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \widehat{\mu}_k)(x_i - \widehat{\mu}_k)^\intercal. \tag{2.8}$$

Proof. The MLE for π has already been derived. For μ_k, the partial derivative is

$$\frac{\partial \ell}{\partial \mu_k} = \sum_{i|y_i=k} \Sigma^{-1}(x_i - \mu_k)$$

Setting this to 0 yields the MLE (2.7). Differentiating with respect to Σ (see Section 0.2), we obtain

$$\frac{\partial \ell}{\partial \Sigma} = \frac{1}{2} \sum_{k=1}^{K} \sum_{i|y_i=k} \Sigma^{-1}(x_i - \mu_k)(x_i - \mu_k)^\intercal \Sigma^{-1} - \frac{n}{2} \Sigma^{-1}.$$

Setting this to 0, then left and right-multiplying by Σ gives Equation (2.8). □

One then takes $\widehat{\pi}_k \varphi(x; \widehat{\mu}_k, \widehat{\Sigma})$ as an approximation to $\pi_k g_k(x)$. As the class densities are normal, this leads to linear discriminant functions and linear decision boundaries, as described in Section 1.3:

$$\widehat{h}^{(d)}(x) = \operatorname*{arg\,max}_{k=1,\ldots,K} \; \log(\widehat{\pi}_k) - \frac{1}{2} \underbrace{(x - \widehat{\mu}_k)^\intercal \widehat{\Sigma}^{-1} (x - \widehat{\mu}_k)}_{\text{squared Mahalanobis distance}}$$

$$= \operatorname*{arg\,max}_{k=1,\ldots,K} \; \underbrace{\log(\widehat{\pi}_k) - \frac{1}{2} \widehat{\mu}_k^\intercal \widehat{\Sigma}^{-1} \widehat{\mu}_k}_{\widehat{a}_k} + \underbrace{\widehat{\mu}_k^\intercal \widehat{\Sigma}^{-1}}_{\widehat{b}_k^\intercal} x.$$

The scalar $\widehat{a}_k$ is the intercept of the linear prediction rule for class k, and the p-dimensional vector $\widehat{b}_k$ its coefficients (called slope if $p = 1$).

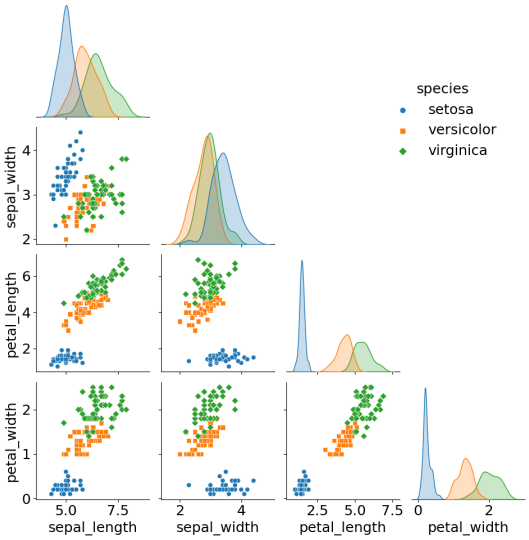


Figure 2.1: Pairplot of the Iris dataset.

Computations for LDA. Assuming $\widehat{\Sigma}$ has full rank, let $\widehat{\Sigma} = V\Lambda V^\intercal$ be the eigenvalue decomposition of the matrix $\widehat{\Sigma}$, where V is a $p \times p$ orthogonal matrix and Λ is a diagonal $p \times p$ matrix. Note that $\widehat{\Sigma}^{-1} = V\Lambda^{-1}V^\intercal$ and $\widehat{\Sigma}^{-1/2} = \Lambda^{-1/2}V^\intercal$. We have

$$(x - \widehat{\mu}_k)^\intercal \widehat{\Sigma}^{-1}(x - \widehat{\mu}_k) = (\Lambda^{-1/2}V^\intercal x - \Lambda^{-1/2}V^\intercal \widehat{\mu}_k)^\intercal (\Lambda^{-1/2}V^\intercal x - \Lambda^{-1/2}V^\intercal \widehat{\mu}_k) \tag{2.9}$$

$$= ||x^\bullet - \widehat{\mu}_k^\bullet||^2 \tag{2.10}$$

that is the squared Euclidian distance between the transformed vectors $x^\bullet = \Lambda^{-1/2}V^\intercal x$ and $\widehat{\mu}_k^\bullet = \Lambda^{-1/2}V^\intercal \widehat{\mu}_k$. Based on the estimated V, Λ and $\widehat{\mu}_k^\bullet$, LDA can therefore be implemented via the following steps.

- Training
  - Compute the MLE $\widehat{\pi}_k$, $\widehat{\mu}_k$ and $\widehat{\Sigma}$.
  - Compute the eigendecomposition V, Λ of $\widehat{\Sigma}$
  - For $k = 1, \ldots, K$, set $\widehat{\mu}_k^\bullet = \Lambda^{-1/2}V^\intercal \widehat{\mu}_k$.
- Prediction. For an input vector x
  - Calculate $x^\bullet = \Lambda^{-1/2}V^\intercal x$
  - Classify to the closest class mean $\widehat{\mu}_k^\bullet$, modulo the effect of the estimated class proportions $\widehat{\pi}_k$

$$\widehat{h}^{(d)}(x) = \arg\min_{k=1,\ldots,K} ||x^\bullet - \widehat{\mu}_k^\bullet||^2 - 2\log(\widehat{\pi}_k)$$

.

Example 12 (Iris dataset (2D))**.** *We consider here the classic Iris dataset. The original dataset contains $p = 4$ measurements (sepal length, sepal width, petal length, petal width) for $n = 50$ Iris plants. Each plant is from one of $K = 3$ types: Iris Setosa, Iris Versicolour and Iris Virginica. A pairplot of the data is represented in Figure 12.*

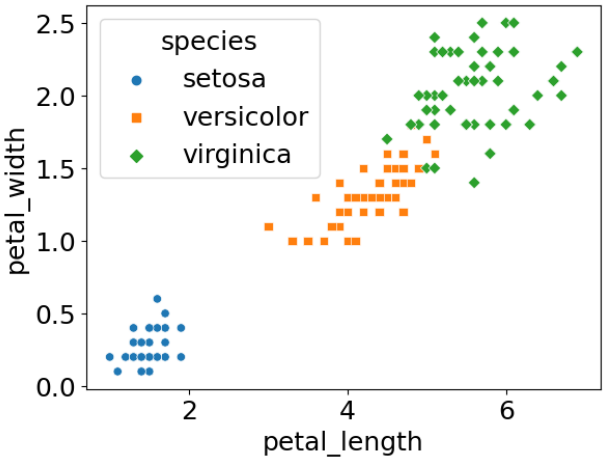
For visualisation purposes, we will first consider two dimensions, the petal width and length. Figure 2.2 shows the scatterplot of the data, together with the LDA learned prediction rule on the two-dimensional data.

LDA for dimensionality reduction. Let $\widehat{\mu} = \frac{1}{n}\sum_{i=1}^n x_i$ be the sample mean and $\widehat{B} = \sum_{k=1}^K \widehat{\pi}_k(\widehat{\mu}_k - \widehat{\mu})(\widehat{\mu}_k - \widehat{\mu})^\intercal$ be an estimate of the between class covariance. Define

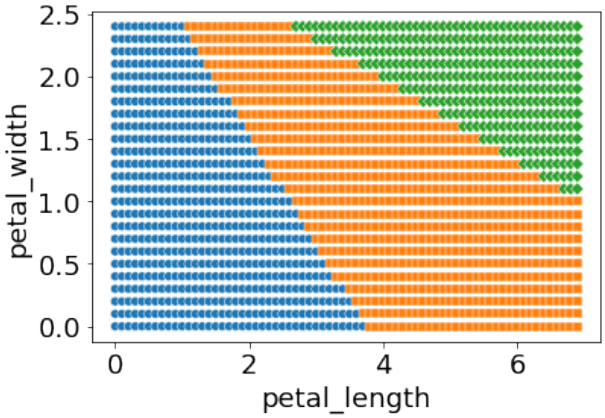
$$\widehat{B}^\bullet = \sum_{k=1}^K \widehat{\pi}_k(\widehat{\mu}_k^\bullet - \widehat{\mu}^\bullet)(\widehat{\mu}_k^\bullet - \widehat{\mu}^\bullet)^\intercal$$

where $\widehat{\mu}^\bullet = \sum_{k=1}^K \pi_k \mu_k^\bullet$. Denote

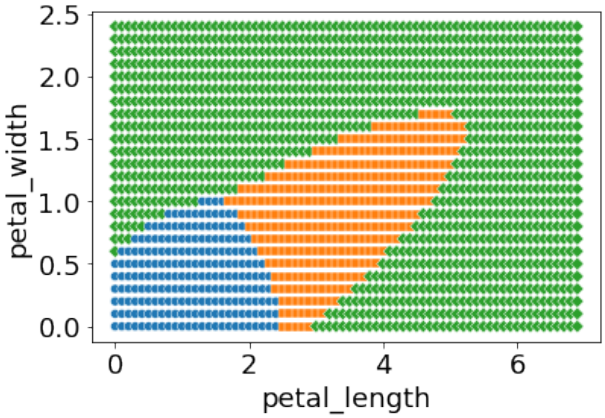
$$\widehat{B}^\bullet = U^\bullet D^\bullet U^{\bullet\intercal}$$



(a) Iris data (2D) - training dataset



(b) Iris data (2D) - LDA prediction rule



(c) Iris data (2D) - QDA prediction rule

Figure 2.2: (a) Pair Scatterplot of the iris dataset in two dimensions, (b) LDA and (c) QDA prediction rules

the eigendecomposition of $\widehat{B}^\bullet$, and $u_\ell^\bullet$ the columns of $U^\bullet$. The $\ell's$ **discriminant coordinate** of a vector x is defined as

$$z_\ell = (u_\ell^\bullet)^\intercal x^\bullet = (u_\ell^\bullet)^\intercal \Lambda^{-1/2} V^\intercal x.$$

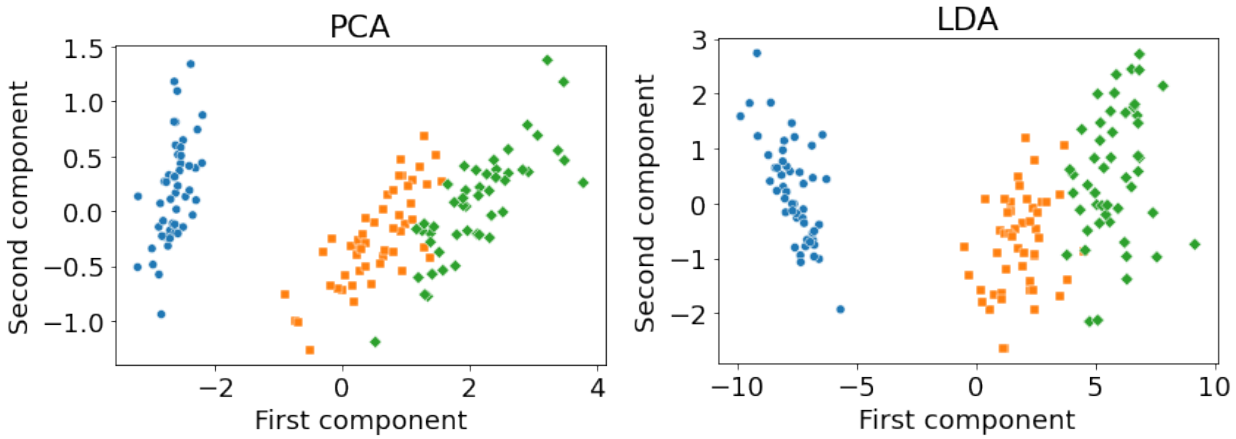
As mentioned in Section 1.3, the prediction $\widehat{h}^{(d)}(x)$ only depends on x through its projections onto the first $K-1$ discriminant coordinates. The projections maximise the separation between the classes. In particular, $a_1 = V\Lambda^{-1/2}u_1^\bullet$ maximises

$$\frac{a_1^\intercal \widehat{B} a_1}{a_1^\intercal \widehat{\Sigma} a_1}$$

and $a_1^\intercal x$ is the one-dimensional projection of the data maximising the separation between classes.

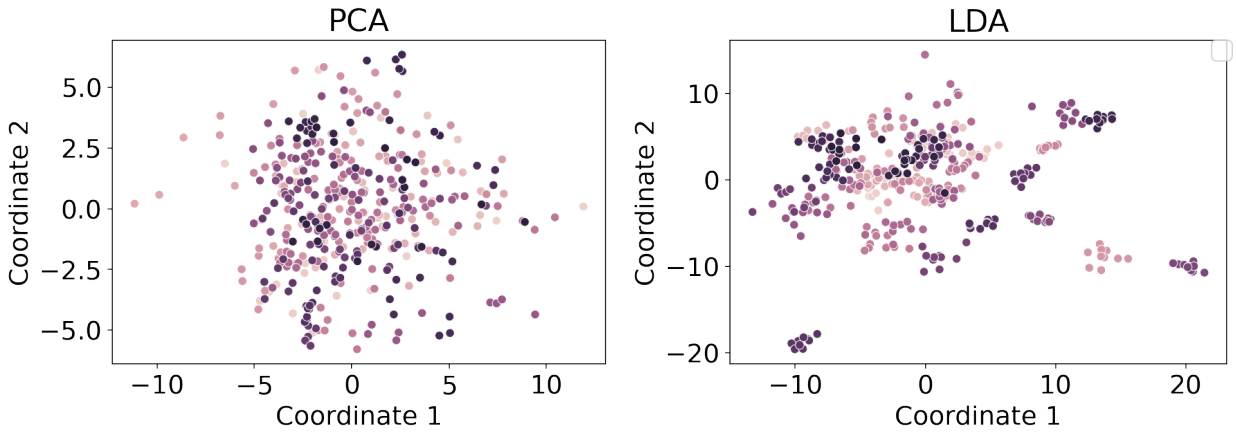
Example 13 (Iris dataset (4D))**.** *We consider again the Iris dataset, with all $p = 4$ features. As $K = 3$, the number of discriminant coordinates is $K - 1 = 2$, and the we can represent the projections of the data on the plane. The projection of the Iris data on the two LDA components is represented in Figure 2.3. Note that the first discriminant coordinate already provide a good separation between the classes. For comparison, the projections onto the first two PCA components are also given. PCA maximises the variance, while LDA maximises the separation between the classes, leading to a better separation of the classes.*

Remark on LDA and SVD removed (not examinable).



(a) PCA (b) LDA

Figure 2.3: Projection of the Iris data ($n = 150, p = 4$) on the first two (a) principal components of PCA and (b) discriminant coordinates of LDA.



(a) PCA (b) LDA

Figure 2.4: Projection of the Olivetti data ($n = 400, p = 4096$) on the first two (a) principal components of PCA and (b) discriminant coordinates of LDA. The different colors represent different individuals ($K = 40$)

Example 14 (Olivetti face dataset)**.** *We revisit the Olivetti dataset, that was analysed using Principal Component Analysis. The dataset contains* $n = 400$ *images from* $K = 40$ *different persons. Each image is* 64×64 *pixel in grey levels, hence* $p = 4096$*. Note that* $p > n$ *here. The first two discriminant coordinates of the training data are represented in Figure 2.4.*

2.3 Quadratic Discriminant Analysis

Quadratic discriminant analysis (LDA) assumes the class conditional densities $g_k(x)$ are normal pdfs, with unconstrained covariance matrices

$$g_k(x) = \varphi(x; \mu_k, \Sigma_k)$$

where μ_k, $k = 1, \ldots, p$ is a vector of size p, called the centroid of the class, and Σ_k is a $p \times p$ covariance matrix of the class k.

The parameters μ_k and Σ_k can be estimated via maximum likelihood, as for the class frequencies π_k. Assuming that the (x_i, y_i) are iid realisations of a random variable (X, Y) with $\Pr(Y = k) = \pi_k$ and conditional

class densities $g_k(x)$ given by 2.11, the log-likelihood takes the form

$$\ell(\pi_1, \ldots, \pi_K, \mu_1, \ldots, \mu_K, \Sigma_1, \ldots, \Sigma_K) \tag{2.11}$$

$$= \sum_{i=1}^{n} (\log(\pi_{y_i}) + \log \varphi(x_i; \mu_{y_i}, \Sigma_{y_i})) \tag{2.12}$$

$$= \sum_{k=1}^{K} \left(m_k \log(\pi_k) + \sum_{i|y_i=k} \log \varphi(x_i; \mu_k, \Sigma_k) \right) \tag{2.13}$$

$$= \left(\sum_{k=1}^{K} m_k \log(\pi_k) \right) - \frac{1}{2} \sum_{k=1}^{K} \sum_{i|y_i=k} (x_i - \mu_k)^\intercal \Sigma_k^{-1} (x_i - \mu_k) - \sum_{k=1}^{K} \frac{m_k}{2} \log |\Sigma_k| + \text{const} \tag{2.14}$$

Maximising the log-likelihood under the constraint $\sum_{k=1}^{K} \pi_k = 1$ gives the maximum likelihood estimates

$$\widehat{\pi}_k = \frac{m_k}{n} \tag{2.15}$$

$$\widehat{\mu}_k = \frac{1}{m_k} \sum_{i:y_i=k} x_i \tag{2.16}$$

$$\widehat{\Sigma}_k = \frac{1}{m_k} \sum_{i:y_i=k} (x_i - \widehat{\mu}_k)(x_i - \widehat{\mu}_k)^T. \tag{2.17}$$

One then takes $\widehat{\pi}_k \varphi(x; \widehat{\mu}_k, \widehat{\Sigma}_k)$ as an approximation to $\pi_k g_k(x)$. This leads to quadratic discriminant functions and quadratic decision boundaries, as described in Section 1.3:

$$\widehat{h}^{(d)}(x) = \underset{k=1,\ldots,K}{\arg\max} \; \log(\widehat{\pi}_k) - \frac{1}{2} \log |\widehat{\Sigma}_k| - \frac{1}{2} \underbrace{(x - \widehat{\mu}_k)^\intercal \widehat{\Sigma}_k^{-1} (x - \widehat{\mu}_k)}_{\text{squared Mahalanobis distance}}$$

$$= \underset{k=1,\ldots,K}{\arg\max} \; \log(\widehat{\pi}_k) - \frac{1}{2} \log |\widehat{\Sigma}_k| - \frac{1}{2} \widehat{\mu}_k^\intercal \widehat{\Sigma}_k^{-1} \widehat{\mu}_k + \widehat{\mu}_k^\intercal \widehat{\Sigma}_k^{-1} x - \frac{1}{2} x^\intercal \widehat{\Sigma}_k^{-1} x.$$

The QDA classifier on the 2D iris dataset is shown in Figure 2.2(c).

