Imperial College
London

LECTURE NOTES

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Mathematics for Machine Learning

Authors:
Marc Deisenroth, Stefanos Zafeiriou

Version: November 7, 2017

Contents

Introduction

These lecture notes support the course “Mathematics for Machine Learning” in the Department of Computing at Imperial College London. The aim of the course is to provide students the basic mathematical background and skills necessary to understand, design and implement modern statistical machine learning methodologies and inference mechanisms. The course will focus on examples regarding the use of mathematical tools for the design of basic machine learning and inference methodologies, such as Principal Component Analysis (PCA), Linear Discriminant Analysis, Bayesian Linear Regression and Support Vector Machines (SVMs).
This course is a hard pre-requisite for the following courses:

- CO-424H: Learning in Autonomous Systems
- CO-433: Advanced Robotics
- CO-493: Data Analysis and Probabilistic Inference[1]
- CO-495: Advanced Statistical Machine Learning and Pattern Recognition

and a soft pre-requisite for CO-477.
Relevant for the exam are

- The lecture notes
- The following chapters from the book by Bishop (2006):
  - Chapter 1
  - Chapter 2.2–2.3
  - Chapter 3
  - Chapter 8.1
  - Chapter 12.1–12.2
- The following chapters from the book by Golub and Van Loan (2012):
  - Chapter 1: Matrix Multiplication (without 1.5 and 1.6)
  - Chapter 2: Matrix Analysis (without 2.6 and 2.7)

[1]MSc Computing Science students are exempt from this hard constraint, but they are required to know the material of this course.

  - Chapter 5: 5.2 The QR Factorisation using only the Gram-Schmidt process

- The papers by Turk and Pentland (1991); Belhumeur et al. (1997)

- The following tutorials:

  - Tutorial by Burges (1998) until Section 4.4
  - From the tutorial by Lin (2006) only the basic topics.

For part two and three the relevant literature is also the lecture notes, as well as the following chapters from the book by Golub and Van Loan (2012):

- Chapter 1: Matrix Multiplication (without 1.5 and 1.6)

- Chapter 2: Matrix Analysis (without 2.6 and 2.7)

- Chapter 5: 5.2 The QR Factorisation using only the Gram-Schmidt process

As well as, the following papers Turk and Pentland (1991); Belhumeur et al. (1997) and tutorials Burges (1998); Lin (2006):

- Tutorial Burges (1998) until Section 4.4

- From Tutorial Lin (2006) only the basic topics.

Chapter 1

Linear Regression

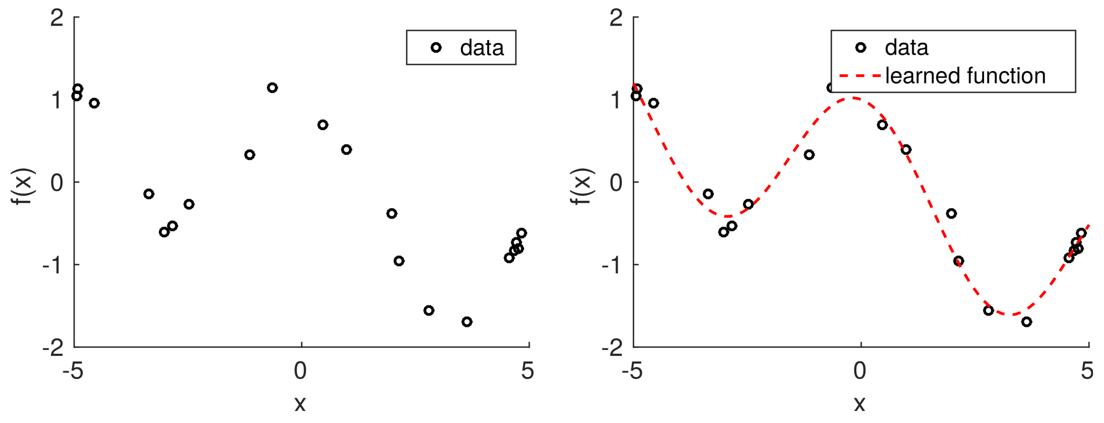
In this part of the course, we will be looking at regression problems, where we want to find a function f that maps inputs $\boldsymbol{x} \in \mathbb{R}^D$ to corresponding function values $f(\boldsymbol{x}) \in \mathbb{R}$ based on noisy observations $y = f(\boldsymbol{x}) + \epsilon$, where ϵ is a random variable.
An example is given in Fig. 1.1. A typical regression problem is given in Fig. 1.1(a): For some values x_i we observe (noisy) function values $y_i = f(x_i) + \epsilon$. The task is to find or infer the function f that generated the data. A possible solution is given in Fig. 1.1(b).
Regression is a fundamental problem, and regression problems appear in a diverse range of research areas and applications, including time-series analysis (e.g., system identification), control and robotics (e.g., reinforcement learning, forward/inverse model learning), optimization (e.g., line searches, global optimization), deep-learning applications (e.g., computer games, speech-to-text translation, image recognition, automatic video annotation).
Finding a regression function requires solving a variety of problems, including

- **Choice of the parametrization** of the regression function. Given a data set, what function classes (e.g., polynomials) are good candidates for modeling the data, and what particular parametrization (e.g., degree of the polynomial) should we choose?

- **Finding good parameters.** Having chosen a parametrization of the regression function, how do we find the parameter values? Here, we will need to look at different loss functions (they determine what a "good" fit is) and optimization algorithms that allow us to minimize this loss function.

- **Probabilistic models.** Loss functions are often motivated by probabilistic models. We will derive a set of useful models and discuss the corresponding underlying assumptions.

- **Overfitting and model selection.** Overfitting is a problem when the regression function fits the training data "too well" but does not generalize to the test data. Overfitting typically occurs if the underlying model (or its parametrization) is overly flexible and expressive. Model selection allows us to compare various models to find the simplest model that explains the training data reasonably well.



(a) Regression problem: Observed noisy function values from which we wish to infer the underlying function that generated the data.

(b) Regression solution: Possible function that could have generated the data.

Figure 1.1: (a) Regression problem and (b) possible solution.

- **Uncertainty modeling.** In any practical setting, we only have access a finite, potentially large, amount of (training) data for selecting the model class and the corresponding parameters. Given that this finite amount of training data does not cover all possible scenarios, we need to model the remaining uncertainty to obtain a measure of confidence of the model's prediction at test time. The smaller the training set the more important uncertainty modeling. Consistent modeling of uncertainty equips model predictions with confidence bounds.

In this chapter, we will be reviewing the necessary mathematical background for solving regression problems. This includes a brief introduction to probabilities and graphical models, which are useful to visualize relationships between random variables. Furthermore, we will go through some vector calculus, which is required for gradient-based optimization in the context of parameter learning. Once we know how to learn regression functions, we will discuss the problem of overfitting and model selection: Assuming we have a set of "realistic" models, are there some models that are better than others? Toward the end of this chapter, we will then discuss Bayesian linear regression, which allows us to reason about parameters at a higher level, thereby removing some of the problems encountered in maximum likelihood and MAP estimation.

1.1 Problem Formulation

We consider the regression problem

$$y = f(\boldsymbol{x}) + \epsilon\,, \tag{1.1}$$

where $\boldsymbol{x} \in \mathbb{R}^D$ are inputs and $y \in \mathbb{R}$ are observed targets. Furthermore, $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is independent, identically distributed (i.i.d.) measurement noise. In this particular

case, ϵ is Gaussian distributed with mean 0 and variance σ^2. The objective is to find a function f that is close to the unknown function that generated the data.
In this course, we focus on parametric models, i.e., we choose a parametrization of f and find parameters that "work well". In the most simple case, the parametrization of linear regression is

$$y = f(\boldsymbol{x}) + \epsilon = \boldsymbol{x}^\top\boldsymbol{\theta} + \epsilon \tag{1.2}$$

where $\boldsymbol{\theta} \in \mathbb{R}^D$ are the parameters we seek and $\epsilon \sim \mathcal{N}\big(0,\, \sigma^2\big)$.[1]
In this course, we will discuss in more detail how to

- Find good parameters $\boldsymbol{\theta}$
- Evaluate whether a parameter set "works well"

We will start by introducing some background material that is necessary and useful: concepts in probability theory, standard probability distributions, and probabilistic graphical models.

1.2 Probabilities

Probability theory is a mathematical foundation of statistics and a consistent way of expressing uncertainty. Jaynes (2003) provides a great introduction to probability theory.

Definition 1 (Probability Density Function)
A function $p : \mathbb{R}^D \to \mathbb{R}$ is called a **probability density function** *if (1) its integral exists, (2) $\forall \boldsymbol{x} \in \mathbb{R}^D : p(\boldsymbol{x}) \geq 0$ and (3)*

$$\int_{\mathbb{R}^D} p(\boldsymbol{x}) d\boldsymbol{x} = 1\,. \tag{1.3}$$

Here, $\boldsymbol{x} \in \mathbb{R}^D$ is a (continuous) **random variable**.[2] *For discrete random variables, the integral in* (1.3) *is replaced with a sum.*

Remark 1
We will be using the expression "probability distribution" not only for discrete distributions but also for continuous probability density functions, although this is technically incorrect.

There are two fundamental rules in probability theory that appear everywhere in machine learning and Bayesian statistics:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y} \qquad \textbf{Sum rule/Marginalization property} \tag{1.4}$$

[1]It would be more precise to call this model "linear in the parameters". We will see later that $\Phi^\top(\boldsymbol{x})\boldsymbol{\theta}$ for nonlinear transformations Φ is also a linear regression model.
[2]We omit the definition of a random variable as this will become too technical for the purpose of this course.