2.4 Regularised discriminant analysis

QDA allows for more flexibility and can capture more complex distributions as it allows for different covariance matrices. There is however a price to pay in terms of increased variance and potential overfitting. The number of parameters to estimate is

- $(K-1) + Kp + p(p+1)/2$ for LDA,
- $(K-1) + Kp + Kp(p+1)/2$ for QDA.

The number of parameters therefore scales quadratically with p. When p is large, this may lead to overfitting, in particular for QDA if some classes have few examples.

Regularised Discriminant Analysis. Regularised discriminant analysis combines LDA and QDA by considering the following estimate for the covariance within class k

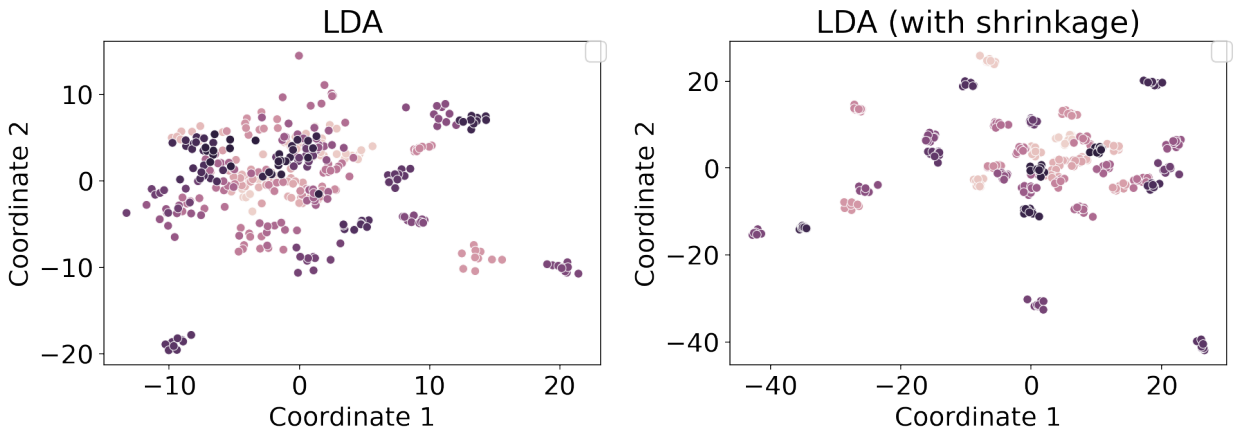
$$\widehat{\Sigma}_k(\alpha) = \alpha \widehat{\Sigma}_k + (1 - \alpha) \widehat{\Sigma}$$

where $\widehat{\Sigma}$ is the MLE for LDA, and $\widehat{\Sigma}_k$ is the MLE for QDA, for some $\alpha \in [0, 1]$. This introduces a new parameter α and allows for a continuum of models between LDA and QDA to be used. The parameter α can be chosen by cross-validation for example.

LDA with shrinkage. An alternative approach is to consider the shrinkage estimate (here for LDA)

$$\widehat{\Sigma}(\delta) = \delta I_p + (1 - \delta) \widehat{\Sigma} \tag{2.18}$$

where $\delta \in (0, 1)$ is a regularising parameter. Note that $\widehat{\Sigma}(\delta)$ is always invertible, even if $\widehat{\Sigma}$ is not.



(a) LDA

(b) LDA with shrinkage

Figure 2.5: Olivetti data ($n = 400, p = 4096$). First two discriminant coordinates for (left) LDA and (right) LDA with shrinkage. The different colors represent different individuals ($K = 40$).

Example 15 (Olivetti face dataset, continued)**.** *We consider again the Olivetti dataset, and look at the discriminant coordinates when using LDA, and LDA with shrinkage estimate* (2.18) *where the shrinkage parameter δ is chosen automatically. The results are shown in Figure* (2.5)*. Using shrinkage provides much improved separation between the classes for the discriminant coordinates. Note that for the LDA, coordinates are obtained using the SVD, while for LDA with shrinkage it is using the eigendecomposition, which is much slower due to the large value p.*

2.5 Naive Bayes classifier

The naive Bayes classifier is another generative plug-in classifier. It makes the assumption that the different univariate components of p-dimensional input variable X are conditionally independent given the class Y. Naive Bayes can easily handle inputs x of mixed type (categorical, binary, continuous), as well as missing data.

To illustrate the method, consider the following example. Imagine we want to predict the voter preferences for the US election based on some attributes. Consider the data

Voted in 2016?	Annual Income	State	Candidate Choice
Y	50K	OK	Biden
N	173K	CA	Biden
Y	80K	NJ	Trump
Y	150K	WA	Biden
Y	85K	IL	Trump
⋮	⋮	⋮	⋮
Y	1050K	NY	Biden
N	35K	CA	Trump
N	**100K**	**NY**	**?**

The input vector x is 3-dimensional and contains a mix of binary (Voted in 2016?), real (Annual Income) and categorical (State) variables. The response variable is binary (Biden/Trump).

The naive Bayes classifier assumes that the different inputs coordinates are independent conditional on the class labels. That is, for an input vector[1] $x = (x_1, \ldots, x_p)^\intercal$, the conditional class distributions, for $k = 1, \ldots, K$, factorise as

$$g_k(x) = \prod_{j=1}^{p} g_{kj}(x_j; \theta_{kj}) \tag{2.19}$$

[1]We use an abuse of notation, where x_j denotes the j one-dimensional component of the vector x. Note to be confused with x_i, which will denote the ith p-dimensional observation.

where g_{kj} are conditional pmf/pdf on $\mathbb{R}$ of $X_j \mid Y = k$, parameterised by θ_{kj}. Clearly, the independence assumption is "naive" and never satisfied. But this assumption allows to significantly reduce the number of parameters to estimate and make inference much easier. Although the generative model is quite simple, naive Bayes often works quite well in practice for predicting the class labels.

In the above example, we need to estimate Pr(person x voted in 2016 | Biden supporter), the probability that someone voted in 2016, given it is a Biden supporter. Similarly, we have to estimate Pr(person x voted in 2016 | Trump supporter). We should do the same for the other variables "Annual income" and "State", for each candidate, resulting in the estimation of the parameters of 6 conditional univariate pmf or pdf.

We need to define the parametric models $g_{kj}(x_j; \theta_{kj})$. There are standard choices depending on the type of variable.

- **Real-Valued Features**. For real-valued variables, such as the annual income, a standard choice is to use a Gaussian model, with unknown mean μ_{kj} and variance σ^2_{kj}, that is

$$g_{kj}(x_j; \theta_{kj}) = \varphi(x_j; \mu_{kj}, \sigma^2_{kj})$$

  where $\theta_{kj} = (\mu_{kj}, \sigma^2_{kj})$. Of course, one can use other parametric distributions.

- **Binary Features** If $x_j \in \{0, 1\}$ (such as "Voted in 2016?"), we use a Bernoulli distribution with parameter $\theta_{kj} \in [0, 1]$

$$g_{kj}(1; \theta_{kj}) = 1 - g_{kj}(0; \theta_{kj}) = \theta_{kj}.$$

- **Categorical Features** If $x_j \in \{1, \dots, C\}$ is a categorical feature (such as the state), we can use the multinomial distribution, with parameters $\theta_{kj,1}, \dots, \theta_{kj,C}$ where $\sum_{c=1}^{C} \theta_{kj,c} = 1$. For any $c = 1, \dots, C$

$$g_{kj}(c; \theta_{kj}) = \theta_{kj,c}$$

For the special binary case $C = 2$, it reduces to the Bernoulli distribution.

Parameter estimation. We can estimate the parameters using maximum likelihood. The log-likelihood takes the form

$$\begin{aligned}
\ell((\pi_k)_{k=1\dots,K}, (\theta_{kj})_{k=1,\dots,K;j=1,\dots,p}) &= \sum_{i=1}^{n} (\log(\pi_{y_i}) + \log g_{y_i}(x_i)) \\
&= \sum_{k=1}^{K} m_k \log(\pi_k) + \sum_{k=1}^{K} \sum_{i|y_i=k} \log g_k(x_i) \\
&= \sum_{k=1}^{K} m_k \log(\pi_k) + \sum_{k=1}^{K} \sum_{i|y_i=k} \sum_{j=1}^{p} \log g_{kj}(x_{ij}; \theta_{kj}) \\
&= \underbrace{\sum_{k=1}^{K} m_k \log(\pi_k)}_{\ell_1((\pi_k))} + \sum_{j=1}^{p} \sum_{k=1}^{K} \underbrace{\sum_{i|y_i=k} \log g_{kj}(x_{ij}; \theta_{kj})}_{\ell_{kj}(\theta_{kj})}
\end{aligned}$$

The log-likelihood therefore factorises as $\ell_1((\pi_k)) + \sum_{j,k} \ell_{kj}(\theta_{kj})$, and each term of the sum can be optimised separately. For π_k, the MLE is given as before:

$$\widehat{\pi}_k = \frac{m_k}{n}.$$

For the other parameters, the MLE is given by

$$\widehat{\theta}_{kj} = \arg\max_{\theta_{kj}} \sum_{i|y_i=k} \log g_{kj}(x_{ij}; \theta_{kj})$$

and depends on the choice of the model. In particular, we have

- Gaussian likelihood: $\widehat{\theta}_{kj} = (\widehat{\mu}_{kj}, \widehat{\sigma}^2_{kj})$ where

$$\widehat{\mu}_{kj} = \frac{1}{m_k} \sum_{i|y_i=k} x_{ij}, \quad \widehat{\sigma}^2_{kj} = \frac{1}{m_k} \sum_{i|y_i=k} (x_{ij} - \widehat{\mu}_{kj})^2$$

- Bernoulli likelihood:
$$\widehat{\theta}_{kj} = \frac{\sum_{i|y_i=k} x_{ij}}{m_k}$$
- Multinomial likelihood: For each $c = 1, \ldots, C$,
$$\widehat{\theta}_{kj,c} = \frac{\sum_{i|y_i=k} \mathbb{1}_{x_{ij}=c}}{m_k}$$

Once the parameters are estimated, the conditional distribution $\Pr(Y = k \mid X = x)$ is approximated by

$$\frac{\widehat{\pi}_k \prod_{j=1}^p g_{kj}(x; \widehat{\theta}_{kj})}{\sum_{k'=1}^K \widehat{\pi}_{k'} \prod_{j=1}^p g_{k'j}(x; \widehat{\theta}_{k'j})}$$

and the naive Bayes classifier is

$$\widehat{h}^{(d)}(x) = \underset{k \in \{1,\ldots,K\}}{\arg\max} \log(\widehat{\pi}_k) + \sum_{j=1}^p \log(g_{kj}(x; \widehat{\theta}_{kj})).$$

Missing data. Going back to our example, consider that the voter did not reveal if they had voted in 2016:

Voted in 2016?	Annual Income	State	Candidate Choice
Y	50K	OK	Biden
N	173K	CA	Biden
Y	80K	NJ	Trump
Y	150K	WA	Biden
Y	85K	IL	Trump
⋮	⋮	⋮	⋮
Y	1050K	NY	Biden
N	35K	CA	Trump
?	**100K**	**NY**	**?**

For our new observation $\widetilde{x} = (\widetilde{x}_1, \widetilde{x}_2, \widetilde{x}_3)$, the value of the first variable $\widetilde{x}_1$ is missing, and we can thus only use $(\widetilde{x}_2, \widetilde{x}_3)$ to predict its candidate choice. Under the naive Bayes assumption,

$$g_k((\widetilde{x}_1, \widetilde{x}_2, \widetilde{x}_3)) = \prod_{j=1}^3 g_{kj}(\widetilde{x}_j; \theta_{kj})$$

Marginalising the above expression with respect to $\widetilde{x}_1$ gives the conditional distribution of $(\widetilde{X}_2, \widetilde{X}_3)$ given $\widetilde{Y} = k$, which is

$$\sum_{\widetilde{x}_1=0,1} g_k((\widetilde{x}_1, \widetilde{x}_2, \widetilde{x}_3)) = \prod_{j=2}^3 g_{kj}(\widetilde{x}_j; \theta_{kj}). \tag{2.20}$$

Hence, using Bayes rule,

$$\Pr(\widetilde{Y} = k \mid (\widetilde{X}_2, \widetilde{X}_3) = (\widetilde{x}_2, \widetilde{x}_3)) \propto \pi_k \prod_{j=2}^3 g_{kj}(\widetilde{x}_j; \theta_{kj}).$$

The naive Bayes prediction is therefore given by

$$\widehat{h}^{(d)}((\widetilde{x}_2, \widetilde{x}_3)) = \underset{k \in \{1,\ldots,K\}}{\arg\max} \log(\widehat{\pi}_k) + \sum_{j=2}^p \log(g_{kj}(x; \widehat{\theta}_{kj})).$$

Note that missing data can be handled in a similar way for any generative approach, by marginalising the variable of interest in $g_k(x)$. This is done without extra computation for naive Bayes (see Equation (2.20)) due to the independence assumption. Without the independence this would require summation/integration.

Dealing with missing data in the training set is also quite easy. Assume that some entries are missing for the variable Voted in 2016:

Voted in 2016?	Annual Income	State	Candidate Choice
?	50K	OK	Biden
?	173K	CA	Biden
Y	80K	NJ	Trump
Y	150K	WA	Biden
?	85K	IL	Trump
⋮	⋮	⋮	⋮
Y	1050K	NY	Biden
N	35K	CA	Trump
?	**100K**	**NY**	**?**

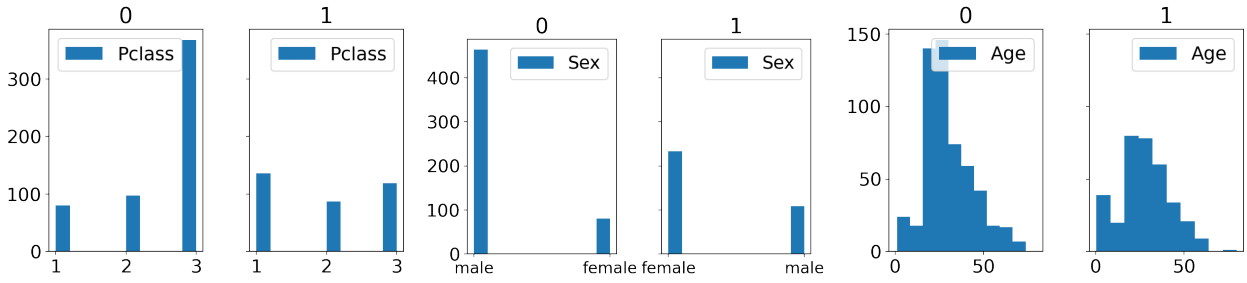
For example, let's say that for Biden voters, 103 had voted in 2016, 54 had not, and 25 didn't answer. We can simply estimate $\theta_{Biden,1}$ based on the non-missing answers, that is $\widehat{\theta}_{Biden,1} = 103/157$.

Naive Bayes and Curse of dimensionality. By making the independence assumption, naive Bayes breaks the curse of dimensionality and avoids overfitting. Assume for illustration that all the features x_j of the vector x are binary, hence $x \in \{0,1\}^p$, where $\{0,1\}^p$ has 2^p elements. If one were to parameterise $g_k(x)$, this would require $O(2^p)$ parameters. By contrast, using naive Bayes only requires $O(p)$ parameters (one for each dimension of the input vector x).

Example 16. *As an example consider the dataset consisting of attributes of 887 passengers of the Titanic, and whether or not they survived the tragedy.*

Class	*Sex*	*Age*	*Survived*
3	*male*	*22*	*N*
1	*female*	*38*	*Y*
3	*female*	*26*	*Y*
1	*female*	*35*	*Y*
3	*male*	*35*	*N*
⋮	⋮	⋮	⋮

The response variable y is binary (survived=1, did not survived=0), and the input vector x is three dimensional $x = (x_1, x_2, x_3)$ where x_1 is the class of the passenger (1st,2nd or 3rd), x_2 is the sex (female/male), x_3 is the age. We aim at deriving a classifier to predict the survival based on the three attributes. Histograms of the input variables for passengers who survived and did not survived are shown in Figure 2.6.



(a) Passenger class (b) Sex (c) Age

Figure 2.6: Histogram of the (a) Passenger class (b) Sex and (c) Age of the passenger amongst those who did not survive (left plot) and did survive (right plot).

We use a multinomial model with $C = 3$ for g_{k1} (passenger class), a Bernoulli model for g_{k2} (sex) and a Gaussian model for g_{k3} (age). As can be seen from Figure 2.6(c) the Gaussian distribution provides a poor fit to the data, and one could consider alternative models. The classifier is defined by

$$\widehat{h}^{(d)}(x) = \arg\max_{k\in\{0,1\}} \log(\widehat{\pi}_k) + \sum_{j=1}^{3} \log(g_{kj}(x;\widehat{\theta}_{kj})).$$

The estimated parameters are reported in the following table

Name	*Parameters*	*Did not Survived (k=0)*	*Survived (k=1)*
Prior class proba	π_k	0.61	0.39
Pass. Class	$(\widehat{\theta}_{k1,1}, \ldots, \widehat{\theta}_{k1,3})$	$(0.15, 0.18, 0.67)$	$(0.40, 0.25, 0.35)$
Sex (F=1)	$\widehat{\theta}_{k2}$	0.15	0.32
Age	$(\widehat{\mu}_{k3}, \widehat{\sigma}_{k3})$	$(30.1, 13.9)$	$(28.4, 14.4)$

Here are some predictions from the naive Bayes classifier.

Class	*Sex*	*Age*	*Predicted Survival*	*Estimated Probability of survival*
3	*female*	*30*	*Y*	*0.587*
1	*female*	*30*	*Y*	*0.882*
3	*male*	*20*	*N*	*0.114*
1	*male*	*20*	*N*	*0.403*

2.6 Summary

In this chapter, we have seen three different classes of generative classifiers:

- LDA and QDA assume that the input X is normally distributed given the class $Y = k$, with the same covariance (LDA) or different covariances (QDA)
- Naive Bayes assumes that the different dimensions of the input X are conditionally independent given the class $Y = k$

The advantages of using a generative approach are

- The assumptions made on the data generating process can be statistically tested (Gaussianity, independence);
- Even if the assumptions do not hold, the classifier may still provide good predictions in practice;
- The classifier can easily handle missing data in the input vector;
- It leads to interpretable predictions.

The main drawback is that it makes strong assumptions on the data generating process, that are rarely met in practice. This limits the flexibility of the generative approach.

3 — Further concepts in statistical learning

In this chapter, we will discuss key concepts in statistical machine learning:

- Nonlinear input transformation/expansion,
- Overfitting and bias-variance tradeoff
- Regularisation
- Cross-validation

Throughout this chapter, we will use a simple regression example to illustrate these concepts, and frame the learning procedure as an empirical risk minimisation problem over some class of prediction rules. The concepts and tools discussed in this chapter however apply equally when dealing with classification tasks, or when using plug-in methods. We start by recalling some background material on multivariate linear regression.

3.1 Multivariate linear regression

Let $((x_1, y_1), \ldots, (x_n, y_n))$ be the input/target data, with $x_i \in \mathbb{R}^p$, and let $y_i \in \mathbb{R}$. For an input variable $x \in \mathbb{R}^p$, let $\widetilde{x} = \begin{pmatrix} 1 \\ x \end{pmatrix} \in \mathbb{R}^{p+1}$. Consider the class of linear prediction rules

$$\mathcal{H} = \{h(x) = \widetilde{x}^\intercal \beta = (1 \;\; x^\intercal)\beta \mid \beta \in \mathbb{R}^{p+1}\}.$$

The empirical risk minimiser under a squared error loss for the class $\mathcal{H}$ is $\widehat{h}^{(d)}(x) = \widetilde{x}^\intercal \widehat{\beta}$ where

$$\begin{aligned}
\widehat{\beta} &= \operatorname*{arg\,min}_{\beta \in \mathbb{R}^{p+1}} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \\
&= \operatorname*{arg\,min}_{\beta \in \mathbb{R}^{p+1}} \sum_{i=1}^{n} (y_i - \widetilde{x}_i^\intercal \beta)^2 \\
&= \operatorname*{arg\,min}_{\beta \in \mathbb{R}^{p+1}} \| \mathbf{y} - \widetilde{\mathbf{X}}\beta \|^2
\end{aligned}$$

where $\mathbf{y} = (y_1, \ldots, y_n)^\intercal \in \mathbb{R}^n$ and $\widetilde{\mathbf{X}}$ is the n-by-$(p+1)$ design matrix defined by

$$\widetilde{\mathbf{X}} = \begin{pmatrix} 1 & x_{11} & \ldots & x_{1p} \\ 1 & x_{21} & \ldots & x_{2p} \\ \ldots & \ldots & \ldots & \ldots \\ 1 & x_{n1} & \ldots & x_{np} \end{pmatrix}. \tag{3.1}$$

We have

$$\begin{aligned}
\| \mathbf{y} - \widetilde{\mathbf{X}}\beta \|^2 &= (\mathbf{y} - \widetilde{\mathbf{X}}\beta)^\intercal (\mathbf{y} - \widetilde{\mathbf{X}}\beta) \\
&= \beta^\intercal \widetilde{\mathbf{X}}^\intercal \widetilde{\mathbf{X}} \beta - 2(\widetilde{\mathbf{X}}^\intercal \mathbf{y})^\intercal \beta + \text{const.}
\end{aligned}$$

Using Equations (3) and (4), we obtain the derivative

$$\frac{\partial(\| \mathbf{y} - \widetilde{\mathbf{X}}\beta \|^2)}{\partial \beta} = 2\widetilde{\mathbf{X}}^\intercal \widetilde{\mathbf{X}} \beta - 2\widetilde{\mathbf{X}}^\intercal \mathbf{y}$$

Setting this to 0, and assuming that $\widetilde{\mathbf{X}}^\intercal \widetilde{\mathbf{X}}$ has full rank, we obtain the estimate

$$\widehat{\beta} = (\widetilde{\mathbf{X}}^\intercal \widetilde{\mathbf{X}})^{-1} \widetilde{\mathbf{X}}^\intercal \mathbf{y}\,.$$

3.2 Nonlinear input expansion

The learned prediction rule $\widehat{h}^{(d)}(x) = \widetilde{x}^\intercal \widehat{\beta}$ has the interesting property that it can be obtained in closed-form, but it can only capture linear relations between the input x and the response y. It is actually possible to obtain **non-linear** prediction rules by using a class of linear prediction rules on an **extended input/feature space**.

For an input space $\mathcal{X}$, consider a function $\phi : \mathcal{X} \to \mathcal{X}'$, where $\mathcal{X}'$ is the extended input (or feature) space. The function ϕ is a known, typically nonlinear function. The extended input/feature space $\mathcal{X}'$ may be of dimension lower, equal or larger than the original input space $\mathcal{X}$. The function ϕ may be used to reduce dimensionality and extract interesting features of the data (for instance using PCA), or to expand the dimension of the feature space to allow for more complex prediction rules. One can apply a given learning method to the transformed inputs $\phi(x_1), \ldots, \phi(x_n)$ instead of the original inputs. This potentially allows to capture more complex prediction rule under simple learning methods.

Let $\phi : \mathcal{X} \to \mathbb{R}^{M+1}$ for some $M \geq 0$. One can for example consider the class of linear prediction rules on the transformed inputs, that is

$$\mathcal{H} = \{h(x) = \phi(x)^\intercal \beta \text{ where } \beta \in \mathbb{R}^{M+1}\}.$$

Note that the class of prediction rules are linear functions in the transformed input space $\mathcal{X}'$, but nonlinear function in the original input space $\mathcal{X}$ if the transformation ϕ is nonlinear. The ERM under a squared loss can be found in closed form, as it corresponds to the ordinary least square estimate on the transformed inputs $\phi(x_1), \ldots, \phi(x_n)$:

$$\widehat{\beta} = \operatorname*{arg\,min}_{\beta \in \mathbb{R}^{M+1}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \phi(x_i)^\intercal \beta)^2 = (\mathbf{\Phi}^\intercal \mathbf{\Phi})^{-1} \mathbf{\Phi}^\intercal \mathbf{y}$$

where

$$\mathbf{\Phi} = (\phi(x_1), \ldots, \phi(x_n))^\intercal \in \mathbb{R}^{n \times (M+1)}, \quad \mathbf{y} = (y_1, \ldots, y_n)^\intercal \in \mathbb{R}^n \,. \tag{3.2}$$

Example 17. *(sinusoid) As an illustrative example, consider that (X, Y) has a joint distribution defined by*

$$Y = \sin(2\pi X) + \epsilon, \quad X \sim U(0,1), \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

where X is independent of ϵ, and $\sigma > 0$. The Bayes prediction rule is

$$h^*(x) = \mathbb{E}[Y|X = x] = \sin(2\pi x).$$

The risk for a prediction rule h is given by

$$R(h) = \mathbb{E}[(Y - h(X))^2] = \mathbb{E}[(\sin(2\pi X) + \epsilon - h(X))^2] = \sigma^2 + \int_0^1 (\sin(2\pi x) - h(x))^2 dx$$

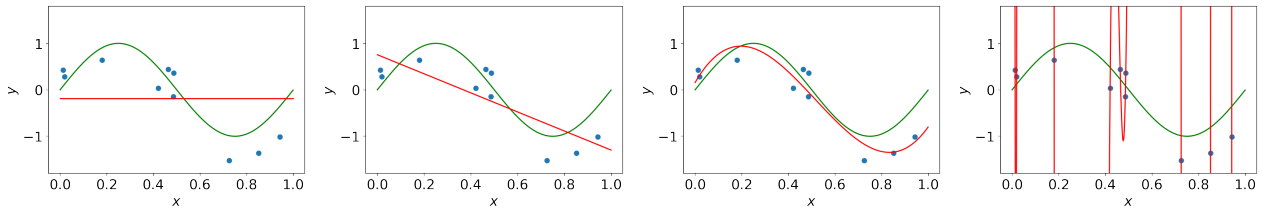
and the Bayes risk is $R(h^*) = \sigma^2$. For $M \geq 0$, let $\mathcal{H}_M$ be the set of polynomials of order M

$$\mathcal{H}_M = \{h : \mathcal{X} \to \mathcal{Y} \mid h(x) = \sum_{j=0}^{M} \beta_j x^j, \beta_j \in \mathbb{R} \text{ for } j = 0 \ldots, M\}.$$

Note that the hypothesis classes are nested $\mathcal{H}_0 \subset \mathcal{H}_1 \subset \ldots$. Note also that $h^* \notin \mathcal{H}_M$, for all M.

We generate $n = 10$ realisations (x_i, y_i), $i = 1, \ldots, n$, and calculate the ERM prediction rule under the classes $\mathcal{H}_M$, for $M = 0, \ldots, 9$. The learned prediction rules $\widehat{h}^{(d)}$ are plotted in Figure 3.1. For $M = 0$ and $M = 1$, the learned prediction rules are too simple. We say that we are underfitting. For $M = 3$, the learned prediction rule is close to the Bayes prediction rule. For $M = 9$, the learned prediction interpolates the data, but is very far from the Bayes prediction rule when x is not in the training set. As shown in Table 3.1, the estimated parameters take very large values for $M = 9$ suggesting the fact that the estimator has very large variance. We say that we are overfitting: the model is too complex for the amount of data available, and is fitting both the signal and the noise. Figure 3.2 shows the (generalisation/population) risk and the empirical risk of the learned prediction rule as a function of the order of the polynomial. The empirical risk decreases with the model complexity, to reach zero for $M = 9$. The true risk has a U shape: it first decreases (underfitting regime), then increases (overfitting regime).

Overfitting can have disastrous effect. When fitting a polynomial of degree 11 on the house pricing data, we obtain a prediction rule that decreases for large living area, as shown in Figure 3.3.



(a) $M = 0$ (b) $M = 1$ (c) $M = 3$ (d) $M = 9$

Figure 3.1: ERM prediction rules (in red) for linear models with polynomial expansion with degrees $M = 0, 1, 3$ and 9. The data are represented by blue dots and the Bayes prediction rule is in green.

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$\widehat{\beta}_0$	-0.19	0.76	0.16	171.61
$\widehat{\beta}_1$		-2.05	8.66	-24859.77
$\widehat{\beta}_2$			-27.29	1052225
$\widehat{\beta}_3$			17.67	-13060914
$\widehat{\beta}_4$				75449695
$\widehat{\beta}_5$				-239878655
$\widehat{\beta}_6$				446010421
$\widehat{\beta}_7$				-483353259
$\widehat{\beta}_8$				282706993
$\widehat{\beta}_9$				-68922625

Table 3.1: Estimates under for polynomial class of degree 0, 1, 3, and 9.

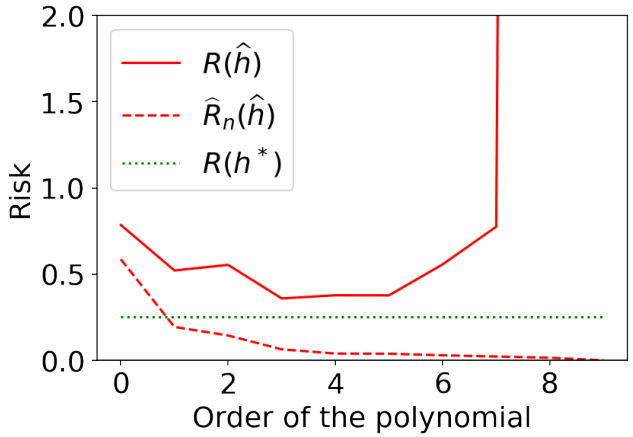


Figure 3.2: Generalisation Risk (plain red line) and Empirical risk (dashed red line) as a function of the model complexity. The Bayes risk is represented by a dotted green line.

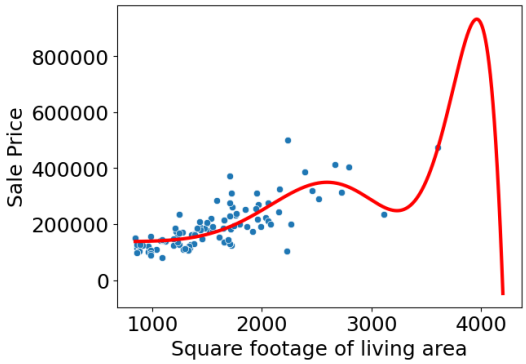


Figure 3.3: Fitting the house price dataset with a polynomial function of order 11.

3.3 Estimation-Approximation and Bias-variance trade-off

Estimation-Approximation error. For a given set of prediction rules $\mathcal{H} \subset \mathcal{F}$, called an **hypothesis class**, denote

$$h^*_{\mathcal{H}} = \arg\min_{h \in \mathcal{H}} R(h)$$

the best prediction rule, in terms of risk minimisation, amongst the set $\mathcal{H}$. Note that by definition, we have $R(\widehat{h}^{(d)}) \geq R(h^*_{\mathcal{H}}) \geq R(h^*)$, where $\widehat{h}^{(d)}$ is the empirical risk minimiser (1.30). The excess risk can be decomposed as a sum of two terms

$$\underbrace{R(\widehat{h}^{(d)}) - R(h^*)}_{\text{excess risk}} = \underbrace{R(\widehat{h}^{(d)}) - R(h^*_{\mathcal{H}})}_{\text{estimation error}} + \underbrace{R(h^*_{\mathcal{H}}) - R(h^*)}_{\text{approximation error}}$$

- The approximation error is the difference between the risk of the best prediction rule within the class and the Bayes risk. If the hypothesis class is large, more complex prediction rules can be captured, and the approximation error is thus smaller.
- The estimation error is the difference between the risk of the learned prediction rule and of the best prediction rule in the class. Larger classes have a large number of parameters to estimate, leading to a larger estimation error.

This approximation-estimation trade-off and how it relates to the model complexity is illustrated in Figure 3.4.

Example 18. *(sinusoid, continued) For the simulated example described in the previous section, the model complexity corresponds to the degree of the polynomial. Figure 3.5 shows the learned prediction rules and the best prediction rule for each class. As the degree increases, the best prediction rule in the class becomes closer and closer to the Bayes prediction rule. Figure 3.6 shows the excess risk and the estimation/approximation error as a function of the degree of the polynomial.*

Approximation error and size of the dataset. The approximation error is a quantity that only depends on the choice of the class $\mathcal{H}$, not the dataset. Under mild assumptions, in general we would have that the estimation error goes to 0 as the number of observations n goes to infinity. Hence the excess risk converges to the approximation error or, similarly, the risk $R(\widehat{h})$ converges to the risk $R(h^*_{\mathcal{H}})$ of the best prediction rule in the class as $n \to \infty$.

The empirical risk underestimates the true risk. The empirical risk of the learned rule $\widehat{R}_n(\widehat{h})$ typically underestimates the true risk $R(\widehat{h})$ of that prediction rule. Note that, by definition of the ERM in the class $\mathcal{H}$, we have $R(\widehat{h}^{(d)}) \geq R(h^*_{\mathcal{H}})$ for any dataset d. But (see Problem Sheet)[1]

$$\mathbb{E}[\widehat{R}^{(D)}_n(\widehat{h}^{(D)})] \leq R(h^*_{\mathcal{H}})$$

where the expectation is taken with respect to the random sample D. The generalisation gap is

$$R(\widehat{h}) - \widehat{R}_n(\widehat{h}).$$

This generalisation gap typically converges to 0 as the sample size n increases, as both $R(\widehat{h})$ and $\widehat{R}_n(\widehat{h})$ will converge to $R(h^*_{\mathcal{H}})$. The relation between the generalisation gap and the sample size is depicted in Figure 3.3.

[1] we add the superscript D to emphasize the dependence on the random sample D.

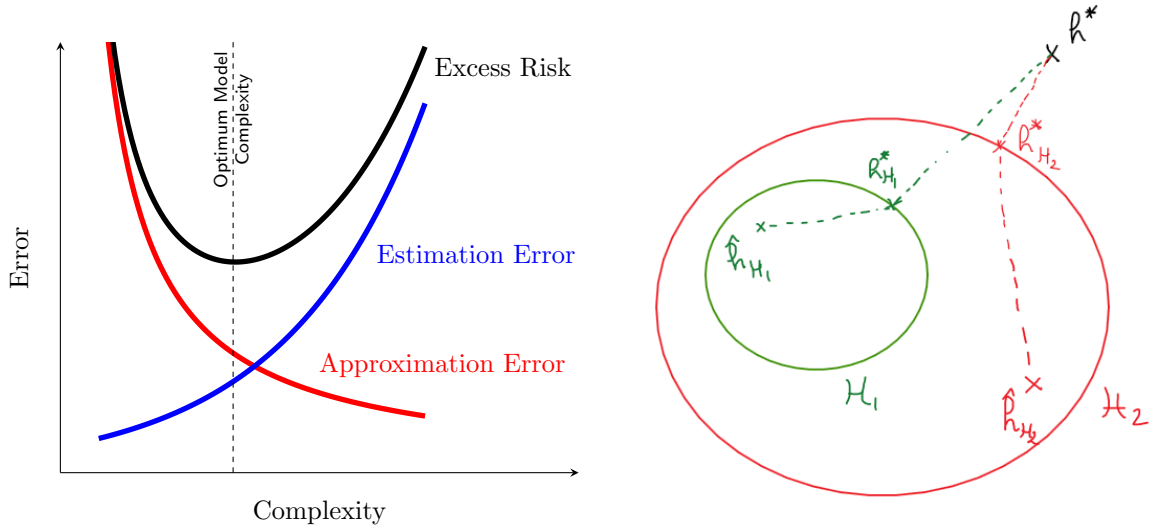
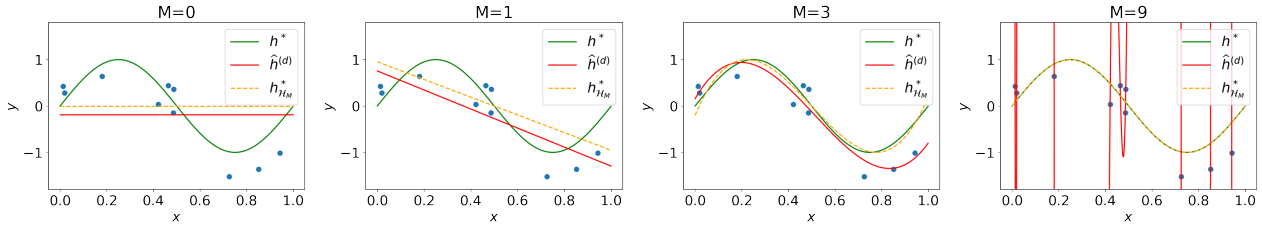


Figure 3.4: Illustration of the approximation-estimation tradeoff. (Left) Approximation and estimation error as a function of the model complexity. Simple models have large approximation error and small estimation error. Complex models have large estimation error and small approximation error. (Right) For two hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2$ ($\mathcal{H}_1$ is simpler than $\mathcal{H}_2$), and a Bayes rule h^* outside of the classes. $\mathcal{H}_1$ will have larger approximation error $R(h^*_{\mathcal{H}_2}) \leq R(h^*_{\mathcal{H}_1})$ but typically smaller estimation error.