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x}) \qquad \textbf{Product rule} \tag{1.5}$$

Here, $p(\boldsymbol{x}, \boldsymbol{y})$ is the **joint distribution** of the two random variables $\boldsymbol{x}, \boldsymbol{y}$, $p(\boldsymbol{x}), p(\boldsymbol{y})$ are the corresponding **marginal distributions**, and $p(\boldsymbol{y}|\boldsymbol{x})$ is the **conditional distribution** of $\boldsymbol{y}$ given $\boldsymbol{x}$. If we consider discrete random variables $\boldsymbol{x}, \boldsymbol{y}$, the integral in (1.4) is replaced by a sum. This is where the name comes from.
In machine learning and Bayesian statistics, we are often interested in making inferences of random variables given that we have observed other random variables. Let us assume, we have some prior knowledge $p(\boldsymbol{x})$ about a random variable $\boldsymbol{x}$ and some relationship $p(\boldsymbol{y}|\boldsymbol{x})$ between $\boldsymbol{x}$ and a second random variable $\boldsymbol{y}$. If we now observe $\boldsymbol{y}$, we can use Bayes' theorem[3] to draw some conclusions about $\boldsymbol{x}$ given the observed values of $\boldsymbol{y}$. **Bayes' theorem** follows immediately from the sum and product rules in (1.4)–(1.5) as

$$p(\boldsymbol{x}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{y})} \,. \tag{1.6}$$

Here, $p(\boldsymbol{x})$ is the **prior**, which encapsulates our prior knowledge of $\boldsymbol{x}$, $p(\boldsymbol{y}|\boldsymbol{x})$ is the **likelihood**[4] that describes how $\boldsymbol{y}$ and $\boldsymbol{x}$ are related. The quantity $p(\boldsymbol{y})$ is the **marginal likelihood** or **evidence** and is a normalizing constant (independent of $\boldsymbol{x}$), which is obtained as the integral $\int p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$ of the numerator with respect to $\boldsymbol{x}$ and ensures that the fraction is normalized. The **posterior** $p(\boldsymbol{x}|\boldsymbol{y})$ expresses exactly what we are interested in, i.e., what we know about $\boldsymbol{x}$ if we observe $\boldsymbol{y}$.

Remark 2 (Marginal Likelihood)
Thus far, we looked at the marginal likelihood simply as a normalizing constant that ensures that the posterior probability distribution integrates to 1. In Section 1.7 we will see that the marginal likelihood also plays an important role in model selection.

1.2.1 Means and Covariances

Mean and (co)variance are often useful statistics to describe properties of probability distributions (expected values and spread) and data sets.

Definition 2 (Mean and Covariance)
The **mean** *of a random variable* $\boldsymbol{x} \in \mathbb{R}^D$ *is defined as*

$$\mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}] = \int \boldsymbol{x} p(\boldsymbol{x}) d\boldsymbol{x} = \begin{bmatrix} \mathbb{E}[x_1] \\ \vdots \\ \mathbb{E}[x_D] \end{bmatrix} \in \mathbb{R}^D \,, \tag{1.7}$$

where the subscript indicates the corresponding dimension of $\boldsymbol{x}$. *Intuitively, the mean is describes the average value of the random variable.*
The expected value of a function of a random variable $\boldsymbol{x} \sim p(\boldsymbol{x})$, *say* $g(\boldsymbol{x})$, *is given by*

$$\mathbb{E}[g(\boldsymbol{x})] = \int g(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \,. \tag{1.8}$$

[3]Also called the "probabilistic inverse"
[4]Also called the "measurement model"

In some cases, it may not be sufficient to only know about the expected value of a random variable. Variances and covariances allow us to characterize the variability of random variables and the relation between two random variables.
If we consider two random variables $\boldsymbol{x} \in \mathbb{R}^D, \boldsymbol{y} \in \mathbb{R}^E$, the **covariance** *between $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as*

$$\mathrm{Cov}[\boldsymbol{x}, \boldsymbol{y}] = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[\boldsymbol{x}\boldsymbol{y}^\top] - \mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}]\mathbb{E}_{\boldsymbol{y}}[\boldsymbol{y}]^\top = \mathrm{Cov}[\boldsymbol{y}, \boldsymbol{x}]^\top \in \mathbb{R}^{D\times E}\,. \tag{1.9}$$

Here, the subscript makes it explicit with respect to which variable we need to average. The covariance $\mathrm{Cov}[\boldsymbol{x}, \boldsymbol{y}]$ is an indicator how related $\boldsymbol{x}$ and $\boldsymbol{y}$ are. For example, if $\mathrm{Cov}[\boldsymbol{x}, \boldsymbol{y}] > 0$, then (on average) increasing $\boldsymbol{x}$ also increases $\boldsymbol{y}$.
The **variance** *of a random variable $\boldsymbol{x} \in \mathbb{R}^D$ with mean vector $\boldsymbol{\mu}$ is defined as*

$$\mathbb{V}_{\boldsymbol{x}}[\boldsymbol{x}] = \mathbb{E}_{\boldsymbol{x}}[(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^\top] = \mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}\boldsymbol{x}^\top] - \mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}]\mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}]^\top \tag{1.10}$$

$$= \begin{bmatrix} \mathrm{Cov}[x_1, x_1] & \mathrm{Cov}[x_1, x_2] & \dots & \mathrm{Cov}[x_1, x_D] \\ \mathrm{Cov}[x_2, x_1] & \mathrm{Cov}[x_2, x_2] & \dots & \mathrm{Cov}[x_2, x_D] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}[x_D, x_1] & \dots & \dots & \mathrm{Cov}[x_D, x_D] \end{bmatrix} \in \mathbb{R}^{D\times D}\,. \tag{1.11}$$

This matrix is called the **covariance matrix** *of the random variable $\boldsymbol{x}$. The covariance matrix is symmetric and positive definite and tells us something about the spread of the data.*
The covariance matrix contains the variances of the marginals

$$p(x_i) = \int p(x_1, \dots, x_D) dx_{\backslash i} \tag{1.12}$$

on its diagonal, where "$\backslash i$" denotes "all variables but i". The off-diagonal terms contain the **cross-covariance** *terms $\mathrm{Cov}[x_i, x_j]$ for $i, j = 1, \dots, D, i \neq j$.*

It generally holds that

$$\mathbb{V}_{\boldsymbol{x}}[\boldsymbol{x}] = \mathrm{Cov}_{\boldsymbol{x}}[\boldsymbol{x}, \boldsymbol{x}]\,. \tag{1.13}$$

Remark 3 (Statistics of Data Sets)
We are often given data sets $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$, where we assume that $\boldsymbol{x}_i \sim p(\boldsymbol{x})$ for some distribution p. Even if p is unknown, it is still possible to compute estimate of the mean and covariance using Monte-Carlo estimation. In particular, the mean and covariance estimates (given the data set) are

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \tag{1.14}$$

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^\top\,, \tag{1.15}$$

respectively.

1.2.1.1 Sum of Random Variables

Consider two random variables $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^D$. It holds that

$$\mathbb{E}[\boldsymbol{x}+\boldsymbol{y}] = \mathbb{E}[\boldsymbol{x}]+\mathbb{E}[\boldsymbol{y}] \tag{1.16}$$

$$\mathbb{E}[\boldsymbol{x}-\boldsymbol{y}] = \mathbb{E}[\boldsymbol{x}]-\mathbb{E}[\boldsymbol{y}] \tag{1.17}$$

$$\mathbb{V}[\boldsymbol{x}+\boldsymbol{y}] = \mathbb{V}[\boldsymbol{x}] + \mathbb{V}[\boldsymbol{y}]+ \mathrm{Cov}[\boldsymbol{x},\boldsymbol{y}]+ \mathrm{Cov}[\boldsymbol{y},\boldsymbol{x}] \tag{1.18}$$

$$\mathbb{V}[\boldsymbol{x}-\boldsymbol{y}] = \mathbb{V}[\boldsymbol{x}] + \mathbb{V}[\boldsymbol{y}]- \mathrm{Cov}[\boldsymbol{x},\boldsymbol{y}]- \mathrm{Cov}[\boldsymbol{y},\boldsymbol{x}] \tag{1.19}$$

1.2.1.2 Affine Transformation

Mean and (co)variance exhibit some useful properties when it comes to affine transformation of random variables. Consider a random variable $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance matrix Σ and a (deterministic) affine transformation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ of $\boldsymbol{x}$. Then, $\boldsymbol{y}$ is itself a random variable whose mean vector and covariance matrix are given by

$$\mathbb{E}_{\boldsymbol{y}}[\boldsymbol{y}] = \mathbb{E}_{\boldsymbol{x}}[\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}] = \boldsymbol{A}\mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}] + \boldsymbol{b} = \boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}\,, \tag{1.20}$$

$$\mathbb{V}_{\boldsymbol{y}}[\boldsymbol{y}] = \mathbb{V}_{\boldsymbol{x}}[\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}] = \mathbb{V}_{\boldsymbol{x}}[\boldsymbol{A}\boldsymbol{x}] = \boldsymbol{A}\mathbb{V}_{\boldsymbol{x}}[\boldsymbol{x}]\boldsymbol{A}^\top = \boldsymbol{A}\Sigma\boldsymbol{A}^\top\,, \tag{1.21}$$

respectively.[5] Furthermore,

$$\mathrm{Cov}[\boldsymbol{x},\boldsymbol{y}] = \int \boldsymbol{x}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b})^\top p(\boldsymbol{x})d\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}]^\top \tag{1.22}$$

$$= \int \boldsymbol{x}p(\boldsymbol{x})d\boldsymbol{x}\boldsymbol{b}^\top + \int \boldsymbol{x}\boldsymbol{x}^\top p(\boldsymbol{x})d\boldsymbol{x}\boldsymbol{A}^\top - \boldsymbol{\mu}\boldsymbol{b}^\top - \boldsymbol{\mu}\boldsymbol{\mu}^\top\boldsymbol{A}^\top \tag{1.23}$$

$$= \boldsymbol{\mu}\boldsymbol{b}^\top - \boldsymbol{\mu}\boldsymbol{b}^\top + \Big(\int \boldsymbol{x}\boldsymbol{x}^\top p(\boldsymbol{x})d\boldsymbol{x} - \boldsymbol{\mu}\boldsymbol{\mu}^\top\Big)\boldsymbol{A}^\top \tag{1.24}$$

$$\overset{(1.10)}{=} \Sigma\boldsymbol{A}^\top \tag{1.25}$$

Affine transformation of random variables are the key idea behind the "reparametrization trick", which is used in many deep learning algorithms to propagate gradients through stochastic functions (stochastic backpropagation), e.g., in the variational auto-encoder proposed by Jimenez Rezende et al. (2014) and Kingma and Welling (2014).