(a) $M = 0$ (b) $M = 1$ (c) $M = 3$ (d) $M = 9$

Figure 3.5: ERM learned prediction rules (in red) for linear models with polynomial expansion with degrees $M = 0, 1, 3$ and 9. The best prediction rule within the class of polynomials of order M is shown in orange. The data are represented by blue dots and the Bayes prediction rule is in green.

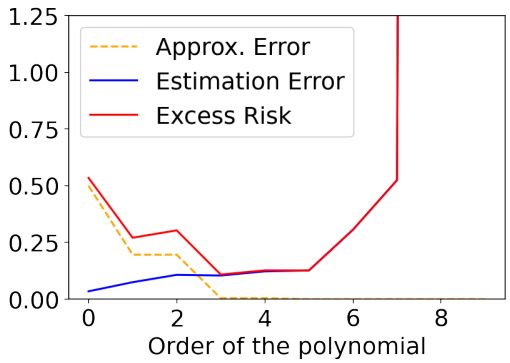


Figure 3.6: Excess risk $R(\widehat{h}^{(d)}) - R(h^*)$ (red), approximation error (orange) and estimation error (blue) as a function of the complexity.

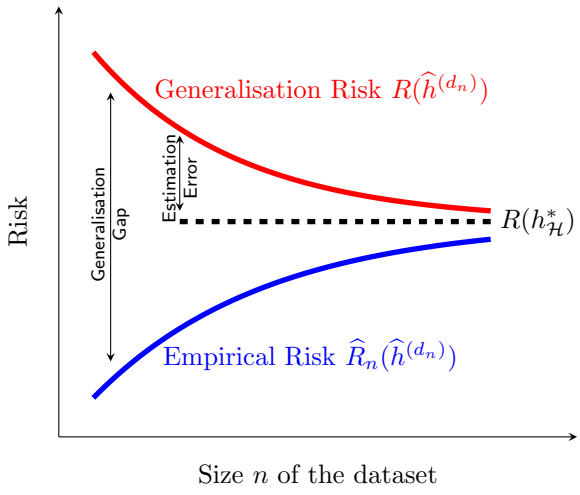


Figure 3.7: Generalisation and Empirical risk as a function of the size n of the dataset for a given hypothesis class $\mathcal{H}$.

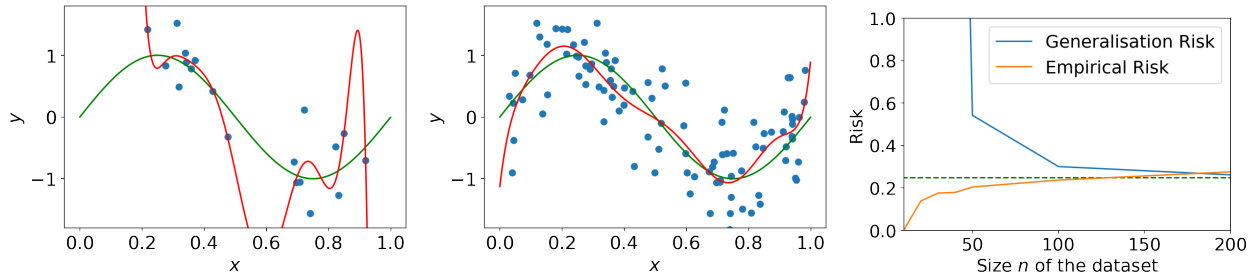


Figure 3.8: Learned prediction rule with a polynomial of order 9 with (left) $n = 20$ and (middle) $n = 100$ observations. (Right) Generalisation and Empirical Risks as a function of the size n of the dataset.

Example 19. *(sinusoid, continued) Going back to our simulated example, consider the ERM with hypothesis class the set of polynomials of order 9, with a dataset of growing size. Figure 3.8 shows the estimated prediction rules for $n = 20, 100$, and the empirical and generalisation risks as a function of the sample size. Both converge to the same value, corresponding to the risk of the best prediction rule within the class of polynomials of order 9.*

Bias-variance decomposition. For the squared loss, another traditional interpretable decomposition of the excess risk is the bias-variance decomposition. Recall that for the squared loss, the excess risk takes the form

$$R(\widehat{h}^{(D)}) - R(h^*) = \mathbb{E}_X[(\widehat{h}^{(D)}(X) - h^*(X))^2].$$

The above excess risk is a random variable, as it depends on the dataset D. Taking the expectation over the dataset D, we obtain the expected excess risk

$$\begin{aligned}\mathbb{E}_D[R(\widehat{h}^{(D)}) - R(h^*)] &= \mathbb{E}_D\mathbb{E}_X[(\widehat{h}^{(D)}(X) - h^*(X))^2] \\ &= \mathbb{E}_X\mathbb{E}_D[(\widehat{h}^{(D)}(X) - h^*(X))^2]\end{aligned}$$

For any $x \in \mathcal{X}$, the random variable $\widehat{h}^{(D)}(x)$ is an estimator of the Bayes predictor $h^*(x)$. This estimator has mean $\bar{h}_{\mathcal{H},n}(x) = \mathbb{E}_D[\widehat{h}^{(D)}(x)]$ and bias and variance

$$b_{\mathcal{H},n}(x) = \bar{h}_{\mathcal{H},n}(x) - h^*(x) \tag{3.3}$$

$$v_{\mathcal{H},n}(x) = \mathbb{E}_D[(\widehat{h}^{(D)}(x) - \bar{h}_{\mathcal{H},n}(x))^2] \tag{3.4}$$

For any x, we have

$$\mathbb{E}_D[(\widehat{h}^{(D)}(x) - h^*(x))^2] = \mathbb{E}_D[(\widehat{h}^{(D)}(x) - \bar{h}_{\mathcal{H},n}(x))^2 + (\bar{h}_{\mathcal{H},n}(x) - h^*(x))^2 + 2(\widehat{h}^{(D)}(x) - \bar{h}_{\mathcal{H},n}(x))(\bar{h}_{\mathcal{H},n}(x) - h^*(x))] \tag{3.5}$$

$$= v_{\mathcal{H},n}(x) + b_{\mathcal{H},n}(x)^2. \tag{3.6}$$

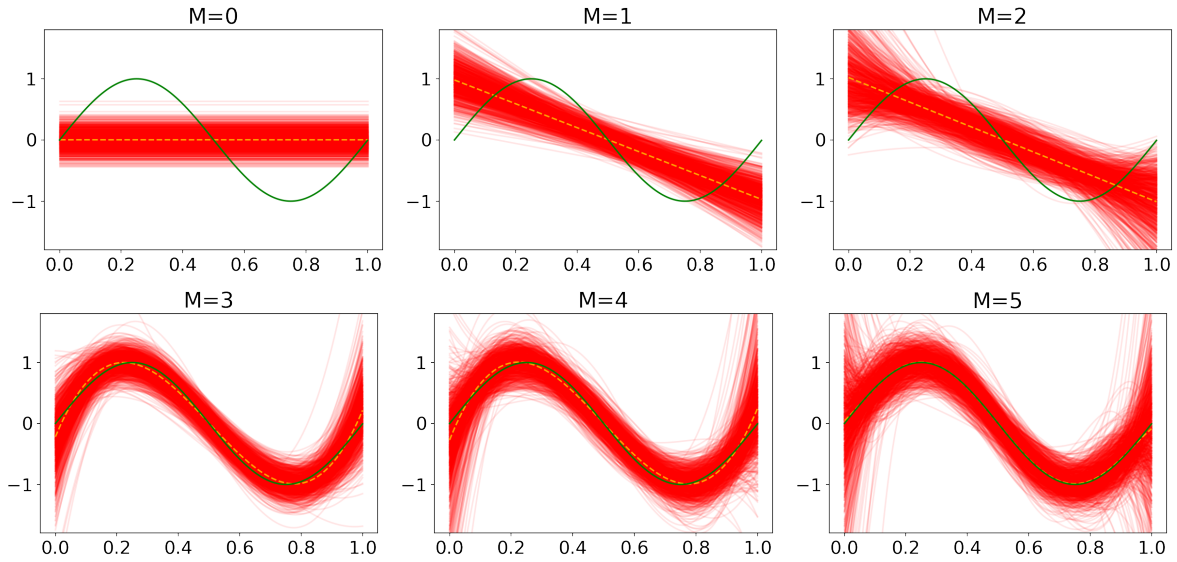


Figure 3.9: Learned ERM rules for different datasets of size $n = 30$, for different orders of the polynomial. As the order M increases, variance of the predictions increases, in particular near the boundary.

It follows that

$$\underbrace{\mathbb{E}_D[R(\widehat{h}^{(D)}) - R(h^*)]}_{\text{average excess risk}} = \underbrace{\mathbb{E}_X[v_{\mathcal{H},n}(X)]}_{\text{variance term}} + \underbrace{\mathbb{E}_X[b_{\mathcal{H},n}(X)^2]}_{\text{bias term}} \tag{3.7}$$

- The variance term relates to the estimation error; it will typically be large for large hypothesis sets $\mathcal{H}$. This term vanishes as the number of observations grows.
- The bias term relates to the approximation error; it will typically be large for small hypothesis sets $\mathcal{H}$. This term does not vanish as the number of observations grows.

Example 20. *(sinusoid, continued) Using the same simulated example, with $n = 30$ examples, we sample different datasets and represent for each dataset the learned prediction rule in Figure 3.3. The mean of these learned prediction rule is represented by a dotted line. For small M, the mean is far from the Bayes prediction rule (green line), and the learned prediction rules have low variance. As M increases, the mean value comes closer to the Bayes prediction rule, but the variance increases. This is further shown in Figure 3.10 that shows the function $b_n(x)$ and $v_n(x)$ for different polynomial orders. For small M, the bias term is large and the variance term small. The variance increases with the model order, while the bias term decreases. The bias-variance errors are shown in Figure 3.11 as a function of the order of the model.*

3.4 Regularised Empirical Risk Minimisation

An alternative to considering classes of different capacity is to consider a single (large) class of prediction rules, and penalise more complex prediction rules within this class. Let $\text{pen} : \mathcal{F} \to \mathbb{R}_+$ be a **penalty** or **regularisation** function. For a given prediction rule h, $\text{pen}(h)$ is a measure of the complexity of the prediction rule h; larger values correspond to more complex prediction rules, while smaller values are less complex prediction rules.

Definition 21. *Let d be a dataset. For a loss function L, a hypothesis class $\mathcal{H} \subset \mathcal{F}$, a penalty function* pen *and a **regularisation parameter** $\lambda \geq 0$, the regularised empirical risk minimiser is given by*

$$\widehat{h}^{(d)} = \operatorname*{arg\,min}_{h \in \mathcal{H}} \left\{ \widehat{R}_n(h) + \frac{\lambda}{n} \text{pen}(h) \right\}. \tag{3.8}$$

The empirical risk $\widehat{R}_n(h)$ measures the fit of the prediction rule to the data. More complex prediction rules will typically give a better fit on the data. The second term $\frac{\lambda}{n}\text{pen}(h)$ penalises the complexity of the prediction rule. The best prediction is the prediction rule achieving the best compromise between the fit to the data and the complexity. The regularisation parameter λ controls this trade-off. If $\lambda = 0$, this corresponds to the ERM

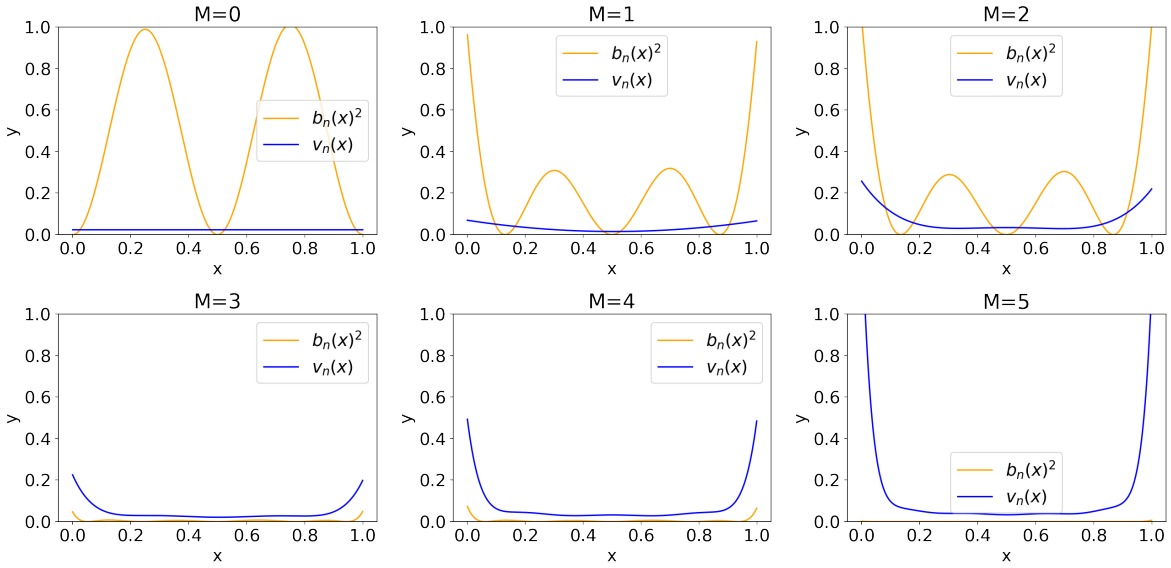


Figure 3.10: Simulated sinusoid example, with $n = 30$. Bias and variance as functions of x for different orders of the polynomial. For small order M, the bias term dominates, while for large M, the variance term dominates.

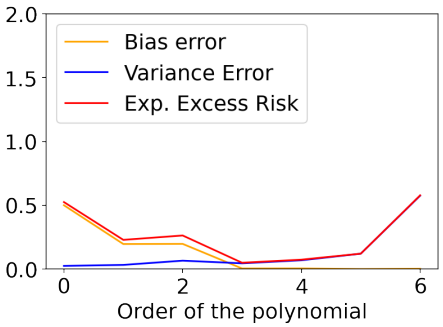


Figure 3.11: Bias-variance error and expected excess risk for the sinus dataset ($n = 30$).

solution: there is no penalty for the complexity, and the learned prediction rule is likely to overfit if the class is large. As $\lambda \to \infty$, a very large penalty is enforced for the model's complexity, and simpler prediction rules will be chosen. In terms of bias-variance decomposition, larger values of λ will lead to higher bias and smaller variance.

If the prediction rules $h \in \mathcal{H}$ are parameterised by a vector $\beta = (\beta_0, \ldots, \beta_M)^\intercal \in \mathbb{R}^{M+1}$, a standard measure of the complexity of the prediction rule is the square of the L2 norm $||\beta||^2 = \sum_{j=0}^{M} \beta_j^2$ of the vector β. The associated penalty, known as **Tikhonov regularisation**, is therefore

$$\text{pen}(h) = \|\beta\|^2 = \sum_{j=0}^{M} \beta_j^2.$$

For example, for linear regression under the squared loss, if $\mathcal{H}$ is the class of linear prediction rules on some transformed inputs $\phi(x)$,

$$\mathcal{H} = \{h(x) = \phi(x)^\intercal \beta \text{ where } \beta \in \mathbb{R}^{M+1}\}$$

the regularised ERM is $\widehat{h}^{(d)}(x) = \phi(x)^\intercal \widehat{\beta}$ where

$$\widehat{\beta} = \underset{\beta \in \mathbb{R}^{M+1}}{\arg\min} \sum_{i=1}^{n} (y_i - \phi(x_i)^\intercal \beta)^2 + \lambda ||\beta||^2. \tag{3.9}$$

This is the **ridge regression estimator**, which admits a closed-form solution

$$\widehat{\beta} = (\mathbf{\Phi}^\intercal \mathbf{\Phi} + \lambda I)^{-1} \mathbf{\Phi}^\intercal \mathbf{y}, \tag{3.10}$$

where I is the identity matrix and Φ and $\mathbf{y}$ are defined in Equation (3.2).

Proof.

$$\begin{aligned}\sum_{i=1}^{n} (y_i - \phi(x_i)^\intercal \beta)^2 + \lambda \|\beta\|^2 &= \|\mathbf{y} - \mathbf{\Phi}\beta\|^2 + \lambda \|\beta\|^2 \\ &= \beta^\intercal \mathbf{\Phi}^\intercal \mathbf{\Phi} \beta - 2(\mathbf{\Phi}^\intercal \mathbf{y})^\intercal \beta + \lambda \beta^\intercal \beta + \text{const.}\end{aligned}$$

Differentiating with respect to β, using (3) and (4), we obtain

$$\frac{\partial(\|\mathbf{y} - \mathbf{\Phi}\beta\|^2 + \lambda \|\beta\|^2)}{\partial \beta} = 2\mathbf{\Phi}^\intercal \mathbf{\Phi}\beta - 2\Phi^\intercal \mathbf{y} + 2\lambda\beta.$$

Setting the derivative to 0 gives the ridge regression estimator

$$\widehat{\beta} = (\mathbf{\Phi}^\intercal \mathbf{\Phi} + \lambda I)^{-1} \mathbf{\Phi}^\intercal \mathbf{y}\,.$$

□

Example 22. *(sinusoid, continued) For $n = 10$ observations, we consider the class of linear models with polynomial expansion of order 9, adding a regularisation term λ. The learned prediction rules are shown in Figure 3.12, and the value of the risk and empirical risk are shown in Figure 3.13(middle). As the value of the regularisation parameter λ increases, the learned prediction rule tends to have smaller coefficients, which are shrunk towards 0 (see Figure 3.13(left)). When λ goes to 0, the learned prediction rule tends to the non-regularised solution which overfits the data. For $n = 30$ observations, we give in Figure 3.13(right).*

Regularisation and sample size. As the sample size $n \to \infty$, for any fixed h,

$$\begin{aligned}\frac{\lambda}{n} \text{pen}(h) &\to 0 \\ \widehat{R}_n(h) &\to R(h) \quad \text{almost surely}\end{aligned}$$

The effect of the regularisation therefore vanishes as we have more data.

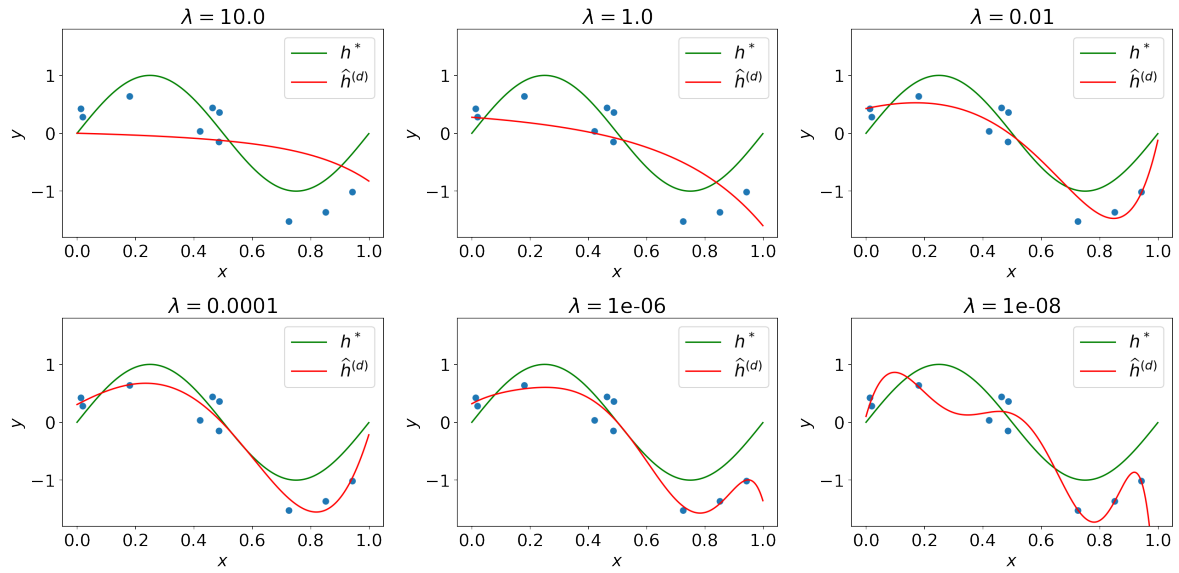


Figure 3.12: Simulated sinusoid example, with $n = 10$. Learned prediction rule for different values of the regularisation parameter λ.

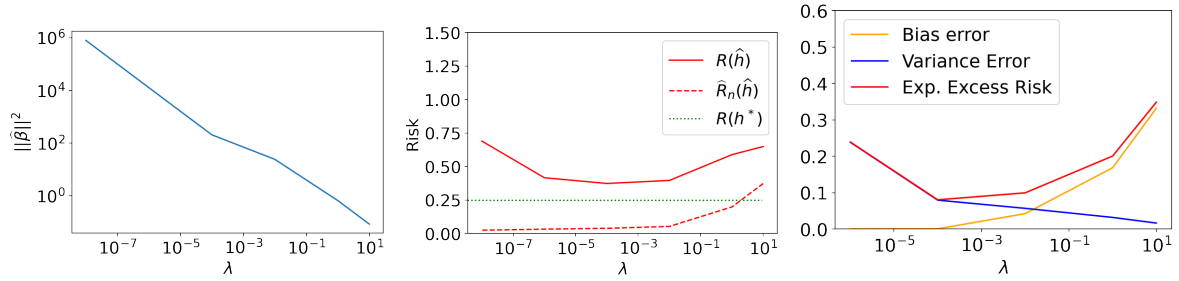


Figure 3.13: (Left) Norm of the estimated parameter β as a function of the regularisation parameter λ, on a log-log plot. (Middle) Risk and empirical risk as a function of the regularisation parameter λ. (Right) Bias vs Variance error as a function of the regularisation parameter λ.

Other penalisation functions. Alternative penalisation functions are also commonly used.

- L1 penalty
$$\text{pen}(h) = 2\|\beta\|_1 = 2\sum_{j=0}^{M} |\beta_j|$$
- L1+L2 (elastic net)
$$\text{pen}(h) = 2\delta\|\beta\|_1 + (1-\delta)\|\beta\|_2^2$$
for some $\delta \in [0, 1]$.

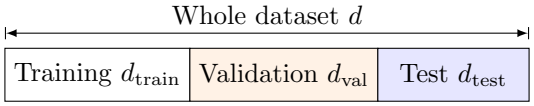
3.5 Test, validation and cross-validation

Assume that we obtain a set of $M_{\max}$ learned prediction rules $\widehat{h}_1^{(d)}, \ldots, \widehat{h}_{M_{\max}}^{(d)}$. These prediction rules may for example be obtained using ERM on hypothesis sets $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_{M_{\max}}$, of increasing complexity. They may obtained by using ERM on a large class $\mathcal{H}$, with different regularisation parameters $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{M_{\max}}$. More generally, they may be obtained by using different hyperparameters, learning algorithms, feature expansions, etc.

As we have seen, some learned prediction rules will overfit or underfit. We would like to choose the prediction rule with the smallest generalisation risk, and report this risk, as an estimate of the average error for predicting new observations. There are two caveats:

- The true risk cannot be computed as the true distribution of the data is unknown;
- The empirical risk of the learned prediction rule can be computed but, as we have seen, it largely underestimates the true risk, in particular for complex models.

The idea is to split the dataset in three parts: a training set, a validation set, and a test set.



Denote d_{train}, d_{val} and d_{test} respectively the training, validation and test set. As before, we assume that the training, validation and test set are realisations from D_{train}, D_{val} and D_{test}. Denote $\widehat{R}^{(d_{\text{train}})}(h)$, $\widehat{R}^{(d_{\text{val}})}(h)$ and $\widehat{R}^{(d_{\text{test}})}(h)$ the empirical risk of a prediction rule h on the training, validation and training sets. The procedure is then as follows.

Algorithm 1 Training-Validation-Test procedure

1. Training: For each $j = 1, \ldots, M_{\max}$, estimate the learned prediction rule $\widehat{h}_j^{(d_{\text{train}})}$ using the training set d_{train}. For instance, when considering the ERM on a class $\mathcal{H}_j$
$$\widehat{h}_j^{(d_{\text{train}})} = \arg\min_{h \in \mathcal{H}_j} \widehat{R}^{(d_{\text{train}})}(h).$$
2. Validation (model selection):
   a) For each j, calculate $\widehat{R}^{(d_{\text{val}})}(\widehat{h}_j^{(d_{\text{train}})})$ and report the best learner $\widehat{M} = \arg\min_j \widehat{R}^{(d_{\text{val}})}(\widehat{h}_j^{(d_{\text{train}})})$
   b) Re-train the prediction rule on both the training and validation set for the chosen class $\widehat{M}$. For instance, for ERM on a class $\mathcal{H}_j$
$$\widehat{h}^{(d_{\text{train}}, d_{\text{val}})} = \arg\min_{h \in \mathcal{H}_{\widehat{M}}} R^{(d_{\text{train}}, d_{\text{val}})}(h)$$
3. Test: Approximate the generalisation error with $\widehat{R}^{(d_{\text{test}})}(\widehat{h}^{(d_{\text{train}}, d_{\text{val}})})$.

Some general comments:

- In Step 1, as we use the training set d_{train} to estimate the $\widehat{h}_j^{(d_{\text{train}})}$, we cannot use $\widehat{R}^{(d_{\text{train}})}(\widehat{h}_j^{(d_{\text{train}})})$ as an estimate for $R(\widehat{h}_j^{(d_{\text{train}})})$
- We therefore use in step 2a) another dataset (the validation set), that has not been used for training. Note that, for any j, and any training set d_{train},
$$\mathbb{E}[\widehat{R}^{(D_{\text{val}})}(\widehat{h}_j^{(D_{\text{train}})}) \mid D_{\text{train}} = d_{\text{train}}] = R(\widehat{h}_j^{(d_{\text{train}})})$$

where the expectation is taken over the validation set. It therefore provides an unbiased estimate of the true risk of the jth learned prediction rule.

- After selecting the best class $\widehat{M}$, we could use $\widehat{h}_{\widehat{M}}^{(d_{\text{train}})}$ as the estimated prediction rule. However, to reduce the estimation error, it is better to re-train on both the training and validation sets for this model.
- The training and validation datasets having been used to estimate $\widehat{h}^{(d_{\text{train}},d_{\text{val}})}$, one has to calculate the empirical on a third dataset (the test set). We have

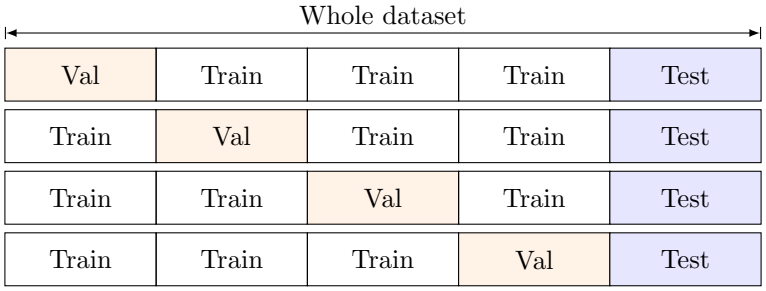
$$\mathbb{E}[\widehat{R}^{(D_{\text{test}})}(\widehat{h}^{(D_{\text{train}},D_{\text{val}})}) \mid D_{\text{train}} = d_{\text{train}}, D_{\text{val}} = d_{\text{val}}] = R(\widehat{h}^{(d_{\text{train}},d_{\text{val}})})$$

where the expectation is taken over the test set. It thus provides an unbiased estimate of the true risk of the learned prediction rule.

How to split the dataset? Any data in the validation and test set is not used to initially train the different models. If the training set is too small, this may lead to large estimation errors for the learned prediction rules. Alternatively, if the validation set or test sets are small, this leads to a large error on estimated risks. In the case where the dataset is small or the learners need a lot of data, an alternative is to use cross-validation.

Cross-validation. The idea of T-fold cross-validation is to split the dataset into a training/validation set and a test set, then to further split the training/validation set into T folds.

1. Training: For each fold $t = 1, \ldots, T$
   - Use fold t as a validation set, and the rest as a training set to train each learner j; denote $\widehat{h}_j^{(d_{\text{train},t})}$ the jth learned prediction rule, and
   $$\widehat{R}^{(d_{\text{val},t})}(\widehat{h}_j^{(d_{\text{train},t})})$$
   its estimated risk
2. Validation:
   a) Choose the learner that minimises the estimated risk average over the folds
   $$\widehat{M} = \arg\min_j \frac{1}{T}\sum_{t=1}^{T} \widehat{R}^{(d_{\text{val},t})}(\widehat{h}_j^{(d_{\text{train},t})})$$
   b) Re-train the prediction rule on the whole training/validation set for the chosen learner $\widehat{M}$.
3. Test: Approximate the generalisation error with $\widehat{R}^{(d_{\text{test}})}(\widehat{h}^{(d_{\text{train/val}})})$.



A special case of cross-validation is **leave-one-out** cross-validation, where we use one data item per fold, that is T equals the number of observations in training/validation set. Cross-validation can be computationally intensive, as the computational cost is multiplied by the number of folds T.

3.6 Binary Classification: Performance evaluation and ROC curve

We focus here on the binary classification case. Let y be the true class, and $h(x)$ be the predicted class. The following table summarises the different configurations.

	$h(x) = -1$	$h(x) = 1$
$y = -1$	*true negative*	*false positive*
$y = 1$	*false negative*	*true positive*

The matrix counting the number of true/false positive/negative over a dataset d is called a **confusion matrix**. We denote TN, FN, FP, TP respectively the number of true negative, false negative, false positive and true positive over the dataset d for the classifier h. So far, as a measure of the quality of a classifier h, we have focused on the (population) risk under a 0-1 loss, which is the misclassification error, or error rate

$$R(h) = \Pr(Y \neq h(X)).$$

The empirical misclassification error is

$$\widehat{R}_n^{(d)}(h) = \frac{FN + FP}{FN + FP + TP + TN}.$$

For binary classification, other quantities are often considered to assess the performance of a classifier. We list below the population and empirical versions.

Name	Population	Empirical
Misclassification error/Error rate	$\Pr(Y \neq h(X))$	$(FP+FN)/(TP+TN+FP+FN)$
Accuracy (1-Error rate)	$\Pr(Y = h(X))$	$(TP+TN)/(TP+TN+FP+FN)$
Sensitivity/Recall/True positive rate	$\Pr(h(X) = 1 \mid Y = 1)$	$TP/(TP+FN)$
Specificity/True negative rate	$\Pr(h(X) = -1 \mid Y = -1)$	$TN/(TN+FP)$
False positive rate (1-specificity)	$\Pr(h(X) = 1 \mid Y = -1)$	$FP/(TN+FP)$
Precision	$\Pr(Y = 1 \mid h(X) = 1)$	$TP/(TP+FP)$

Weighted loss. Two possible types of error may occur:

- False positive ($y = -1, h(x) = 1$), also called Type I error,
- False negative ($y = 1, h(x) = -1$), also called Type II error.

The $0-1$ loss assigns an equal cost of 1 to each type of error

	$h(x) = -1$	$h(x) = 1$
$y = -1$	$L(y, h(x)) = 0$	$L(y, h(x)) = 1$
$y = 1$	$L(y, h(x)) = 1$	$L(y, h(x)) = 0$

In some cases, it is more sensible to assign different costs to the different types of error. For example, when predicting if a patient has a given disease, we may assign a higher cost to a false negative error than a false positive. We consider in this case a loss

$$L(y, h(x)) = a\mathbb{1}_{y=-1,h(x)=1} + b\mathbb{1}_{y=1,h(x)=-1}$$

where $a, b > 0$ are respectively the type I and type II cost. The Bayes classifier under this cost is (see problem sheet)

$$h_t^*(x) = \begin{cases} 1 & \text{if } \eta(x) \geq t := \frac{a}{a+b} \\ -1 & \text{otherwise} \end{cases}$$

where $\eta(x) = \Pr(Y = 1|X = x)$. Note that the Bayes classifier only depends on a and b through the ratio $t = a/(a + b)$, the value $t = 1/2$ corresponding to the Bayes classifier under the $0 - 1$ loss. A large value t will allow more false negative and less false positive; a small value allows for more false positive and less false negative.

ROC curve and AUC. For a given weighted loss with ratio t, let $\widehat{h}_t$ be some plug-in (generative or discriminative) classifier, defined by

$$\widehat{h}_t(x) = \begin{cases} 1 & \text{if } \widehat{\eta}(x) \geq t \\ -1 & \text{otherwise} \end{cases}$$

where $\widehat{\eta}(x)$ is the estimate of $\eta(x) = \Pr(Y = 1 \mid X = x)$, and does not depend on the choice of the threshold t.

How can we assess the performance of the family of plug-in classifiers $(\widehat{h}_t)$ for $t \in [0, 1]$? A classical strategy is the ROC curve and the Area Under the Curve (AUC). For a threshold $t \in [0, 1]$, denote $\beta(t) = \Pr(\widehat{h}_t(X) = 1 \mid Y = 1)$ the true positive rate (TPR) of $\widehat{h}_t$ and $\alpha(t) = \Pr(\widehat{h}_t(X) = 1 \mid Y = -1)$ its false positive rate (FPR). α and β are monotone functions of t, with $\alpha(t) = \beta(t) = 0$ if $t = 1$, and $\alpha(t) = \beta(t) = 1$ if $t = 0$. A good classifier will have a large true positive rate for a small false positive rate. To quantify this over a range of values, the ROC curve plots the TPR $\beta(t)$ vs the FPR $\alpha(t)$ for t ranging from 1 to 0. The area under the ROC curve, called AUC and defined as

$$AUC = \int_1^0 \beta(t) d\alpha(t),$$

is used as a measure of the performance of the family of plug-in classifiers over the whole range of loss functions. This is illustrated in Figure 3.14. In practice the population TPR and FPR cannot be computed analytically, and one reports their empirical approximation over a test set.

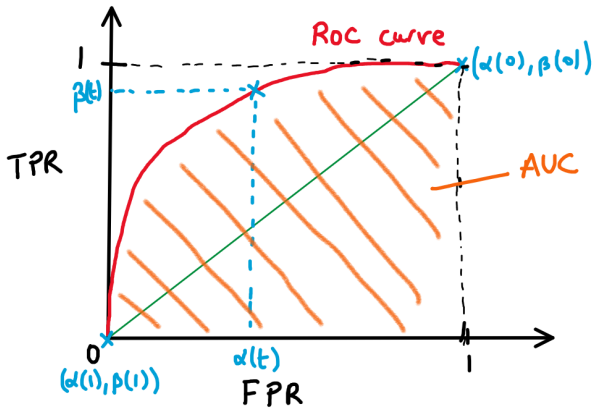


Figure 3.14: Illustration of the ROC curve and Area Under the Curve. Each point of the ROC curve corresponds to the value of the False positive rate vs the true positive rate of a plug-in classifier for a threshold t.

Probabilistic Interpretation of the AUC. Consider a plug-in classifier with estimate $\widehat{\eta}(x)$, obtained from a dataset d, which is a realisation from a random sample D. For two random variables $(\widetilde{X}_0, \widetilde{Y}_0)$ and $(\widetilde{X}_1, \widetilde{Y}_1)$, mutually independent and independent from D, with the same distribution as (X, Y), we have

$$AUC = \Pr\left(\widehat{\eta}(\widetilde{X}_1) \geq \widehat{\eta}(\widetilde{X}_0) \mid \widetilde{Y}_1 = 1, \widetilde{Y}_0 = -1, D = d\right).$$

That is the AUC is the probability that, for two independent samples from class 1 and -1, the discriminant score of the sample from class 1 is greater than the discriminant score of the sample from class -1.

Proof. The conditioning on $D = d$ is implicit in what follows. Denote $F_{-1}(s)$ the cdf of the random variable $\widehat{\eta}(X) \in [0, 1]$ given $Y = -1$, and $F_1(s)$ the cdf of $\widehat{\eta}(X)$ given $Y = 1$:

$$F_1(s) = \Pr(\widehat{\eta}(X) \leq s \mid Y = 1)$$
$$F_{-1}(s) = \Pr(\widehat{\eta}(X) \leq s \mid Y = -1).$$

Let $f_1 = F_1'$ and $f_{-1} = F_{-1}'$ be their pdf. For a threshold $t = a/(a + b)$, both the (population) true positive rate $\beta(t)$ and false positive rate $\alpha(t)$ of the plug-in classifier $\widehat{h}_t$ can be expressed as

$$\beta(t) = \Pr(\widehat{\eta}(X) \geq t \mid Y = 1) = 1 - F_1(t) \tag{3.11}$$
$$\alpha(t) = \Pr(\widehat{\eta}(X) \geq t \mid Y = -1) = 1 - F_{-1}(t) \tag{3.12}$$

The Area Under the ROC Curve, denoted AUC is

$$\begin{aligned}\int_1^0 \beta(t) d\alpha(t) = \int_0^1 (1 - F_1(t)) f_{-1}(t) dt &= \mathbb{E}\left[(1 - F_1(\widehat{\eta}(\widetilde{X}_0))) \mid \widetilde{Y}_0 = -1\right] \\ &= \mathbb{E}\left[\Pr\left(\widehat{\eta}(\widetilde{X}_1) \geq \widehat{\eta}(\widetilde{X}_0) \mid \widetilde{Y}_1 = 1, \widetilde{X}_0\right) \mid \widetilde{Y}_0 = -1\right] \\ &= \Pr\left(\widehat{\eta}(\widetilde{X}_1) \geq \widehat{\eta}(\widetilde{X}_0) \mid \widetilde{Y}_1 = 1, \widetilde{Y}_0 = -1\right)\end{aligned}$$

where $(\widetilde{X}_0, \widetilde{Y}_0)$ and $(\widetilde{X}_1, \widetilde{Y}_1)$ are iid with the same distribution as (X, Y). □

4 — Optimisation for machine learning

Let $\theta \in \mathbb{R}^p$. Consider either

- ERM with a class of prediction rules h_θ parameterised by a parameter θ or
- a conditional plug-in method, where the conditional distribution $f_\theta(y|x)$ is parameterised by θ or
- a generative method, where the joint distribution $\pi_\theta(x, y)$ is parameterised by θ.