1.2.2 Statistical Independence

Definition 3 (Independence)
Two random variables $\boldsymbol{x}, \boldsymbol{y}$ *are* **statistically independent** *if and only if*

$$p(\boldsymbol{x},\boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y})\,. \tag{1.26}$$

Intuitively, two random variables $\boldsymbol{x}$ and $\boldsymbol{y}$ are independent if the value of $\boldsymbol{y}$ (once known) does not add any additional information about $\boldsymbol{x}$ (and vice versa).
If $\boldsymbol{x}, \boldsymbol{y}$ are (statistically) independent then

[5] This can be shown directly by using the definition of the mean and covariance.

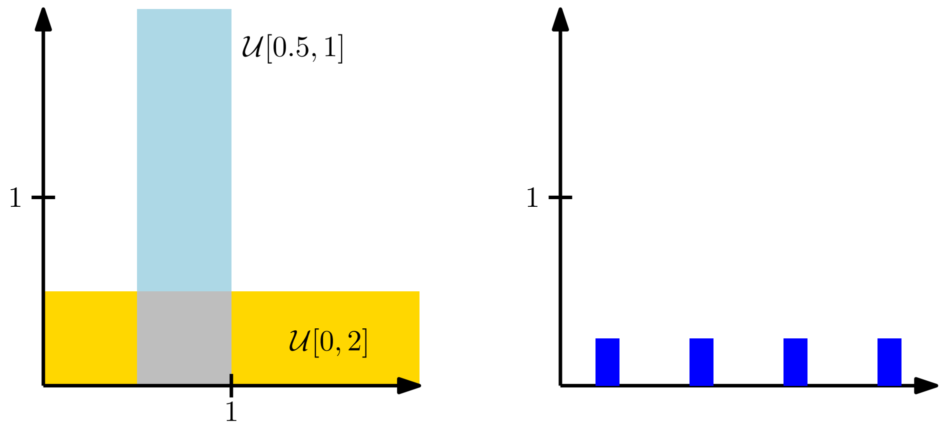


Figure 1.2: Examples of uniform distributions. Left: Continuous uniform distribution that distributes probability mass equally everywhere in a (compact) region. Right: Discrete uniform distribution that assigns equal probability to four possible (discrete) events.

- $p(\boldsymbol{y}|\boldsymbol{x}) = p(\boldsymbol{y})$
- $p(\boldsymbol{x}|\boldsymbol{y}) = p(\boldsymbol{x})$
- $\mathbb{V}[\boldsymbol{x} + \boldsymbol{y}] = \mathbb{V}[\boldsymbol{x}] + \mathbb{V}[\boldsymbol{y}]$
- $\mathrm{Cov}[\boldsymbol{x}, \boldsymbol{y}] = \mathbf{0}$

Another concept that is important in machine learning is conditional independence.

Definition 4 (Conditional Independence)
Formally, $\boldsymbol{x}$ and $\boldsymbol{y}$ are **conditionally independent given $\boldsymbol{z}$** *if and only if*

$$p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{y}|\boldsymbol{z})\,. \tag{1.27}$$

We write $\boldsymbol{x} \perp \boldsymbol{y}|\boldsymbol{z}$.

Independence can be cast as a special case of conditional independence if we write $\boldsymbol{x} \perp \boldsymbol{y}|\emptyset$.

1.2.3 Basic Probability Distributions

In the following, we will briefly introduce basic probability distributions.

1.2.3.1 Uniform Distribution

The uniform distribution is a distribution that assigns equal probability mass in a region. For $a, b \in \mathbb{R}$ and $a < b$, the uniform distribution is defined as

$$\mathcal{U}[a, b] = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{1.28}$$

Mean and variance of the uniform distribution $\mathcal{U}[a, b]$ are $\frac{1}{2}(a + b)$ and $\frac{1}{12}(b - a)^2$, respectively.

Figure 1.3: The Bernoulli distribution can be used to model the binary outcome probability of a coin flip experiment.

1.2.3.2 Bernoulli Distribution

The Bernoulli distribution is a distribution for a single binary variable $x \in \{0, 1\}$ and is governed by a single continuous parameter $\mu \in [0, 1]$ that represents the probability of $x = 1$. The Bernoulli distribution is defined as

$$p(x|\mu) = \mu^x(1-\mu)^{1-x}\,, \quad x \in \{0,1\}\,, \tag{1.29}$$
$$\mathbb{E}[x] = \mu\,, \tag{1.30}$$
$$\mathbb{V}[x] = \mu(1-\mu)\,, \tag{1.31}$$

where $\mathbb{E}[x]$ and $\mathbb{V}[x]$ are the mean and variance of the binary random variable x.
An example where the Bernoulli distribution can be used is when we are interested in modeling the probability of "head" when flipping a coin.

1.2.3.3 Binomial Distribution

The Binomial distribution is a generalization of the Bernoulli distribution to a distribution over integers. In particular, the Binomial can be used to describe the probability of observing m occurrences of $x = 1$ in a set of N samples from a Bernoulli distribution where $p(x = 1) = \mu \in [0, 1]$. The Binomial distribution is defined as

$$p(m|N,\mu) = \binom{N}{m}\mu^m(1-\mu)^{N-m}\,, \tag{1.32}$$
$$\mathbb{E}[m] = N\mu\,, \tag{1.33}$$
$$\mathbb{V}[m] = N\mu(1-\mu) \tag{1.34}$$

where $\mathbb{E}[m]$ and $\mathbb{V}[m]$ are the mean and variance of m, respectively.
An example where the Binomial could be used is if we want to describe the probability of observing m "heads" in N coin-flip experiments if the probability for observing head in a single experiment is μ?

1.2.3.4 Beta Distribution

The Beta distribution is a distribution over a continuous variable $\mu \in [0, 1]$, which is often used to represent the probability for some binary event (e.g., the parameter

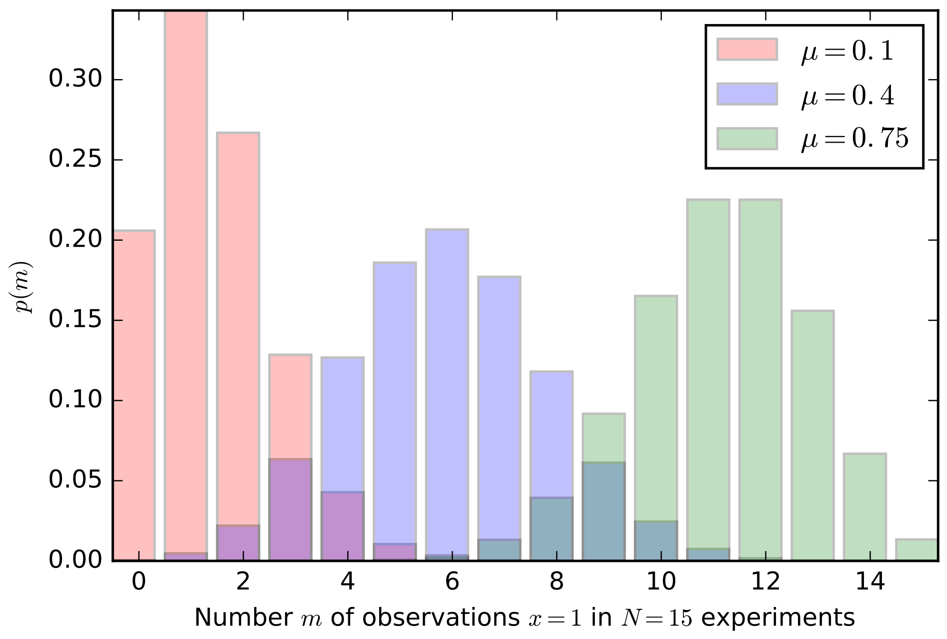


Figure 1.4: Examples of the Binomial distribution for $\mu \in \{0.1, 0.4, 0.75\}$ and $N = 15$.

governing the Bernoulli distribution). The Beta distribution itself is governed by two parameters $\alpha > 0,\ \beta > 0$ and is defined as

$$p(\mu|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\mu^{\alpha-1}(1-\mu)^{\beta-1} \tag{1.35}$$

$$\mathbb{E}[\mu] = \frac{\alpha}{\alpha+\beta}, \qquad \mathbb{V}[\mu] = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \tag{1.36}$$

where $\Gamma(\cdot)$ is the Gamma function defined as

$$\Gamma(t) := \int_0^\infty x^{t-1}\exp(-x)dx, \qquad t > 0\,. \tag{1.37}$$

$$\Gamma(t+1) = t\Gamma(t)\,. \tag{1.38}$$

Note that the fraction of Gamma functions in (1.35) normalizes the Beta distribution. Intuitively, α moves probability mass toward 1, whereas β moves probability mass toward 0. There are some special cases (Murphy, 2012):

- For $\alpha = 1 = \beta$ we obtain the uniform distribution $\mathcal{U}[0,1]$.
- For $\alpha, \beta < 1$, we get a bimodal distribution with spikes at 0 and 1.
- For $\alpha, \beta > 1$, the distribution is unimodal.
- For $\alpha, \beta > 1$ and $\alpha = \beta$, the distribution is unimodal, symmetric and centered in the interval $[0,1]$, i.e., the mode/mean is at $\frac{1}{2}$.