For the plug-in approach, we assume that the parameter θ is fitted using maximum likelihood, potentially with the addition of a regularisation term.
In all three cases, we aim at minimising an objective function $J : \mathbb{R}^p \to \mathbb{R}_+$ of the form

$$J(\theta) = \sum_{i=1}^{n} J_i(\theta) + J_0(\theta) \tag{4.1}$$

where J_0 is a penalisation/regularisation term and each term J_i is associated to an example (x_i, y_i):

- For empirical risk minimisation, $J_i(\theta) = \frac{1}{n} L(y_i, h_\theta(x_i))$
- For a conditional plug-in approach, $J_i(\theta) = \log(f_\theta(y_i|x_i))$
- For a generative plug-in approach, $J_i(\theta) = \log(\pi_\theta(x_i, y_i))$.

To keep the notations simple, in this chapter, we consider an ERM objective function, but the algorithms apply equally to estimate the parameters of plug-in methods. For ERM, the objective function J is therefore of the form

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, h_\theta(x_i)) + \frac{\lambda}{n} \operatorname{pen}(h_\theta). \tag{4.2}$$

Assuming it exists, the global minimum to (4.2) may not be available in closed-form and one has to resort to some numerical method. Even if it is available in closed-form, computing it may be impractical for a large number n of observations and/or a large number of parameters p.

For example, for empirical risk minimisation with the class of linear functions with input expansion $\phi : \mathcal{X} \to \mathbb{R}^p$, and L2 regularisation with regularisation parameter λ, the objective function is

$$J(\beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \phi(x_i)^\intercal \beta)^2 + \frac{\lambda}{n} \|\beta\|^2. \tag{4.3}$$

As shown in Section 3.4, the estimated parameters of the learned prediction rule are computed as

$$\widehat{\beta} = (\mathbf{\Phi}^\intercal \mathbf{\Phi} + \lambda I)^{-1} \mathbf{\Phi}^\intercal \mathbf{y}$$

where $\mathbf{\Phi}$ is an $n \times p$ matrix, $\mathbf{y}$ is a vector of length n, and $\mathbf{\Phi}^\intercal \mathbf{\Phi} + \lambda I$ is a $p \times p$ matrix. The number of computations is of order

- $O(np^2)$ for computing $\mathbf{\Phi}^\intercal \mathbf{\Phi} + \lambda I$ and $\mathbf{\Phi}^\intercal \mathbf{y}$.
- $O(p^3)$ (using e.g. Gauss-Jordan elimination) for solving the linear system

$$(\mathbf{\Phi}^\intercal \mathbf{\Phi} + \lambda I)\beta = \mathbf{\Phi}^\intercal \mathbf{y}\,. \tag{4.4}$$

This becomes impractical for large p and/or n. For other methods, such as logistic regression or neural networks, there is no closed-formed solution. Standard numerical approaches are gradient descent (GD) and stochastic gradient descent (SGD), which are reviewed in this chapter.

Notations. For a vector $\theta = (\theta_1, \ldots, \theta_p)^\intercal$, a twice differentiable function $J : \mathbb{R}^p \to \mathbb{R}_+$, let $\nabla_\theta J(\theta)$ be the gradient of J with respect to θ, and $\nabla^2_\theta J(\theta)$ be the Hessian of J with respect to θ, defined as

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{d\theta_1} \\ \cdots \\ \frac{\partial J(\theta)}{d\theta_p} \end{pmatrix}, \quad \nabla^2_\theta J(\theta) = \begin{pmatrix} \frac{\partial^2 J(\theta)}{d\theta_1 d\theta_1} & \cdots & \frac{\partial^2 J(\theta)}{d\theta_1 d\theta_p} \\ \cdots & \cdots & \cdots \\ \frac{\partial^2 J(\theta)}{d\theta_p d\theta_1} & \cdots & \frac{\partial^2 J(\theta)}{d\theta_p d\theta_p} \end{pmatrix}.$$

A symmetric real-valued p-by-p matrix H is said to be positive semi-definite, iff

$$z^\intercal H z \geq 0$$

for all $z \in \mathbb{R}^p$. We use the notation $H \succeq 0$ if H is positive semi-definite.

4.1 Convex functions

Let $J : \mathbb{R}^p \to \mathbb{R}$. We review a few definitions and basic properties of convex functions.

Definition 23 (Convex function)**.** *A function J is said to be convex if, for all $u, v \in \mathbb{R}^p$, $\alpha \in [0, 1]$,*

$$J(\alpha u + (1 - \alpha)v) \leq \alpha J(u) + (1 - \alpha)J(v)$$

Examples of convex and non-convex functions are given in Figure 4.1. Here are some examples of convex functions:

- Univariate: θ^2, $\exp(-\theta)$, $\log(1 + \exp(-\theta))$, $\max(0, 1 - \theta)$
- Affine functions: $A\theta + b$
- Quadratic functions: $\theta^\intercal H\theta$ where $H \succeq 0$

A key property of convex functions is the following, which induces that the convergence of an algorithm to a local minimum implies the convergence to a global minimum.

Proposition 24. *If the function J is convex, all local minima are also global minima.*

For differentiable and twice differentiable convex functions, we have the following characterisations and properties.

Definition 25 (Differentiable convex function)**.** *A differentiable function J is convex iff, for any u, v*

$$J(u) \geq J(v) + \nabla J(v)^\intercal (u - v)$$

That is, the function is above the tangent at v.

Proposition 26. *Let J be a differentiable convex function. Any stationary point u of J, that is such that*

$$\nabla_u J(u) = 0$$

is also a global minimum.

Definition 27 (Twice differentiable convex function)**.** *A twice differentiable function J is convex iff*

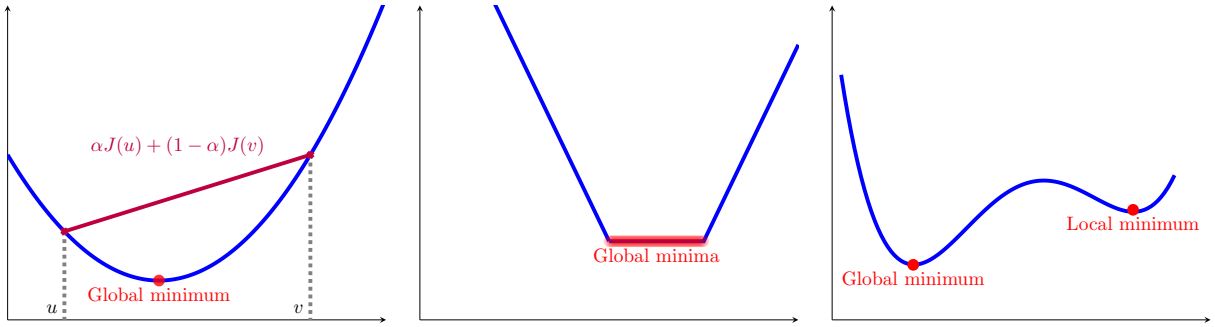
$$\nabla^2_u J(u) \succeq 0$$

for all $u \in \mathbb{R}^p$.

Convex functions can be combined in a number of ways:

- **Nonnegative linear combination of convex functions is convex.** Let J_1 and J_2 be convex functions. Then
$$J = \alpha_1 J_1 + \alpha_2 J_2$$
is also convex for any $\alpha_1, \alpha_2 \geq 0$.
- **Affine composition of convex functions is convex**. If g is convex, then $J(\theta) = g(A\theta + b)$ is convex.

It follows from the first property above that, if for any (x_i, y_i), the function $J_i : \theta \to L(y_i, h_\theta(x_i))$ is a convex function (of θ) and the regularisation function pen is convex, then the objective function (4.2) is also a convex function, and all local minima are global minima. This is the case for ridge regression, as shown below. The objective functions of logistic regression and of the other linear classifiers discussed in Chapter 5 are also all convex. Other popular machine learning methods, such as support vector machines (not covered in this course) also have a convex objective function. Note that the composition of convex functions is in general non-convex, and the objective function of (deep) neural networks, covered later in the course, is non-convex.



(a) Convex, differentiable function, with a unique minimum

(b) Convex, not differentiable everywhere, function with a non-unique global minimum

(c) Non-convex function

Figure 4.1: Examples of convex and non-convex functions.

Ridge regression. The objective function (4.3) is convex. The gradient and Hessian are

$$\nabla_\beta J(\beta) = \frac{1}{n}\left(2\mathbf{\Phi}^\intercal\mathbf{\Phi}\beta - 2\Phi^\intercal\,\mathbf{y} + 2\lambda\beta\right)$$
$$\nabla^2_\beta J(\beta) = \frac{2}{n}\left(\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda I\right).$$

In this case, the Hessian does not depend on the parameter β, and is positive semi-definite for any $\lambda \geq 0$.

4.2 Gradient Descent

Consider the objective function (4.2). Denote

$$\nabla_\theta J(\theta) = \frac{1}{n}\left(\sum_{i=1}^{n} \nabla_\theta L(y_i, h_\theta(x_i)) + \lambda\nabla_\theta \operatorname{pen}(h_\theta)\right) \tag{4.5}$$

the gradient of J with respect to θ. Let $\eta > 0$. The gradient descent algorithm proceeds as follows.

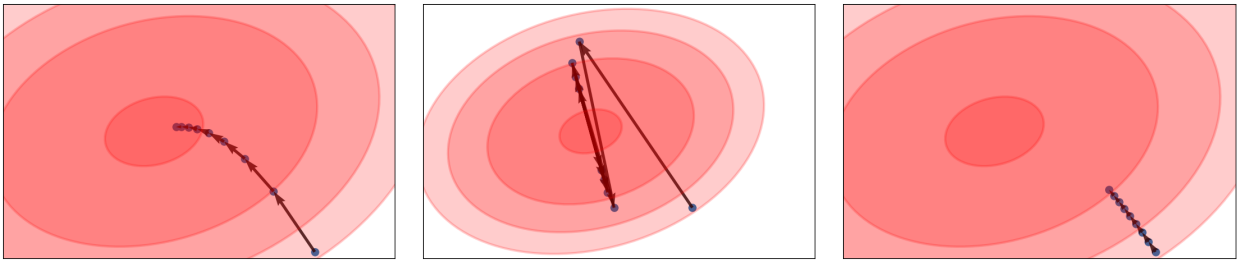
Algorithm 2 Gradient Descent

- Initialise $\theta^{(0)}$ and set $t = 0$;
- Repeat until convergence
  1. Compute the gradient $\nabla_\theta J(\theta^{(t)})$
  2. Update the parameters
  $$\theta^{(t+1)} = \theta^{(t)} - \eta\nabla_\theta J(\theta^{(t)})$$
  3. Set $t \leftarrow t + 1$

An illustration of the iterations of the gradient descent algorithm is given in Figure 4.2(a). The negative gradient is going in the direction that decreases the function J. Large values of the gradient will induce large changes in the value of the parameter. As the parameter becomes closer to a stationary point of J, the gradient takes smaller values, and the parameter changes more slowly. For convex functions (under some additional assumptions omitted here), the algorithm will stop at a point close to the minimum solution, independent of the starting point. For non-convex functions, the algorithm will attain different local minima depending on the initialisation.

Learning rate. The parameter η is known as the **step size**, or **learning rate**. Its value is critical to the good behaviour of the gradient descent algorithm. Too large values may lead the algorithm to have a zigzaguing behaviour, or even to diverge, whereas too small values will lead to slow convergence. See the figure 4.2 for an illustration. This parameter can also be updated adaptively.

Interpretation of gradient descent via a quadratic approximation, and Newton-Raphson algorithm. Assume that J is twice differentiable. Using a second-order Taylor expansion of the objective function



(a) Adequate step-size η (b) Too large η (c) Too small η

Figure 4.2: Illustration of the first iterations of the gradient descent algorithm for different step-sizes η. Ellipses represent contour of constant value of the objective function.

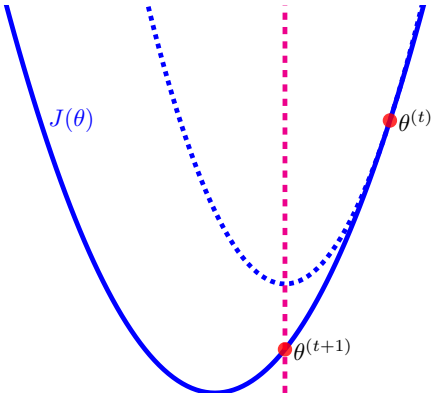


Figure 4.3: Gradient descent as minimisation of a quadratic function.

J around a point $\theta \in \mathbb{R}^p$, we have, for $\delta \in \mathbb{R}^p$,

$$J(\theta + \delta) \simeq J(\theta) + \nabla_\theta J(\theta)^\intercal \delta + \frac{1}{2}\delta^\intercal \nabla_\theta^2 J(\theta)\delta. \tag{4.6}$$

Replacing the Hessian matrix $\nabla_\theta^2 J(\theta)$ with the diagonal p-by-p matrix $\frac{1}{\eta}I$ gives the approximation

$$J(\theta + \delta) \simeq J(\theta) + \nabla_\theta J(\theta)^\intercal \delta + \frac{1}{2\eta}\|\delta\|^2. \tag{4.7}$$

At a given point θ, we use the above quadratic function as an approximation to $J(\theta + \delta)$. Taking the derivative with respect to δ, we obtain

$$\nabla_\theta J(\theta) + \frac{1}{\eta}\delta = 0,$$

hence the minimum is achieved for $\delta = -\eta \nabla_\theta J(\theta)$, which corresponds to the gradient descent update. See Figure 4.3 for an illustration. Instead of taking approximating the Hessian, we can use directly the second-order approximation (4.6). Differentiating with respect to δ and setting to 0, we obtain

$$\nabla_\theta J(\theta) + \nabla_\theta^2 J(\theta)\delta = 0$$

hence a minimum at

$$\delta = -(\nabla_\theta^2 J(\theta))^{-1} \nabla_\theta J(\theta).$$

The corresponding update is therefore $\theta^{(t+1)} = \theta^{(t)} - (\nabla_\theta^2 J(\theta^{(t)})^{-1} \nabla_\theta J(\theta^{(t)})$, and the iterative algorithm using this update is known as the **Newton-Raphson** gradient descent algorithm. Note that the Hessian is a p-by-p matrix, which may be computationally too expensive to compute for large p.

Gradient descent for linear regression. For instance, for ordinary linear regression (no regularisation), we have

$$nJ(\beta) = (\mathbf{y} - \Phi\beta)^\intercal (\mathbf{y} - \Phi\beta). \tag{4.8}$$

Algorithm 3 Newton-Raphson algorithm

- Initialise $\theta^{(0)}$ and set $t = 0$;
- Repeat until convergence
  1. Update the parameters
  $$\theta^{(t+1)} = \theta^{(t)} - (\nabla^2_\theta J(\theta^{(t)}))^{-1} \nabla_\theta J(\theta^{(t)})$$
  2. Set $t \leftarrow t + 1$

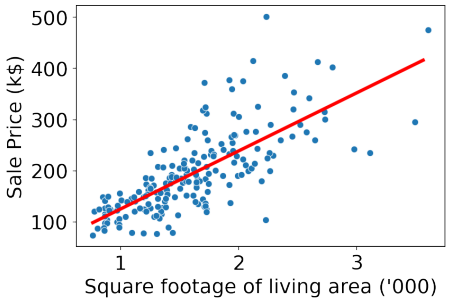


Figure 4.4: House pricing data and learned prediction rule.

The gradient is expressed as

$$n\nabla_\beta J(\beta) = 2\Phi^\intercal(\Phi\beta - \mathbf{y})$$

leading to the parameter update

$$\beta^{(t+1)} = \beta^{(t)} - \frac{2\eta}{n}\Phi^\intercal(\Phi\beta^{(t)} - \mathbf{y}).$$

(non examinable) Each iteration has a computational cost of $O(np)$. If T, the total number of gradient descent steps, is small compared to the number of parameters p, the overall computational cost of the gradient descent algorithm is lower than that of solving the linear system (4.4).

Input/Feature normalisation. We can rewrite the gradient step a bit differently to gain some insights on the gradient descent update, and on its limitations:

$$\begin{aligned}\beta^{(t+1)} &= \beta^{(t)} - \frac{2\eta}{n}(\Phi^\intercal\Phi\beta^{(t)} - \Phi^\intercal\mathbf{y}) \\ &= \beta^{(t)} + \frac{2\eta}{n}\Phi^\intercal\Phi(\widehat{\beta} - \beta^{(t)})\end{aligned}$$

where

$$\widehat{\beta} = (\mathbf{\Phi}^\intercal\mathbf{\Phi})^{-1}\mathbf{\Phi}^\intercal\mathbf{y}$$

is the empirical risk minimiser. If $\Phi^\intercal\Phi = c \times I$ for some constant $c > 0$, then a gradient descent update will go in the direction of $\widehat{\beta} - \beta^{(t)}$, and will converge quickly to $\widehat{\beta}$. Otherwise, the update will have a zigzaguing behaviour. A way to have a better conditioned problem is to apply some transformation to the data. Standardising the different dimensions of the transformed inputs $\phi(x_i)$ so that they have zero mean and unit variance for example leads to a more spherical $\Phi^\intercal\Phi$ and a faster convergence of the GD algorithm. This is illustrated in the example below.