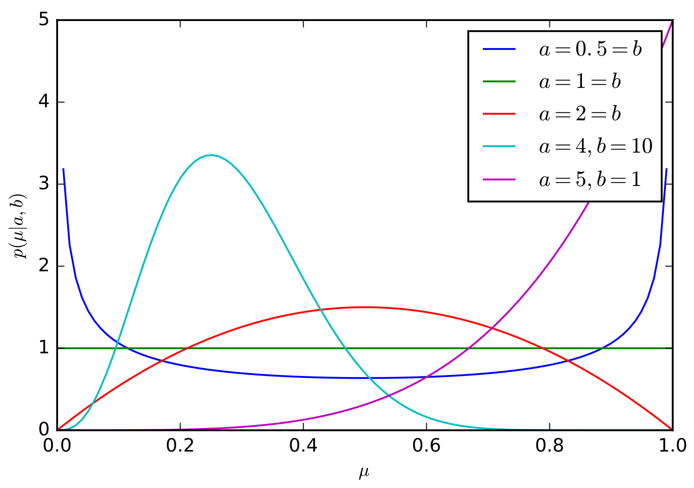


Figure 1.5: Examples of the Beta distribution for different values of α and β.

1.2.3.5 Gaussian Distribution

The **multivariate Gaussian distribution**[6] is fully characterized by a **mean vector** $\boldsymbol{\mu}$ and a **covariance matrix** Σ and defined as

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{D}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\big(-\tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\big)\,, \tag{1.39}$$

where $\boldsymbol{x} \in \mathbb{R}^D$ is a random variable. We write $\boldsymbol{x} \sim \mathcal{N}\big(\boldsymbol{x}\,|\,\boldsymbol{\mu},\,\Sigma\big)$ or $\boldsymbol{x} \sim \mathcal{N}\big(\boldsymbol{\mu},\,\Sigma\big)$. Figure 1.6 shows a bi-variate Gaussian (mesh), with the corresponding contour plot. The Gaussian distribution is the most important probability distribution[7] for continuous-valued random variables. Its importance originates from the fact that it has many computationally convenient properties, which we will be discussing in the following. Application areas in which the Gaussian plays a central role range from signal processing (e.g., Kalman filter) to control (e.g., linear quadratic regulator) and machine learning (e.g., Gaussian processes, principal component analysis, clustering with Gaussian mixture models and k-means, linear regression, deep learning with squared errors, variational inference, reinforcement learning).

Conditional and Marginal Consider the joint Gaussian distribution

$$p(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy}\end{bmatrix}\right) \tag{1.40}$$

of two random variables $\boldsymbol{x}, \boldsymbol{y}$, where $\boldsymbol{\Sigma}_{xx} = \mathrm{Cov}[\boldsymbol{x}, \boldsymbol{x}]$ and $\boldsymbol{\Sigma}_{yy} = \mathrm{Cov}[\boldsymbol{y}, \boldsymbol{y}]$ are the marginal covariance matrices of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively, and $\boldsymbol{\Sigma}_{xy} = \mathrm{Cov}[\boldsymbol{x}, \boldsymbol{y}]$ is the cross-covariance matrix between $\boldsymbol{x}$ and $\boldsymbol{y}$.

[6]Also: multivariate normal distribution
[7]We will be adopting a common, but mathematically slightly sloppy, language and call the "probability density function" a "distribution".

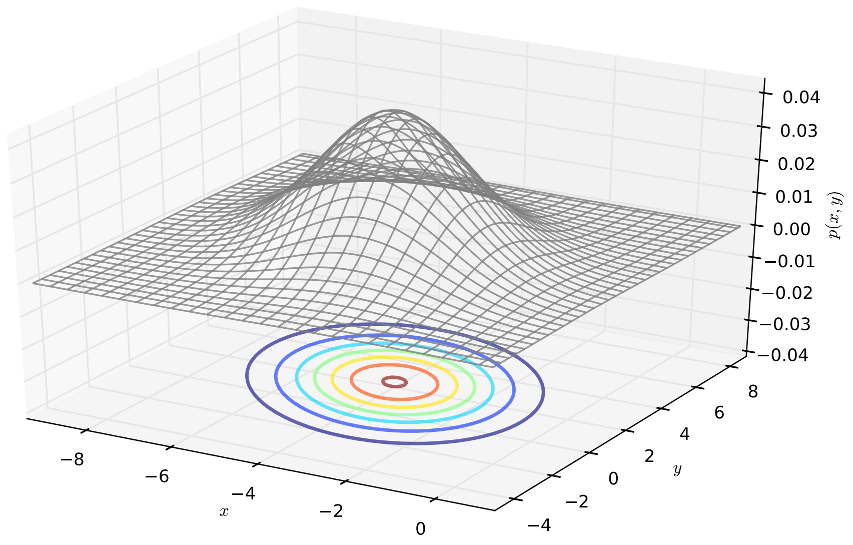


Figure 1.6: Gaussian distribution of two random variables x, y.

The conditional distribution $p(\boldsymbol{x}|\boldsymbol{y})$ is also Gaussian and given by

$$p(\boldsymbol{x}|\boldsymbol{y}) = \mathcal{N}\big(\boldsymbol{\mu}_{x|y},\, \boldsymbol{\Sigma}_{x|y}\big) \tag{1.41}$$

$$\boldsymbol{\mu}_{x|y} = \boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_y) \tag{1.42}$$

$$\boldsymbol{\Sigma}_{x|y} = \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\,. \tag{1.43}$$

Note that in the computation of the mean in (1.42) the $\boldsymbol{y}$-value is an observation and no longer random.

Remark 4

The conditional Gaussian distribution shows up in many places, where we are interested in posterior distributions:

- *The Kalman filter (Kalman, 1960), one of the most central algorithms for state estimation in signal processing, does nothing but computing Gaussian conditionals of joint distributions (Deisenroth and Ohlsson, 2011).*
- *Gaussian processes (Rasmussen and Williams, 2006), which are a practical implementation of a distribution over functions. In a Gaussian process, we make assumptions of joint Gaussianity of random variables. By (Gaussian) conditioning on observed data, we can determine a posterior distribution over functions.*
- *Latent linear Gaussian models (Roweis and Ghahramani, 1999; Murphy, 2012), which include probabilistic PCA (Tipping and Bishop, 1999).*

The marginal distribution $p(\boldsymbol{x})$[8] of a joint Gaussian distribution $p(\boldsymbol{x}, \boldsymbol{y})$, see (1.40), is itself Gaussian and computed by applying the sum-rule in (1.4) and given by

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{y})d\boldsymbol{y} = \mathcal{N}\big(\boldsymbol{x}\,|\,\boldsymbol{\mu}_x,\, \boldsymbol{\Sigma}_{xx}\big)\,. \tag{1.44}$$

[8]The same holds for $p(\boldsymbol{y})$.

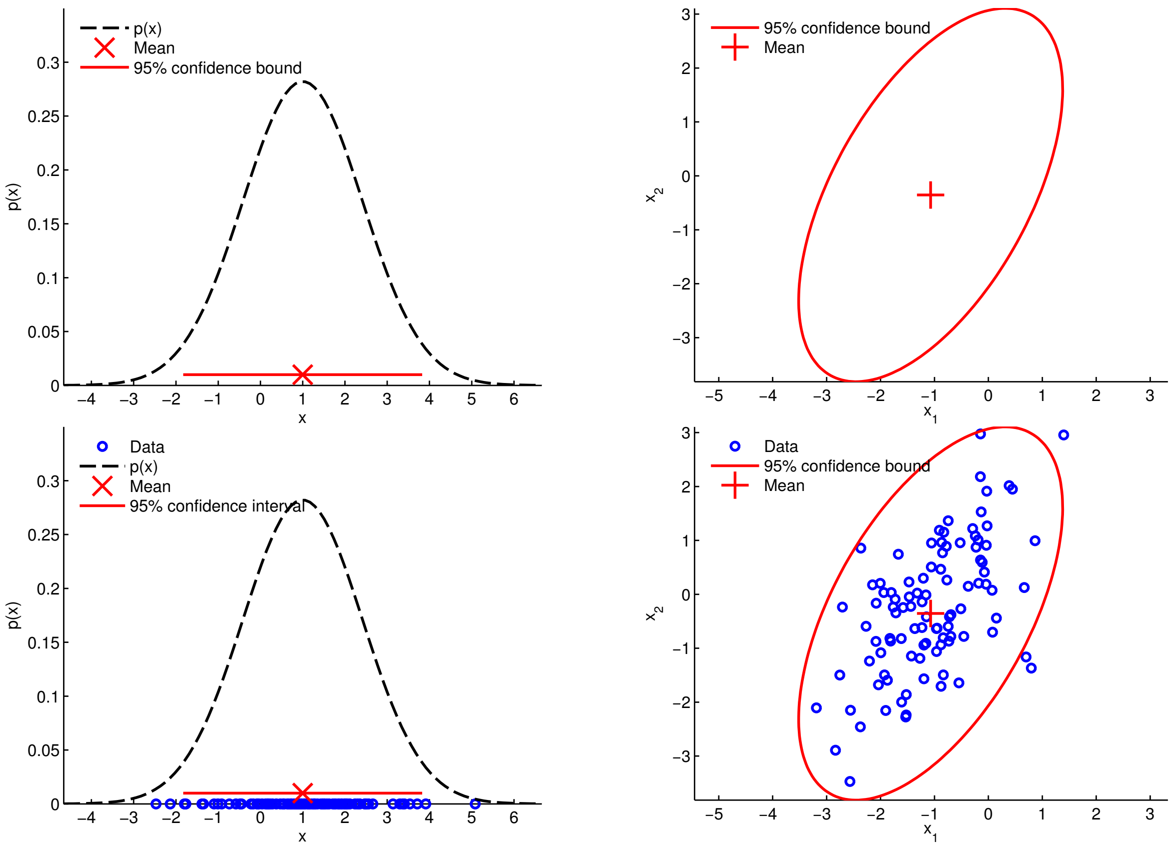


Figure 1.7: Gaussian distributions. Left: Univariate (1-dimensional) Gaussians; Right: Multivariate (2-dimensional) Gaussians, viewed from top. Crosses show the mean of the distribution, the red solid (contour) lines represent the 95% confident bounds. The bottom row shows the Gaussians overlaid with 100 samples. We expect that on average 95/100 samples are within the red contour lines/intervals that indicate the 95% confidence bounds.