Example 28. *We consider again the house sale prices dataset introduced in Section 1.4.3 ($n = 200, p = 1$). We first consider a linear prediction rule with transformation $\phi(x_i) = (1\ x_i)^\intercal$. Let $\widehat{\beta} \in \mathbb{R}^2$ be the ordinary least square estimate. The learned prediction rule $\widehat{h}(x) = x^\intercal\widehat{\beta}$ is shown in Figure 4.4, together with the data. The estimate $\widehat{\beta}$ can be found in closed-form, but we are going to estimate it using gradient descent, starting from $\beta^{(0)} = (0, 0)^\intercal$. The first iterates, and the result after 200 iterations are represented in Figure 4.5. As can be seen, the contours of constant value of the empirical risk have en ellipsoid shape and the negative gradients do not point towards the minimum of the objective function. The algorithm takes a large number of iterations to converge, and is zigzaguing. Increasing or decreasing the learning rate η would not resolve this issue. We can obtain a much faster convergence by standardising the inputs. Consider now the transformed inputs $\phi(x_i) = (1\ ,\ \frac{x_i - \bar{x}}{\sigma})^\intercal$ where $\bar{x}$ and σ respectively denote the sample mean and standard deviation of $(x_1, \ldots, x_n)$. Starting from the same initialisation, the first iterates are given in Figure 4.6. The contours of the objective function, in this particular case, are now spherical, and the negative gradients point towards the minimum. Using a not too large learning rate, gradient descent converges in a few iterations.*

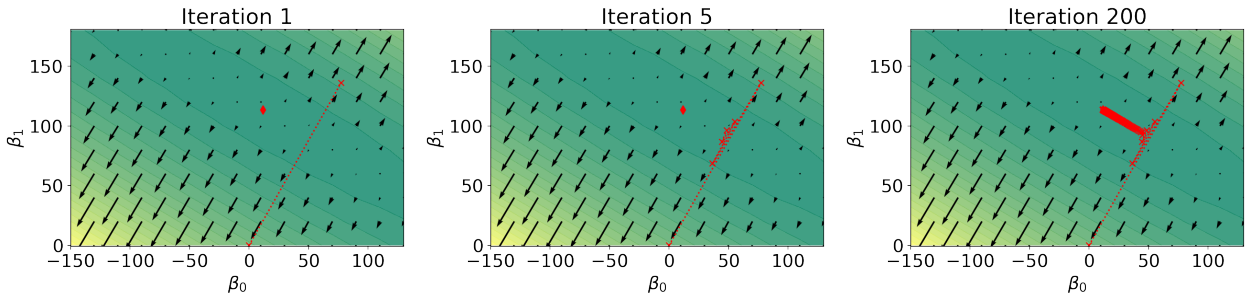


Figure 4.5: Contours of the objective function (unnormalised case), miminum of the objective function (red diamond), and gradient descent updates (crosses). The gradients are represented by arrows.

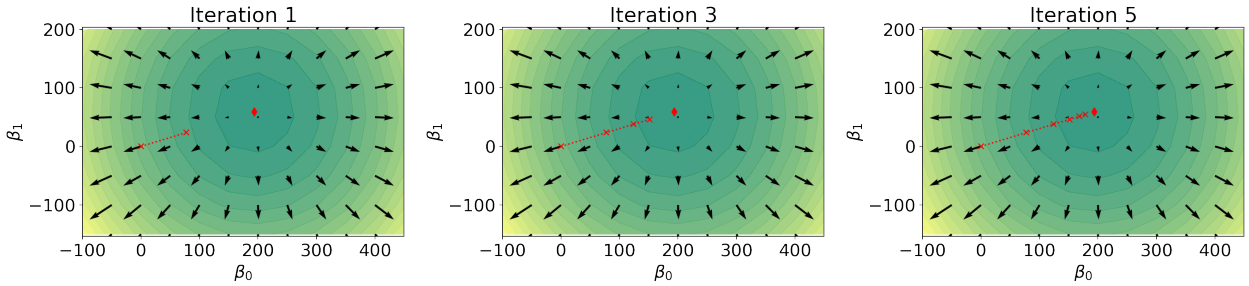


Figure 4.6: Contours of the objective function (normalised case), miminum of the objective function (red diamond), and gradient descent updates (crosses). The gradients are represented by black arrows.

4.3 Stochastic Gradient Descent

Stochastic gradient descent (and its variants) is certainly the most popular algorithm for large-scale (large n and/or p) machine learning. It is in particular the default algorithm to learn deep learning models.

Gradient descent requires to evaluate the gradient (4.5) at each iteration, which typically scales linearly in the number n of observations. This is prohibitive for very large datasets. Stochastic gradient descent replaces the gradient by an unbiased estimator of the gradient, obtained by using a subset of the data (called a **mini-batch**) at each iteration. Let $(\eta_t)_{t=0,1,\dots}$ be a monotone decreasing sequence, and $1 \leq n_b \leq n$ be the **mini-batch size**. The algorithm proceeds as follows.

Algorithm 4 Stochastic Gradient Descent

- Initialise $\theta^{(0)}$ and set $t = 0$;
- Repeat until convergence
  1. Randomly select n_b observations from the dataset d; denote them $(\widetilde{x}_i^{(t)}, \widetilde{y}_i^{(t)})_{i=1,\dots,n_b}$
  2. Compute the gradient estimate $\nabla_\theta \widetilde{J}^{(t)}(\theta^{(t)})$ where

$$\nabla_\theta \widetilde{J}^{(t)}(\theta) = \frac{1}{n_b} \sum_{i=1}^{n_b} \nabla_\theta L(\widetilde{y}_i^{(t)}, h_\theta(\widetilde{x}_i^{(t)})) + \frac{\lambda}{n} \nabla_\theta \operatorname{pen}(h_\theta)$$

  3. Update the parameters

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla_\theta \widetilde{J}^{(t)}(\theta^{(t)})$$

  4. Set $t \leftarrow t + 1$

For any θ, the mini-batch gradient $\nabla_\theta \widetilde{J}^{(t)}(\theta)$ is an **unbiased estimator** of the true gradient $\nabla_\theta J(\theta)$.

Proof. Note that, for any $1 \leq i \leq n_b$, $(\widetilde{X}_i^{(t)}, \widetilde{Y}_i^{(t)})$ is a uniform discrete random variable taking values in the set $\{(x_1, y_1), \dots, (x_n, y_n)\}$. Hence, for any $1 \leq i \leq n_b$ and $1 \leq j \leq n$,

$$\Pr((\widetilde{X}_i^{(t)}, \widetilde{Y}_i^{(t)}) = (x_j, y_j)) = \frac{1}{n}.$$

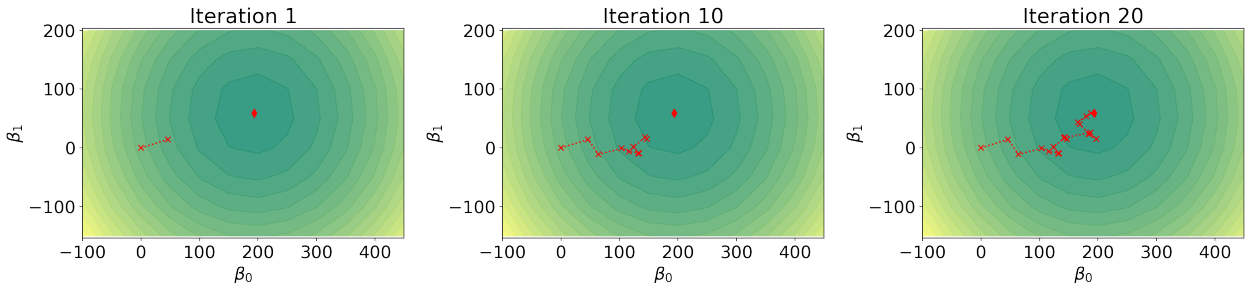


Figure 4.7: Illustration of the stochastic gradient descent algorithm. Contours of the objective function (normalised case), ERM (red diamond), and stochastic gradient descent updates (crosses).

It follows that, for any $1 \leq i \leq n_b$,

$$\mathbb{E}\left[L(\widetilde{Y}_i^{(t)}, h_\theta(\widetilde{X}_i^{(t)}))\right] = \frac{1}{n}\sum_{j=1}^{n} L(y_j, h_\theta(x_j)) = \widehat{R}_n(h_\theta)$$

and

$$\begin{aligned}
\mathbb{E}[\nabla_\theta \widetilde{J}^{(t)}(\theta)] &= \frac{1}{n_b}\sum_{i=1}^{n_b} \mathbb{E}\,\nabla_\theta L(\widetilde{Y}_i^{(t)}, h_\theta(\widetilde{X}_i^{(t)})) + \frac{\lambda}{n}\nabla_\theta \operatorname{pen}(h_\theta) \\
&= \frac{1}{n_b}\sum_{i=1}^{n_b} \nabla_\theta\, \mathbb{E}\, L(\widetilde{Y}_i^{(t)}, h_\theta(\widetilde{X}_i^{(t)})) + \frac{\lambda}{n}\nabla_\theta \operatorname{pen}(h_\theta) \\
&= \nabla_\theta \widehat{R}_n(h_\theta) + \frac{\lambda}{n}\nabla_\theta \operatorname{pen}(h_\theta) = \nabla_\theta J(\theta).
\end{aligned}$$

□

For SGD to converge, the learning rate η_t should be such that $\eta_t \to 0$. A standard parametrisation is

$$\eta_t = \frac{\eta_0}{(1 + t/t_0)^a}$$

where $\eta_0 > 0$, $t_0 > 0$ and $1/2 < a \leq 1$ are tuning parameters. Alternatively, the learning rate may be tuned adaptively.

Example 29. *An illustration of stochastic gradient descent, with a batch size of 1, is given in Figure 4.7. After 20 iterations (that is, using only 1/10 of the data), the algorithm has already converged close to the minimum.*

4.4 Early stopping in (stochastic) gradient descent and implicit regularisation

Consider using (stochastic) gradient descent to minimise the empirical risk $\widehat{R}_n(h_\theta)$ over a large class $\mathcal{H}$. As we have seen earlier, without regularisation, the ERM will have a large estimation error, hence a large risk. Early stopping is a strategy that allows to **implicitly regularise** by stopping the algorithm before it reaches convergence.

Assume we start with an initial prediction rule $h_{\theta^{(0)}}$ with small complexity (hence large bias). For example, take $\theta^{(0)} = 0$. Each step of the (stochastic) gradient descent tends to increase the complexity of the prediction rule, hence to decrease the bias and increase the variance. For small number of iterations t the bias part dominates, while for large t, the variance part dominates. One can find the optimal number of steps of the algorithm using a validation set. For each iteration $t = 0, 1, \ldots, t_{\max}$ of the algorithm, we approximate the risk on the validation set, and return the value $\theta^{(t)}$ that minimises the validation risk.

Example 30. *An illustration of early stopping 4.8 is given in Figure 4.8. On the house pricing dataset, we now fit a polynomial of order 12 (hence $\beta \in \mathbb{R}^{13}$). We split the dataset into a training, validation and test set. We run gradient descent on the training set, starting from the origin. As the number of iterations increases, the norm of the vector $\beta^{(t)}$ increases. In the first steps, its norm is very small and we are underfitting. For large*

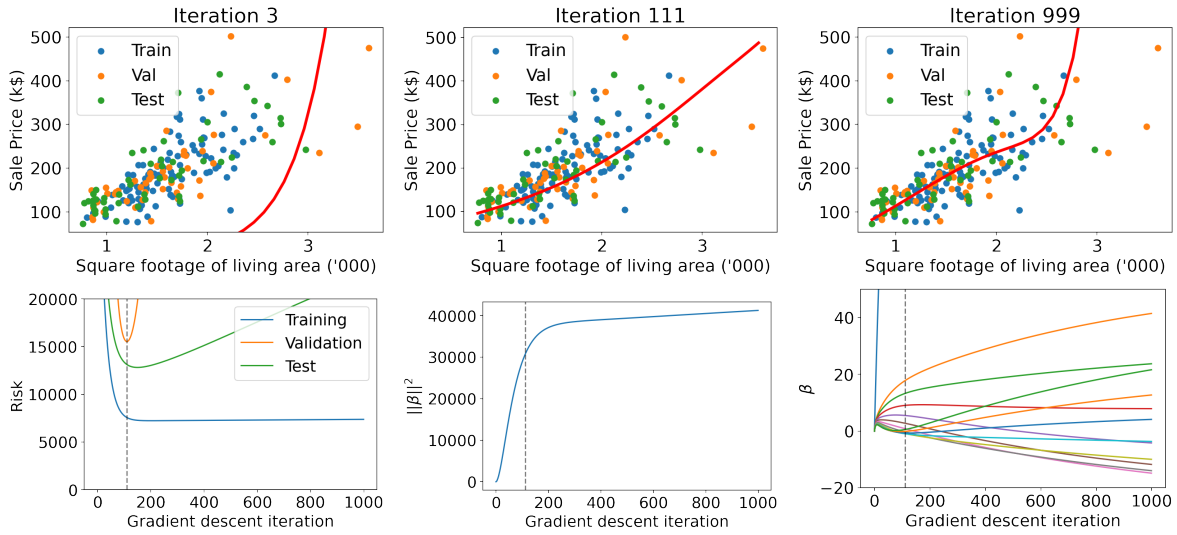


Figure 4.8: Illustration of early stopping and implicit regularisation in gradient descent. Top: Prediction rules at some iterations of the Gradient Descent algorithm. Bottom left: Training, empirical and test risk as a function of the gradient descent iteration. The dotted line corresponds to the iteration where the validation risk is minimised. Bottom middle: Squared norm of beta as a function of the GD iterations. Bottom right: Values of the different dimensions of the parameter β as a function of the number of GD iterations.

number of iterations, the iterate $\beta^{(t)}$ become closer to the minimum, and we start overfitting. The best value is chosen by looking at the empirical risk on the validation set, which is minimised at iteration 111. The learned prediction rule is represented in the top figure (middle).

5 — Linear classifiers

We focus here on binary classification, with $Y \in \{-1, 1\}$. Recall that, without loss of generality, a prediction rule can be expressed in terms of a discriminant function $f : \mathcal{X} \to \mathbb{R}$, such that $h(x) = 1$ if $f(x) \geq 0$ and -1 otherwise. Let $\phi : \mathcal{X} \to \mathbb{R}^p$ be some feature expansion function. Let $\mathcal{H}$ be the set of classifiers h with a linear discriminant function

$$f(x) = \phi(x)^\intercal \beta$$

where f is parameterised by the vector $\beta \in \mathbb{R}^p$. Denote $\mathcal{H}_f = \{f : f(x) = \phi(x)^\intercal \beta\}$ the set of linear discriminant functions on the transformed inputs $\phi(x)$. We have seen in Section 2.2 a generative classifier belonging to this class, the linear discriminant analysis classifier. We describe in this chapter three other linear classifiers.

A natural choice for a classifier would be the empirical risk minimiser under a 0-1 loss with

$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{y_i \neq \phi(x_i)^\intercal \beta}.$$

However, as we discuss in the next section, this classifier is impractical as one cannot usually learn its parameters.

5.1 Surrogate loss functions

The 0-1 loss for binary classification is defined as

$$L(y, h(x)) = \mathbb{1}_{y \neq h(x)} = \mathbb{1}_{\text{sign}(yf(x))=-1} = \psi_{0-1}(yf(x))$$

where $\psi_{0-1} : \mathbb{R} \to \{0, 1\}$ is such that $\psi_{0-1}(z) = \mathbb{1}_{\text{sign}(z)=-1}$. Under the ERM framework, we aim to find the minimiser $\widehat{f}$ of the empirical risk objective function

$$\frac{1}{n} \sum_{i=1}^{n} \psi_{0-1}(y_i f(x_i)) \tag{5.1}$$

over some class of functions (e.g. the class of linear functions). The function ψ_{0-1} is piecewise constant, non-convex, with a discontinuity at 0. The objective function 5.1 is therefore piecewise constant, non-convex and discontinuous, which makes the optimisation problem challenging, and prevents the use of gradient algorithms. To resolve this issue, we typically use an alternative loss function, called a **surrogate loss function**, which will make the objective function easier to minimise.

Definition 31. *A surrogate loss function is a continuous function such that*
1. *$\psi(x) \geq \psi_{0-1}(x)$ for all x*
2. *ψ is a convex function.*

The function ψ is also often chosen to be differentiable, although optimisation techniques exist for non-differentiable, convex functions. The ERM under the surrogate loss function ψ is defined as $\widehat{h}(x) = 1$ if $\widehat{f}(x) \geq 0$ and -1 otherwise, where

$$\widehat{f} = \arg\min_{f \in \mathcal{H}_f} \frac{1}{n} \sum_{i=1}^{n} \psi(y_i f(x_i)).$$

If $f(x) = \phi(x)^\intercal \beta$, this leads to the estimate

$$\widehat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \psi(y_i \phi(x_i)^\intercal \beta)$$

Here are some standard surrogate loss functions, which are plotted in Figure 5.1.

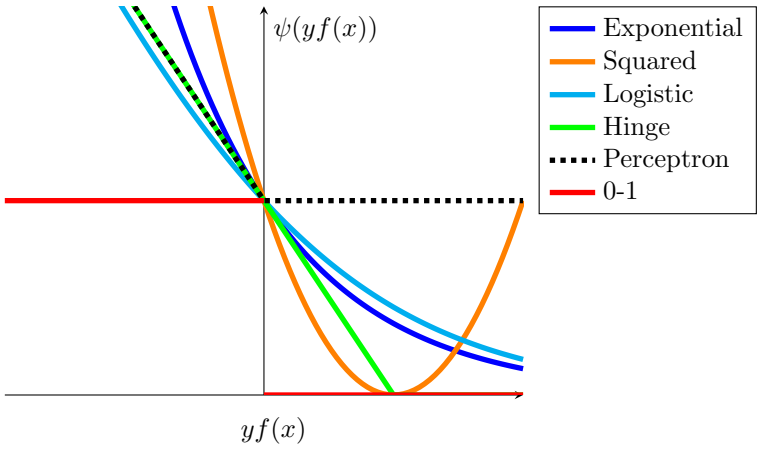


Figure 5.1: Surrogate loss functions for classification.

- Exponential: $\psi(z) = e^{-z}$
- Squared: $\psi(z) = (1-z)^2$
- Logistic: $\psi(z) = \log(1+e^{-z})/\log(2)$
- Perceptron: $\psi(z) = 1 + \max(0, -z)$
- Hinge: $\psi(z) = \max(0, 1-z)$

5.2 Least-squares classifier

Consider the squared surrogate loss function

$$\psi(z) = (1-z)^2. \tag{5.2}$$

The ERM under this surrogate loss and the class of linear classifiers $\mathcal{H}$ is given by

$$\begin{aligned}\widehat{\beta} &= \arg\min_{\beta\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n}(1-y_i\phi(x_i)^\intercal\beta)^2 \\ &= \arg\min_{\beta\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n} y_i^2(1-y_i\phi(x_i)^\intercal\beta)^2 \\ &= \arg\min_{\beta\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n}(y_i-\phi(x_i)^\intercal\beta)^2\end{aligned}$$

and $\widehat{\beta}$ is therefore the typical least-squares estimate

$$\widehat{\beta} = (\mathbf{\Phi}^\intercal\mathbf{\Phi})^{-1}\mathbf{\Phi}^\intercal\,\mathbf{y}\,.$$

This surrogate loss function has the advantage to lead to a closed-form solution. A drawback of the least-squares classifier is that the squared loss is a poor proxy for the 0-1 loss. As is apparent in Figure 5.1, it penalises well-classified points that are far from the boundary, as $\psi(z)$ takes large values for large positive z.