Intuitively, looking at the joint distribution in (1.40), we ignore (i.e., integrate out) everything we are not interested in.

Product of Gaussians The product of two Gaussians $\mathcal{N}\big(\boldsymbol{x}\,|\,\boldsymbol{a},\,\boldsymbol{A}\big)\mathcal{N}\big(\boldsymbol{x}\,|\,\boldsymbol{b},\,\boldsymbol{B}\big)$ is an unnormalized Gaussian distribution $c\mathcal{N}\big(\boldsymbol{x}\,|\,\boldsymbol{c},\,\boldsymbol{C}\big)$ with

$$\boldsymbol{C} = (\boldsymbol{A}^{-1} + \boldsymbol{B}^{-1})^{-1} \tag{1.45}$$

$$\boldsymbol{c} = \boldsymbol{C}(\boldsymbol{A}^{-1}\boldsymbol{a} + \boldsymbol{B}^{-1}\boldsymbol{b}) \tag{1.46}$$

$$c = (2\pi)^{-\frac{D}{2}}|\boldsymbol{A}+\boldsymbol{B}|^{-\frac{1}{2}}\exp\big(-\tfrac{1}{2}(\boldsymbol{a}-\boldsymbol{b})^\top(\boldsymbol{A}+\boldsymbol{B})^{-1}(\boldsymbol{a}-\boldsymbol{b})\big)\,. \tag{1.47}$$

Note that the normalizing constant c itself is a Gaussian either in $\boldsymbol{a}$ or in $\boldsymbol{b}$ with an “inflated” covariance matrix $\boldsymbol{A}+\boldsymbol{B}$, i.e., $c = \mathcal{N}\big(\boldsymbol{a}\,|\,\boldsymbol{b},\,\boldsymbol{A}+\boldsymbol{B}\big) = \mathcal{N}\big(\boldsymbol{b}\,|\,\boldsymbol{a},\,\boldsymbol{A}+\boldsymbol{B}\big)$.

Linear Transformation of Gaussian Random Variables A linear (or affine) transformation of a Gaussian random variable is Gaussian distributed.

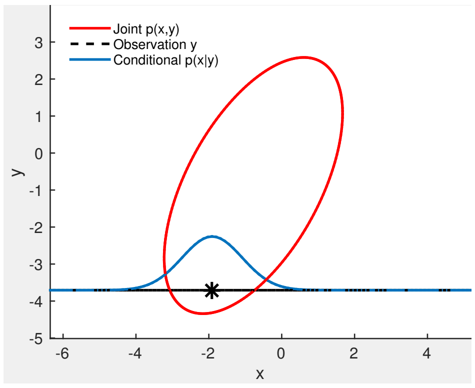


Figure 1.8: The conditional distribution of a Gaussian is also Gaussian

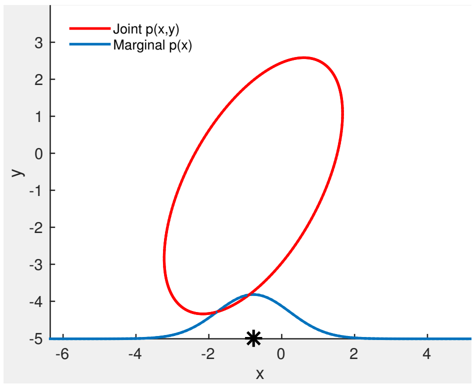


Figure 1.9: Marginal of a joint Gaussian distribution is Gaussian.

- If $p(\boldsymbol{x}) = \mathcal{N}\big(\boldsymbol{x}\,|\,\boldsymbol{\mu},\,\boldsymbol{\Sigma}\big)$ and $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ then $p(\boldsymbol{y}) = \mathcal{N}\big(\boldsymbol{y}\,|\,\boldsymbol{A}\boldsymbol{\mu},\,\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top\big)$.
- If $p(\boldsymbol{y}) = \mathcal{N}\big(\boldsymbol{y}\,|\,\boldsymbol{A}\boldsymbol{x},\,\boldsymbol{\Sigma}\big)$ then we can re-write this as a probability distribution in $\boldsymbol{x}$: If $\boldsymbol{A}$ was invertible, we could write $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{y}$. However, $\boldsymbol{A}$ is not generally invertible. Therefore, we perform a transformation with $\boldsymbol{A}^\top$:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \Leftrightarrow \boldsymbol{A}^\top\boldsymbol{y} = \boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x}\,. \tag{1.48}$$

$\boldsymbol{A}^\top\boldsymbol{A}$ is symmetric and positive definite[9], i.e., it can be inverted. Then,

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \Leftrightarrow (\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top\boldsymbol{y} = \boldsymbol{x}\,. \tag{1.49}$$

Hence, $\boldsymbol{x}$ is a linear transformation of $\boldsymbol{y}$, and we obtain

$$p(\boldsymbol{x}) = \mathcal{N}\big(\boldsymbol{x}\,|\,(\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top\boldsymbol{y},\,(\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top\boldsymbol{\Sigma}\boldsymbol{A}(\boldsymbol{A}^\top\boldsymbol{A})^{-1}\big)\,. \tag{1.50}$$

Sampling from Multivariate Gaussian Distributions Assume we are interested in generating samples $\boldsymbol{x}_i, i = 1,\dots,n$, from a multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. However, we only have access to a sampler that allows us to generate samples from the standard normal $\mathcal{N}\big(\boldsymbol{0},\,\boldsymbol{I}\big)$.

[9]Actually, only positive semi-definite, but with mild assumptions we arrive at positive definite.

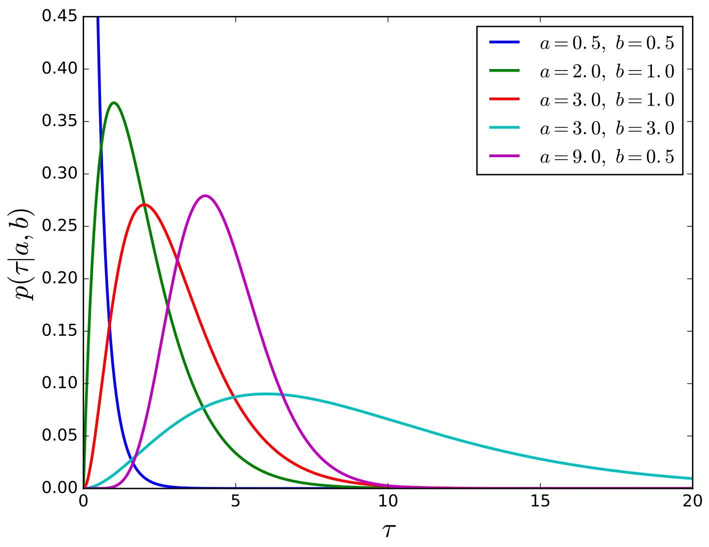


Figure 1.10: Gamma distribution for different values of a, b.

To obtain samples from a multivariate normal $\mathcal{N}(\boldsymbol{\mu},\, \boldsymbol{\Sigma})$, we can use the properties of a linear transformation of a Gaussian random variable: If $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0},\, \boldsymbol{I})$ then $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\mu}$, where $\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{\Sigma}$, is Gaussian distributed with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. We call $\boldsymbol{\Sigma} = \boldsymbol{A}\boldsymbol{A}^\top$ the **Cholesky factorization** of $\boldsymbol{\Sigma}$.[10] The idea of an affine transformation of $\mathcal{N}(\boldsymbol{0},\, \boldsymbol{I})$-distributed random variables is an example of the **reparametrization trick**, which is key to propagating gradient information in the variational auto-encoder (Jimenez Rezende et al., 2014; Kingma and Welling, 2014).[11]

Sum of Independent Gaussian Random Variables If $\boldsymbol{x}, \boldsymbol{y}$ are independent Gaussian random variables (i.e., the joint is given as $p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y})$) with $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x} \,|\, \boldsymbol{\mu}_x,\, \boldsymbol{\Sigma}_x)$ and $p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{y} \,|\, \boldsymbol{\mu}_y,\, \boldsymbol{\Sigma}_y)$, then $\boldsymbol{x}+\boldsymbol{y}$ is also Gaussian distributed and given by

$$p(\boldsymbol{x} + \boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}_x + \boldsymbol{\mu}_y,\, \boldsymbol{\Sigma}_x + \boldsymbol{\Sigma}_y)\,. \tag{1.51}$$

Knowing that $p(\boldsymbol{x} + \boldsymbol{y})$ is Gaussian, the mean and covariance matrix can be determined immediately using the results from (1.16)–(1.19). This property will be important when we consider i.i.d. Gaussian noise acting on random variables.

1.2.3.6 Gamma Distribution

The Gamma distribution is a distribution over positive real numbers $\tau > 0$. The Gamma distribution is itself governed by two parameters $a, b > 0$, where a is a shape parameter and b a scale parameter of the corresponding density.

[10]To compute the Cholesky factorization of a matrix, it is required that the matrix is symmetric and positive definite. Covariance matrices possess this property.

[11]An excellent blog post about the reparametrization trick is available at `http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/`.

The Gamma distribution is defined as

$$p(\tau|a,b) = \frac{1}{\Gamma(a)} b^a \tau^{a-1} \exp(-b\tau)\,, \tag{1.52}$$

$$\mathbb{E}[\tau] = \frac{a}{b}\,, \tag{1.53}$$

$$\mathbb{V}[\tau] = \frac{a}{b^2}\,. \tag{1.54}$$

The Gamma distribution is often used as a prior on the precision (inverse variance) of a univariate Gaussian distribution.

Remark 5
Other distributions are special cases of the Gamma distribution (Murphy, 2012): The Exponential distribution with parameter λ is obtained for $(a,b) = (1,\lambda)$. The exponential distribution describes the time between events in a Poisson process. The Erlang distribution is a Gamma distribution with $a \in \mathbb{N}$. In the Chi-Squared distribution, which is the distribution of the sum of Gaussian random variables, the scale parameter b is fixed to $b = \frac{1}{2}$.

1.2.3.7 Wishart Distribution

The Wishart distribution is the multivariate generalization of the Gamma distribution: It is a family of probability distributions defined over symmetric, nonnegative-definite matrix-valued random variables ("random matrices").
For $D \times D$ matrices the **Wishart distribution** is defined as

$$\mathcal{W}(\boldsymbol{\Sigma}|\boldsymbol{W},\nu) = B|\boldsymbol{\Sigma}|^{\frac{\nu-D-1}{2}} \exp\left(-\tfrac{1}{2}\mathrm{tr}(\boldsymbol{W}^{-1}\boldsymbol{\Sigma})\right) \tag{1.55}$$

where ν is called the **number of degrees of freedom** of the distribution, and $\boldsymbol{W}$ is a $D \times D$ scale matrix. B is a normalization constant (Bishop, 2006, p. 102) that ensures that the distribution is normalized.
These distributions are of great importance in the estimation of covariance matrices in multivariate statistics: The Wishart distribution is the conjugate prior for the precision matrix (inverse covariance matrix) of a Gaussian-distributed random variable $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu},\, \boldsymbol{\Sigma})$.

1.2.4 Conjugacy

According to Bayes' theorem (1.6), the posterior is proportional to the product of the prior and the likelihood. The specification of the prior can be tricky for two reasons: First, the prior should encapsulate our knowledge about the problem before we see some data. This is often difficult to describe. Second, it is often not possible to compute the posterior distribution analytically. However, there are some priors that are computationally convenient: **conjugate priors**.

Definition 5 (Conjugate Prior)
A prior is **conjugate** *for the likelihood function if the posterior is of the same form/type as the prior.*

Example (Beta-Binomial Conjugacy)
Consider a Binomial random variable $x \sim \text{Bin}(m|N, \mu)$ where

$$p(x|\mu, N) = \binom{N}{m} \mu^m (1-\mu)^{N-m} \;\propto\; \mu^a (1-\mu)^b \tag{1.56}$$

for some constants a, b. We place a Beta prior on the parameter μ:

$$\text{Beta}(\mu|\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \mu^{\alpha-1}(1-\mu)^{\beta-1} \;\propto\; \mu^{\alpha-1}(1-\mu)^{\beta-1} \tag{1.57}$$

If we now observe some outcomes $\boldsymbol{x} = (x_1, \ldots, x_N)$ of a repeated coin-flip experiment with h heads and t tails, we compute the posterior distribution on μ as

$$p(\mu|\boldsymbol{x} = h) \;\propto\; p(\boldsymbol{x}|\mu)p(\mu|\alpha, \beta) = \mu^h (1-\mu)^t \mu^{\alpha-1}(1-\mu)^{\beta-1} \tag{1.58}$$

$$= \mu^{h+\alpha-1}(1-\mu)^{t+\beta-1} \;\propto\; \text{Beta}(h+\alpha, t+\beta) \tag{1.59}$$

i.e., the posterior distribution is a Beta distribution as the prior, i.e., the Beta prior is conjugate for the parameter μ in the Binomial likelihood function.

Table 1.1 lists examples for conjugate priors for the parameters of some of the standard distributions that we discussed in this section.

Table 1.1: Standard examples of conjugate priors.

Conjugate prior	Likelihood	Posterior
Beta	Bernoulli	Beta
Beta	Binomial	Beta
Gaussian/inverse Gamma	Gaussian	Gaussian/inverse Gamma
Gaussian/inverse Wishart	Gaussian	Gaussian/inverse Wishart
Dirichlet	Multinomial	Dirichlet

The Beta distribution is the conjugate prior for the parameter μ in both the Binomial and the Bernoulli likelihood. For a Gaussian likelihood function, we can place a conjugate Gaussian prior on the mean. The reason why the Gaussian likelihood appears twice in the table is that we need distinguish the univariate from the multivariate case. In the univariate (scalar) case, the inverse Gamma is the conjugate prior for the variance[12]. In the multivariate case, we use a conjugate inverse Wishart distribution as a prior on the covariance matrix[13]. The Dirichlet distribution is the conjugate prior for the multinomial likelihood function. For further details, we refer to Bishop (2006).

[12]Alternatively, the Gamma prior is conjugate for the precision (inverse variance) in the Gaussian likelihood.

[13]Alternatively, the Wishart prior is conjugate for the precision matrix (inverse covariance matrix) in the Gaussian likelihood.

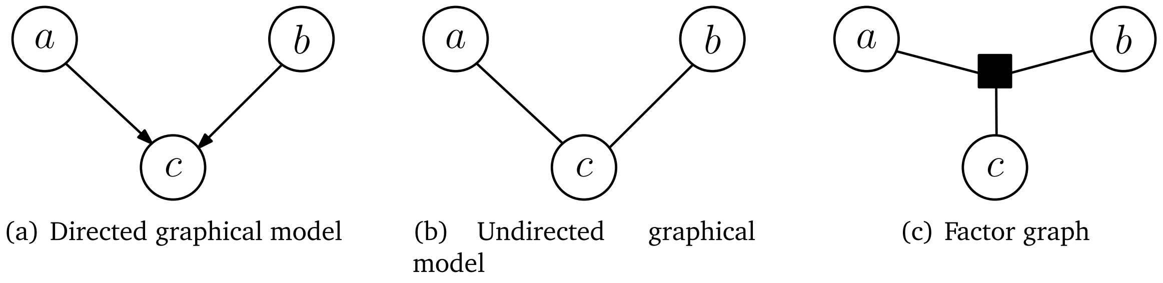


Figure 1.11: Three types of graphical models: (a) Directed graphical models (Bayesian network); (b) Undirected graphical models (Markov random field); (c) Factor graphs.

1.3 Probabilistic Graphical Models

A graphical model captures the way in which the joint distribution over all random variables can be decomposed into a product of factors depending only on a subset of these variables.
There are three main types of graphical models:

- Directed graphical models (Bayesian networks)
- Undirected graphical models (Markov random fields)
- Factor graphs

Nodes are (random) variables, edges represent probabilistic relations between variables. In this course, we will focus on directed graphical models.[14]
Probabilistic graphical models have some convenient properties:

- They are a simple way to visualize the structure of a probabilistic model
- They can be used to design or motivate new kind of statistical models
- Inspection of the graph alone gives us insight into properties, e.g., conditional independence
- Complex computations for inference and learning in statistical models can be expressed in terms of graphical manipulations.

1.3.1 From Joint Distributions to Graphs

Consider the joint distribution

$$p(a,b,c) = p(c|a,b)p(b|a)p(a) \tag{1.60}$$

of three random variables a, b, c. The factorization of the joint in (1.60) tell us something about the relationship between the random variables:

[14]"Data Analysis and Probabilistic Inference" (CO-493) will discuss all three types of graphical models.

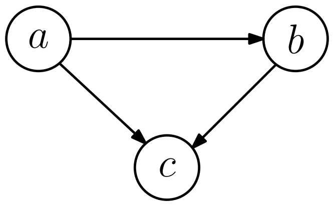


Figure 1.12: Directed graphical model for the factorization of the joint distribution in (1.60).

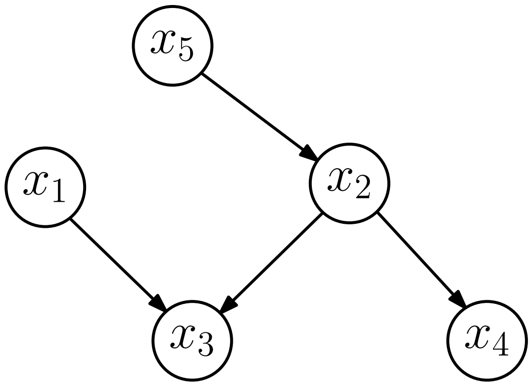


Figure 1.13: Directed graphical model for which we seek the corresponding joint distribution and its factorization.

- c depends directly on a and b
- b depends directly on a
- a depends neither on b nor on c

We can build the corresponding directed graphical model as follows:

1. Create a node for all random variables
2. For each conditional distribution, we add a directed link (arrow) to the graph from the nodes corresponding to the variables on which the distribution is conditioned on

Note that the graph layout depends on the choice of factorization. For the factorization in (1.60), we obtain the directed graphical model in Fig. 1.12.

1.3.2 From Graphs to Joint Distributions

In the following, we will discuss how to extract the joint distribution of a set of random variables from a given graphical model. We will immediately look at the graphical model in Fig. 1.13, and exploit two observations:

- The joint distribution $p(x_1, \dots, x_5)$ we seek is the product of a set of conditionals, one for each node in the graph. In this particular example, we will need five conditionals.

- Each conditional depends only on the parents of the corresponding node in the graph. For example, x_4 will be conditioned on x_2.

Using these two properties, we arrive at the desired factorization of the joint distribution

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_5)p(x_2|x_5)p(x_3|x_1, x_2)p(x_4|x_2)\,. \tag{1.61}$$

In general, the joint distribution $p(\boldsymbol{x}) = p(x_1, \ldots, x_K)$ is given as

$$p(\boldsymbol{x}) = \prod_{k=1}^{K} p(x_k|\text{pa}_k) \tag{1.62}$$

where pa_k means "the parent nodes of x_k".