Remark 32. *When the number of examples in each class is the same, the least-squares binary classifier is the same as the binary LDA classifier (see Problem Sheets).*

5.3 Perceptron

The perceptron algorithm is a classic machine learning classifier algorithm introduced in 1962 by Rosenblatt. Consider the surrogate perceptron loss

$$\psi(z) = 1 + \max(0, -z). \tag{5.3}$$

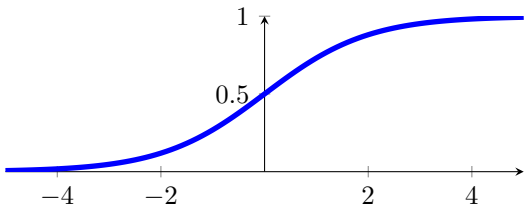


Figure 5.2: Sigmoid function.

This leads to the following objective function

$$J(\beta) = 1 + \frac{1}{n}\sum_{i=1}^{n} \max(0, -y_i \phi(x_i)^\intercal \beta). \tag{5.4}$$

The minimisation cannot be solved in closed form. The classic perceptron algorithm resorts to a stochastic gradient descent algorithm with a mini-batch size of 1, and a fixed step size η to find the minimiser of (5.4).

Algorithm 5 Perceptron algorithm

- Initialise $\beta^{(0)}$ and set $t = 0$;
- For $t = 1, \ldots, n$
  - If $y_t \phi(x_t)^\intercal \beta^{(t)} \geq 0$, set $\beta^{(t+1)} = \beta^{(t)}$;
  - otherwise, set
$$\beta^{(t+1)} = \beta^{(t)} + \eta y_t \phi(x_t)$$

Note that instead of randomly selecting a batch observation, it just runs over the dataset. The algorithm will converge if the transformed inputs are linearly separable; that is, there exists a function $h \in \mathcal{H}$ such that $\widehat{R}_n(h) = 0$.

5.4 Logistic regression

Logistic regression is another linear classifier. Like standard linear regression, this classifier can either be interpreted as a plug-in classifier, or as an empirical risk minimiser with a given surrogate function.

Logistic regression as a plug-in classifier. Assume that the conditional distribution of Y given $X = x$ takes the form

$$\Pr(Y = 1 \mid X = x) = \mathrm{sig}(\phi(x)^\intercal \beta)$$

and $\Pr(Y = -1 \mid X = x) = 1 - \Pr(Y = 1 \mid X = x) = \mathrm{sig}(-\phi(x)^\intercal \beta)$, where $\beta \in \mathbb{R}^p$ is an unknown parameter and

$$\mathrm{sig}(x) = \frac{1}{1 + e^{-x}} \tag{5.5}$$

is the sigmoid function[1], see Figure 5.2. An illustration of the logistic regression model for $x \in \mathbb{R}^2$ is given in Figure 5.4. Note that we have

$$\log \frac{\Pr(Y = 1 \mid X = x)}{\Pr(Y = -1 \mid X = x)} = \phi(x)^\intercal \beta$$

hence the Bayes classifier under this model is linear and can be written as

$$h^*(x) = \begin{cases} 1 & \text{if } f(x) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

where $f(x) = \phi(x)^\intercal \beta \geq 0$ is the linear discriminant function. The (linear) plug-in classifier is therefore given by

$$\widehat{h}(x) = \begin{cases} 1 & \text{if } \phi(x)^\intercal \widehat{\beta} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

[1] It is also called the logistic function. Not to be confused with the logistic loss function.

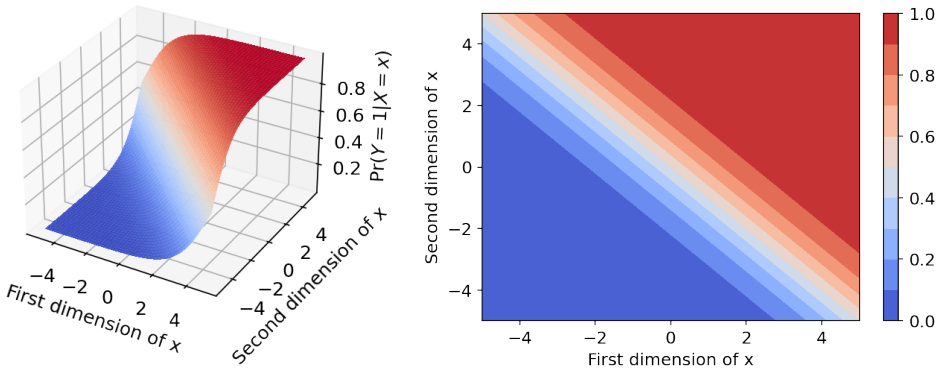


Figure 5.3: Illustration of the logistic regression model, for $x \in \mathbb{R}^2$, $\phi(x) = x$ and $\beta = (1\ 1)^\intercal$.

where $\widehat{\beta}$ is some estimate of the parameter β. For instance, one can use a maximum likelihood estimate. The log-likelihood is

$$\begin{aligned}\ell(\beta) &= \sum_{i=1}^{n} \log \operatorname{sig}(y_i \phi(x_i)^\intercal \beta) \\ &= -\sum_{i=1}^{n} \log(1 + e^{-y_i \phi(x_i)^\intercal \beta})\end{aligned}$$

The MLE is therefore

$$\widehat{\beta} = \arg\max_{\beta \in \mathbb{R}^p} -\sum_{i=1}^{n} \log(1 + e^{-y_i \phi(x_i)^\intercal \beta}). \tag{5.6}$$

Logistic regression as a ERM under a surrogate loss function. The above MLE can be written as

$$\widehat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \frac{1}{n \log(2)} \sum_{i=1}^{n} \log(1 + e^{-y_i \phi(x_i)^\intercal \beta})$$

hence it can also be interpreted as the ERM under the surrogate logistic loss $\psi(z) = -\log(\operatorname{sig}(z))/\log(2) = \log(1 + e^{-z})/\log(2)$ amongst the class $\mathcal{H}$ of linear classifiers.

Parameter estimation. Let (dropping the $\log 2$ factor)

$$J(\beta) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y_i \phi(x_i)^\intercal \beta}) = -\frac{1}{n} \sum_{i=1}^{n} \log \operatorname{sig}(y_i \phi(x_i)^\intercal \beta)$$

be the objective function to minimise. First note the following properties of the sigmoid function:

$$\begin{aligned}\operatorname{sig}(-z) &= 1 - \operatorname{sig}(z) \\ \frac{\partial \operatorname{sig}(z)}{dz} &= \operatorname{sig}(z) \operatorname{sig}(-z) \\ \frac{\partial \log \operatorname{sig}(z)}{dz} &= \operatorname{sig}(-z) \\ \frac{\partial^2 \log \operatorname{sig}(z)}{dz^2} &= -\operatorname{sig}(z) \operatorname{sig}(-z)\end{aligned}$$

J is differentiable, with gradient and Hessian

$$\nabla_\beta J(\beta) = -\frac{1}{n} \sum_{i=1}^{n} y_i \phi(x_i) \operatorname{sig}(-y_i \phi(x_i)^\intercal \beta) \tag{5.7}$$

$$\nabla_\beta^2 J(\beta) = \frac{1}{n} \sum_{i=1}^{n} \operatorname{sig}(y_i \phi(x_i)^\intercal \beta) \operatorname{sig}(-y_i \phi(x_i)^\intercal \beta) \phi(x_i) \phi(x_i)^\intercal \succeq 0 \tag{5.8}$$

We cannot solve $\nabla_\beta J(\beta) = 0$ as there is no closed form solution, and we need to resort to numerical methods. The Hessian matrix $H(\beta) = \nabla^2_\beta J(\beta)$ is positive semi-definite, and the objective function J is therefore **convex**. The objective function J can be minimised using e.g.

- Gradient descent, with update
$$\beta^{(t+1)} = \beta^{(t)} + \frac{\eta}{n}\sum_{i=1}^{n} y_i\phi(x_i)\,\mathrm{sig}(-y_i\phi(x_i)^\intercal\beta^{(t)})$$
where η is the learning rate.
- Stochastic gradient descent, with update
$$\beta^{(t+1)} = \beta^{(t)} + \frac{\eta_t}{n_b}\sum_{i=1}^{n_b} \widetilde{y}_i^{(t)}\phi(\widetilde{x}_i^{(t)})\,\mathrm{sig}(-\widetilde{y}_i^{(t)}\phi(\widetilde{x}_i^{(t)})^\intercal\beta^{(t)})$$
where $(\widetilde{x}_i^{(t)}, \widetilde{y}_i^{(t)})$ is the tth mini-batch of size n_b and (η_t) is a decreasing sequence of learning rates.
- Newton-Raphson algorithm (known as iterative reweighted least square (IRLS) in this case), with update
$$\beta^{(t+1)} = \beta^{(t)} - (\nabla^2_\beta J(\beta^{(t)}))^{-1}\nabla_\beta J(\beta^{(t)}).$$

Details on the implementation of iterative reweighted least squares. We can write the gradient and Hessian in a more compact form. Let $\mu \in \mathbb{R}^n$ be such that $\mu_i = \mathrm{sig}(\phi(x_i)^\intercal\beta)$ and S be a n-by-n diagonal entries with entries $\mu_i(1-\mu_i)$. Let c be the vector such that $c_i = \mathbb{1}_{y_i=+1}$. Then

$$\begin{aligned}
n\nabla_\beta J(\beta) &= -\sum_{i=1}^{n}\frac{y_i\phi(x_i)e^{-y_i\phi(x_i)^\intercal\beta}}{1+e^{-y_i\phi(x_i)^\intercal\beta}}\\
&= \sum_{i=1}^{n}\phi(x_i)(\mu_i - c_i)\\
&= \boldsymbol{\Phi}^\intercal(\mu - c)\\
n\nabla^2_\beta J(\beta) &= \sum_{i=1}^{n}\mathrm{sig}(y_i\phi(x_i)^\intercal\beta)\,\mathrm{sig}(-y_i\phi(x_i)^\intercal\beta)\phi(x_i)\phi(x_i)^\intercal\\
&= \boldsymbol{\Phi}^\intercal S\boldsymbol{\Phi}
\end{aligned}$$

Let $\beta^{(t)}$ be the parameter value after t iterations of the Newton-Raphson algorithm. Let $\mu^{(t)}$ and $S^{(t)}$ be the corresponding vectors and matrices. The Newton-Raphson update is

$$\begin{aligned}
\beta^{(t+1)} &= \beta^{(t)} - (\nabla^2_\beta J(\beta^{(t)}))^{-1}\nabla_\beta J(\beta^{(t)})\\
&= \beta^{(t)} + (\boldsymbol{\Phi}^\intercal S^{(t)}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\intercal(c-\mu^{(t)})\\
&= (\boldsymbol{\Phi}^\intercal S^{(t)}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\intercal S^{(t)}(\boldsymbol{\Phi}\beta^{(t)} + (S^{(t)})^{-1}(c-\mu^{(t)}))\\
&= (\boldsymbol{\Phi}^\intercal S^{(t)}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\intercal S^{(t)}z^{(t)}
\end{aligned}$$

where $z^{(t)} = \boldsymbol{\Phi}\beta^{(t)} + (S^{(t)})^{-1}(c-\mu^{(t)})$. Then $\beta^{(t+1)}$ is a solution of the **weighted least squares** problem

$$\begin{aligned}
\beta^{(t+1)} &= \arg\min_{\beta\in\mathbb{R}^p}\sum_{i=1}^{n} S^{(t)}_{ii}(z_i^{(t)} - \phi(x_i)^\intercal\beta)^2\\
&= \arg\min_{\beta\in\mathbb{R}^p}(z^{(t)} - \boldsymbol{\Phi}\beta)^\intercal S^{(t)}(z^{(t)} - \boldsymbol{\Phi}\beta).
\end{aligned}$$

Each iteration of the Newton-Raphson algorithm is therefore solving a reweighted least squares problem, hence the name iterative reweighted least square algorithm.

Linearly separable data. Assume that the data are linearly separable. That is, there exists a vector $\widetilde{\beta}$ such that $y_i(\phi(x_i)^\intercal\widetilde{\beta}) > 0$ for $i = 1,\ldots,n$. For any $c > 0$, the objective function for $c\widetilde{\beta}$ is

$$J(c\widetilde{\beta}) = \frac{1}{n}\sum_{i=1}^{n}\log(1+e^{-cy_i\phi(x)^\intercal\widetilde{\beta}})$$

which can be made arbitrarily close to 0 as $c \to \infty$. The associated estimated discriminant function $\widehat{f}(x) = \lim_{c\to\infty} c\phi(x)^\intercal\widetilde{\beta}$ is therefore equal to $+/-\infty$, depending on the sign of $\phi(x)^\intercal\widetilde{\beta}$. Similarly, the plug-in estimator of $\eta(x) = \Pr(Y = 1 \mid X = x)$ is $\widehat{\eta}(x) = 1$ or 0, yielding overconfident class predictions. One way to address this problem is to add a regularisation term to the objective function.

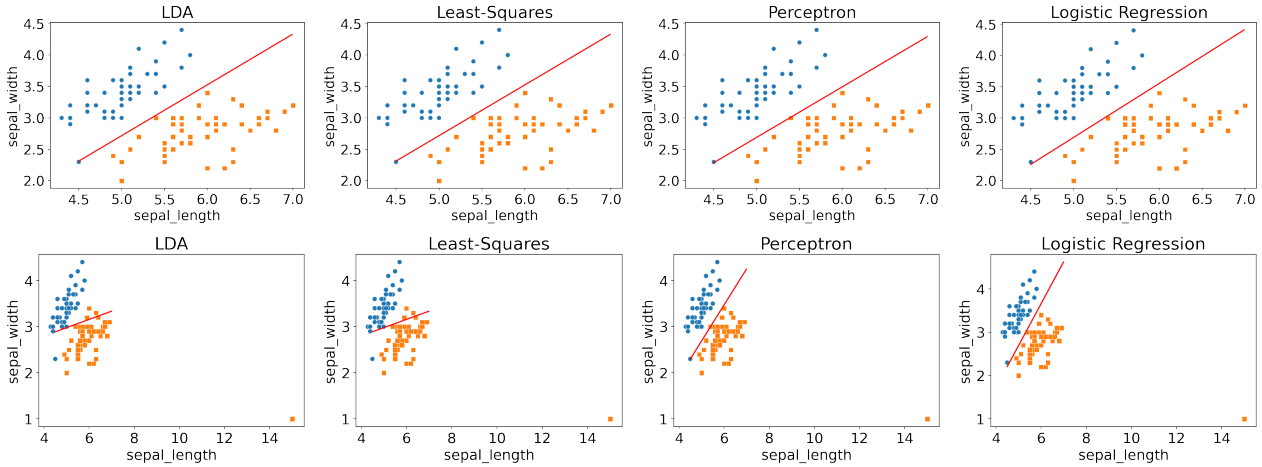


Figure 5.4: Top: Decision boundaries for 4 different linear classifiers on the iris data (2D, sepal length and width), with two classes (setosa and versicolor). Bottom: Same as above, but with the addition of an outlier far from the decision boundary.

Logistic regression for multi-class classification. Let $Y \in \{1, \ldots, K\}$. Multi-class logistic regression uses the **softmax function** to model the conditional class probabilities. For $k = 1, \ldots, K$

$$\Pr(Y = k \mid X = x) = \frac{\exp(\phi(x)^\intercal \beta_k)}{\sum_{k'=1}^{K} \exp(\phi(x)^\intercal \beta_{k'})}$$

where $\beta_k \in \mathbb{R}^p$ for $k = 1, \ldots, K$. This yields linear decision boundaries as

$$\log \frac{\Pr(Y = k \mid X = x)}{\Pr(Y = \ell \mid X = x)} = \phi(x)^\intercal (\beta_k - \beta_\ell).$$

5.5 Examples

5.5.1 Binary classification on the Iris data

Consider first the iris dataset, studied in Section 2.2. We consider the $n = 100$ observations from the classes setosa and versicolor, and consider only the sepal length and sepal width. The classification boundaries of linear discriminant analysis, logistic regression, perceptron and least-squares are shown in Figure 5.4(top). Note that as the classes have the same number of elements, the least-squares classifier and LDA are equivalent. In this example, the data are linearly separable, and all methods give similar results.

To emphasise the differences between the different methods, consider that we have an additional data example from the class versicolor, far from the decision boundary. As shown in Figure 5.4(bottom), the presence of this outlier has a large influence on the decision boundary for LDA/least-squares. For both the perceptron and logistic regression, the outlier has almost no influence on the decision boundary.

5.5.2 Multiclass classification on the Crabs data

In this example, we consider the Crabs dataset ($K = 4$ classes). We first extract the first two LDA discriminant coordinates of each example, denoted $z_1, \ldots, z_n$, and use these coordinates as our input data. We first consider a simple linear model, that is a transformation function $\phi(z_i) = (1 \; z_{i1} \; z_{i2})^\intercal$. The learned linear prediction rules for LDA and logistic regression are shown in Figure 5.5(left-middle). Next, we consider an input expansion that includes interaction terms. That is, we set $\phi(z_i) = (1 \; z_{i1} \; z_{i2} \; z_i 1 z_{i2})^\intercal$. The learned prediction rule, which now has nonlinear decision boundaries in the 2D input space, is represented in Figure 5.5(right).

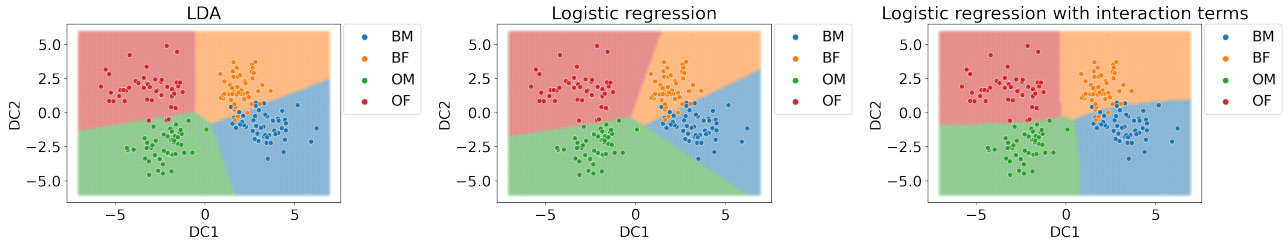


Figure 5.5: Multiclass classification on the Crabs dataset (2D projections), using LDA, logistic regression and logistic regression with interaction terms. The data are represented by dots.