Example (Linear Regression)
We seek the graphical model representation for the linear regression setting

$$y_n = \boldsymbol{x}_n^\top \boldsymbol{\theta} + \epsilon\,, \quad \epsilon \sim \mathcal{N}\big(0,\, \sigma^2\big)\,, \tag{1.63}$$

for $n = 1, \ldots, N$, where y_n are observed variables.
In this example, we have three different types of "variables":

- Unknown random variables $\boldsymbol{\theta}$
- Observed random variables y_n
- Deterministic parameters $\boldsymbol{x}_n, \sigma$, which are fixed

To make the distinction between these three types easier, we introduce additional nodes for graphical models:

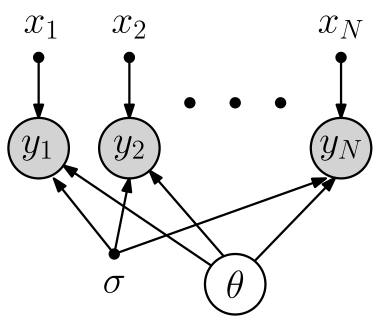
- Shaded nodes represent observed random variables
- Dots represent deterministic parameters

To find the directed graphical model, for all (observed and unobserved) random variables we write down all probability distributions with explicit conditioning on the parameters/variables they depend on. In our case, we end up with:

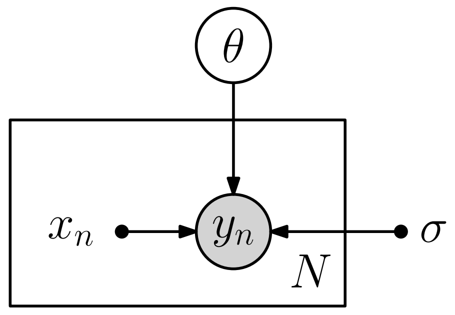
- $p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}, \sigma)$
- $p(\boldsymbol{\theta})$

This gives us the joint distribution of all random variables

$$p(y_1, \ldots, y_N, \boldsymbol{\theta}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \sigma) = p(\boldsymbol{\theta}) \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}, \sigma)\,. \tag{1.64}$$



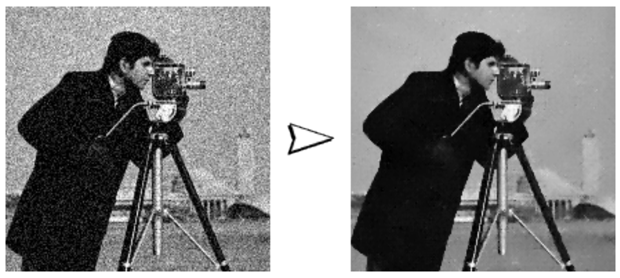
(a) Version 1



(b) Version 2 using the plate notation.

Figure 1.14: Two graphical models for linear regression. Observed random variables are shaded, deterministic parameters are dots. (a) Graphical model without plate notation; (b) Graphical model with plate notation, which allows for a more compact representation than (a).

(a) Online ranking with the TrueSkill system.



(b) Image restoration

Figure 1.15: Examples of message passing using graphical models: (a) Microsoft's TrueSkill system (Herbrich et al., 2007) is used for ranking in online video games. (b) Image restoration (Kittler and Föglein, 1984) is used to remove noise from images.

Now, we follow the steps Section 1.3.1 and find the graphical model in Fig. 1.14(a). Observed random variables are shaded, deterministic parameters are dots, unobserved random variables are "hollow". The graphical model is somewhat repetitive because, and we can write it in a more compact form using the **plate notation** in Fig. 1.14(b). The plate essentially can be read as "for $n = 1, \dots, N$ locally copy/repeat everything inside the plate". Therefore, the plate replaces the dots in Fig. 1.14(a). Note that the parameter σ for the noise and the random variable $\boldsymbol{\theta}$ are "global" and, therefore, outside the plate.

1.3.3 Further Reading

A good and extensive introduction to probabilistic graphical models can be found in the book by Koller and Friedman (2009).

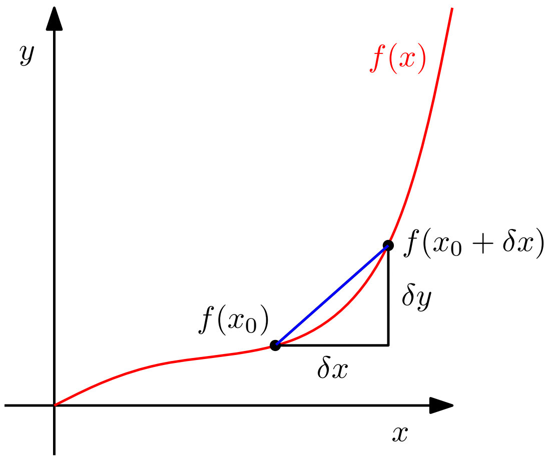


Figure 1.16: The average incline of a function f between x_0 and $x_0 + \delta x$ is the incline of the secant (blue) through $f(x_0)$ and $f(x_0 + \delta x)$ and given by $\delta y / \delta x$.

Directed graphical models allow us to find conditional independence relationship properties of the joint distribution only by looking at the graph. A concept called **d-separation** (Pearl, 1988) is key to this. D-separation will be discussed in more detail in "Data Analysis and Probabilistic Inference" (CO-493).
Graphical models allow for graph-based algorithms for inference and learning, e.g., via local message passing. Applications range from ranking in online games (Herbrich et al., 2007) and computer vision (e.g., image segmentation, semantic labeling, image de-noising, image restoration (Sucar and Gillies, 1994; Shotton et al., 2006; Szeliski et al., 2008; Kittler and Föglein, 1984)) to coding theory (McEliece et al., 1998), solving linear equation systems (Shental et al., 2008) and iterative Bayesian state estimation in signal processing (Bickson et al., 2007; Deisenroth and Mohamed, 2012).

1.4 Vector Calculus

Now, we return to the linear regression setting outlined in Section 1.1: We are interested in finding "good" parameters for the linear regression model. Finding good parameters can be phrased as an optimization problem[15]. We will use gradient-based approaches for solving this optimization problem. Hence, in this section, we will be looking at differentiation of multi-variate functions with respect to parameter vectors.

1.4.1 Introduction: Scalar Differentiation

In the following, we briefly revisit differentiation of a univariate function, i.e., the differentiation we know from school. We start with the difference quotient of a univariate function $y = f(x)$, $x, y \in \mathbb{R}$, which we will subsequently use to define derivatives.

[15]We will do this in Section 1.5.

Definition 6 (Difference Quotient)
The ***difference quotient***

$$\frac{\delta y}{\delta x} = \frac{f(x+\delta x) - f(x)}{\delta x} \tag{1.65}$$

computes the slope of the secant line through two points on the graph of f. These are the points with x-coordinates x and $x + \delta x$, see Fig. 1.16.

The difference quotient can also be considered the average slope of f between x and $x + \delta x$ if we assume a f to be linear. In the limit for $\delta x \to 0$, we obtain the tangent of f at x, if f is differentiable. The tangent is then the derivative of f at x.

Definition 7 (Derivative)
More formally, for $h > 0$ the ***derivative*** *of f at x is defined as the limit*

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}, \tag{1.66}$$

and the secant in Fig. 1.16 becomes a tangent.

Example (Derivative of a Polynomial)
We want to compute the derivative of $f(x) = x^n, n \in \mathbb{N}$. From school, we already know that the answer will be nx^{n-1}, but we want to derive this result using the definition of the gradient as the limit of the difference quotient.
By using (1.66), we obtain

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h} \tag{1.67}$$

$$= \lim_{h\to 0} \frac{(x+h)^n - x^n}{h} \tag{1.68}$$

$$= \lim_{h\to 0} \frac{\sum_{i=0}^{n} \binom{n}{i} x^{n-i} h^i - x^n}{h}. \tag{1.69}$$

We see that $x^n = \binom{n}{0} x^{n-0} h^0$. By starting the sum at 1 the x^n-term cancels, and we obtain

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{h\to 0} \frac{\sum_{i=1}^{n} \binom{n}{i} x^{n-i} h^i}{h} \tag{1.70}$$

$$= \lim_{h\to 0} \sum_{i=1}^{n} \binom{n}{i} x^{n-i} h^{i-1} \tag{1.71}$$

$$= \lim_{h\to 0} \binom{n}{1} x^{n-1} + \underbrace{\sum_{i=2}^{n} \binom{n}{i} x^{n-i} h^{i-1}}_{\to 0 \text{ as } h\to 0} \tag{1.72}$$

$$= \frac{n!}{1!(n-1)!} x^{n-1} = nx^{n-1}. \tag{1.73}$$

1.4.1.1 Taylor Series

The Taylor series is a representation of a function f as an (infinite) sum of terms. These terms are derivatives of f, evaluated at a point x_0.

Definition 8 (Taylor Series)
For a function $f : \mathbb{R} \to \mathbb{R}$, the **Taylor series** *of a smooth*[16] *function $f \in \mathcal{C}^\infty$ at x_0 is defined as*

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \tag{1.74}$$

where $f^{(k)}(x_0)$ is the kth derivative of f at x_0 and $\frac{f^{(k)}(x_0)}{k!}$ are the coefficients of the polynomial. For $x_0 = 0$, we obtain the **Maclaurin series**, *a special instance of the Taylor series.*

Definition 9 (Taylor Polynomial)
The **Taylor polynomial of degree n** *of f at x_0 contains the first $n+1$ components of the series in* (1.74) *and is defined as*

$$T_n = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \,, \tag{1.75}$$

i.e., the Taylor polynomial consists of the first n terms of the Taylor series in (1.74).

Remark 6
In general, a Taylor polynomial of degree n is an approximation of a function. The approximation is best close to x_0. However, a Taylor polynomial of degree n is an exact representation of a polynomial f of degree $k \leq n$ since all derivatives $f^{(i)}$, $i > k$ vanish.

Example (Taylor Polynomial)
We consider the polynomial

$$f(x) = x^4 \tag{1.76}$$

and seek the Taylor polynomial T_6, evaluated at $x_0 = 1$. We start by computing the coefficients $f^{(k)}(1)$ for $k = 0, \ldots, 6$:

$$f(1) = 1 \tag{1.77}$$
$$f'(1) = 4 \tag{1.78}$$
$$f''(1) = 12 \tag{1.79}$$
$$f^{(3)}(1) = 24 \tag{1.80}$$
$$f^{(4)}(1) = 24 \tag{1.81}$$
$$f^{(5)}(1) = 0 \tag{1.82}$$

[16] i.e., $f \in \mathcal{C}^\infty$ (infinitely often continuously differentiable)

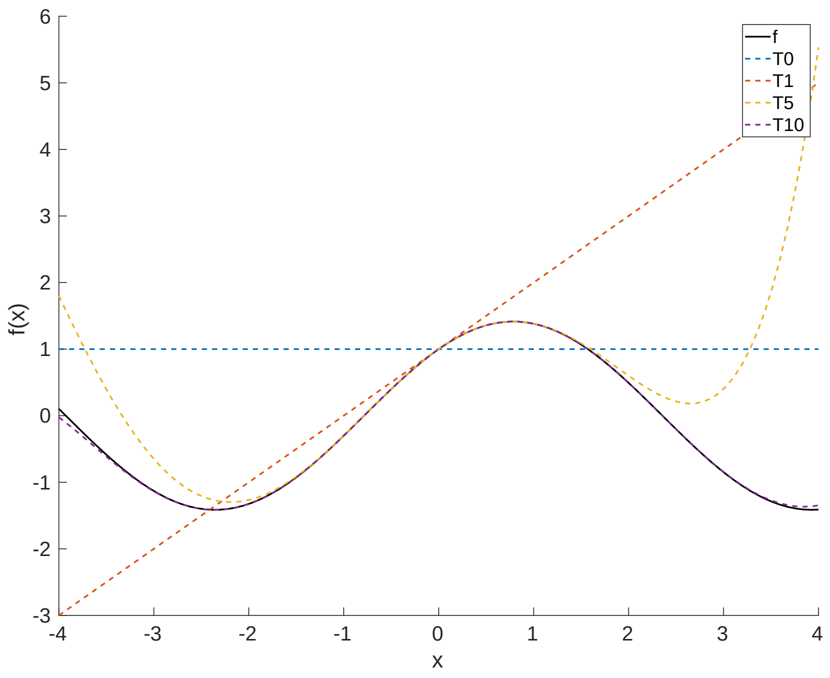


Figure 1.17: Taylor polynomials. The original function $f(x) = \sin(x) + \cos(x)$ (black, solid) is approximated by Taylor polynomials (dashed) around $x_0 = 0$. Higher-order Taylor polynomials approximate the function f better more globally. T_{10} is already very close to f in $[-4, 4]$.

$$f^{(6)}(1) = 0 \tag{1.83}$$

Therefore, the desired Taylor polynomial is

$$T_6(x) = \sum_{k=0}^{6} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k = 1 + 4(x-1) + 6(x-1)^2 + 4(x-1)^3 + (x-1)^4 + 0 \tag{1.84}$$

Multiplying out and re-arranging yields

$$T_6(x) = (1 - 4 + 6 - 4 + 1) + x(4 - 12 + 12 - 4) + x^2(6 - 12 + 6) + x^3(4 - 4) + x^4 \tag{1.85}$$

$$= x^4 = f(x)\,, \tag{1.86}$$

i.e., we obtain an exact representation of the original function.

Example (Taylor Series)

Consider the smooth function

$$f(x) = \sin(x) + \cos(x) \in \mathcal{C}^\infty\,. \tag{1.87}$$

We seek a Taylor series expansion of f at $x_0 = 0$ (which is the Maclaurin series expansion of f).

$$f(0) = \sin(0) + \cos(0) = 1 \tag{1.88}$$
$$f'(0) = \cos(0) - \sin(0) = 1 \tag{1.89}$$
$$f''(0) = -\sin(0) - \cos(0) = -1 \tag{1.90}$$
$$f^{(3)}(0) = -\cos(0) + \sin(0) = -1 \tag{1.91}$$
$$f^{(4)}(0) = \sin(0) + \cos(0) = f(0) = 1 \tag{1.92}$$
$$\vdots \tag{1.93}$$

We can see a pattern here: The coefficients in our Taylor series are only ± 1 (since $\sin(0) = 0$), each of which occurs twice before switching to the other one. Furthermore, $f^{(k+4)}(0) = f^{(k)}(0)$.
Therefore, we can write down the full Taylor series expansion of f at $x_0 = 0$ is given by

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \tag{1.94}$$
$$= 1 + x - \frac{1}{2!}x^2 - \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \frac{1}{5!}x^5 - \cdots \tag{1.95}$$
$$= 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 \mp \cdots + x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 \mp \cdots \tag{1.96}$$
$$= \sum_{k=0}^{\infty} (-1)^k \frac{1}{(2k+1)!} x^{2k+1} + \sum_{k=0}^{\infty} (-1)^k \frac{1}{(2k)!} x^{2k} \tag{1.97}$$
$$= \sin(x) + \cos(x) \tag{1.98}$$

Figure 1.17 shows the corresponding first Taylor polynomials T_n for $n = 0, 1, 5, 10$.

1.4.1.2 Differentiation Rules

We now state basic differentiation rules, where we denote the derivative of f by f'.

Product Rule: $$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \tag{1.99}$$

Quotient Rule: $$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2} \tag{1.100}$$

Sum Rule: $$(f + g)' = f' + g' \tag{1.101}$$

Chain Rule: $$\big(g(f(x))\big)' = (g \circ f)' = g'(f)f' \tag{1.102}$$

1.4.2 Partial Differentiation and Gradients

Ordinary differentiation $\frac{\mathrm{d}f}{\mathrm{d}x}$ applies to functions f of a scalar variable $x \in \mathbb{R}$, see Chapter 1.4.1 for an introduction. In the following, we consider the general case

where the function f depends on one or more variables $\boldsymbol{x} \in \mathbb{R}^n$, e.g., $f(\boldsymbol{x}) = f(x_1, x_2)$. The generalization of the derivative to functions of several variables is the *gradient*.
We find the gradient of the function f with respect to $\boldsymbol{x}$ by *varying one variable at a time* and keeping the others constant. The gradient is then the vector of these *partial derivatives*.

Definition 10 (Partial Derivative)
For a function $f : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{x} \mapsto f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^n$ of n variables $x_1, \dots, x_n$ we define the **partial derivatives** *as*

$$\frac{\partial f}{\partial x_1} = \lim_{h \to 0} \frac{f(x_1 + h, x_2, \dots, x_n) - f(\boldsymbol{x})}{h} \tag{1.103}$$

$$\vdots$$

$$\frac{\partial f}{\partial x_n} = \lim_{h \to 0} \frac{f(x_1, \dots, x_{n-1}, x_n + h) - f(\boldsymbol{x})}{h} \tag{1.104}$$

and collect them in the row vector

$$\nabla_{\boldsymbol{x}} f = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} & \frac{\partial f(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{1 \times n}, \tag{1.105}$$

where n is the number of variables and 1 is the dimension of the image of f.[17] *Here, we used the compact vector notation $\boldsymbol{x} = [x_1, \dots, x_n]^\top$.*

Remark 7 (Gradient as a Row Vector)
It is not uncommon in the literature to define the gradient vector as a column vector, following the convention that vectors are generally column vectors. The reason why we define the gradient vector as a row vector are twofold: First, we can consistently generalize the gradient to a setting where $f : \mathbb{R}^n \to \mathbb{R}^m$ no longer maps onto the real line (then the gradient becomes a matrix). Second, we can immediately apply the chain-rule without paying attention to the dimension of the gradient. We will discuss both points later.

Remark 8 (Verifying the Correctness of the Gradient Implementation)
The definition of the gradient as the limit of the difference quotient can be exploited when checking gradients in computer programs: When we compute gradients and implement them, we can use finite differences to numerically test our computation and implementation: We choose the value h to be small (e.g., $h = 10^{-4}$) and compare the finite-difference approximation from Definition 10 with our (analytic) implementation of the gradient. If the error is small, our gradient implementation is probably correct. "Small" could mean that $\sqrt{\frac{\sum_i (dh_i - df_i)^2}{\sum_i (dh_i + df_i)^2}} < 10^{-3}$, where dh_i is the finite-difference approximation and df_i is the analytic gradient of f with respect to the ith variable x_i.

Example (Partial Derivative)

[17] Note that we currently consider only functions $f : \mathbb{R}^n \to \mathbb{R}^1$.

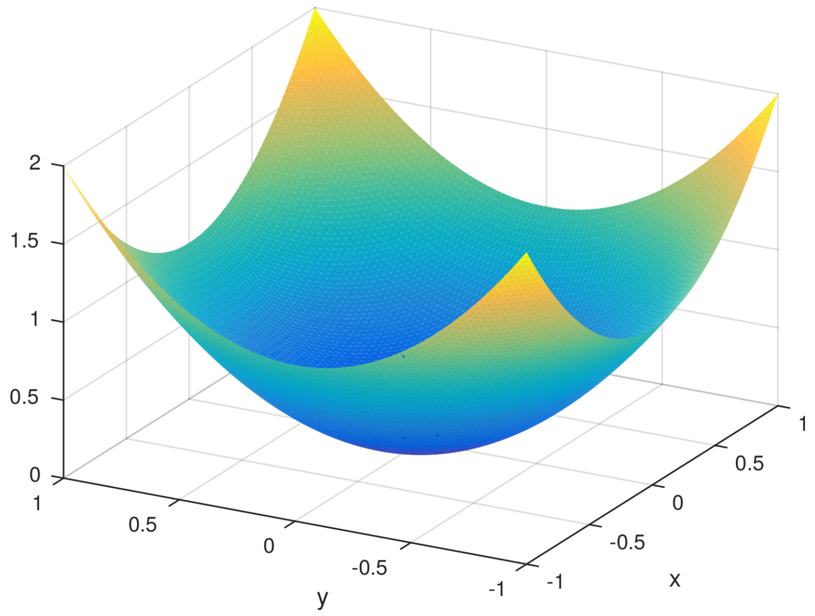


Figure 1.18: $f(x,y) = x^2 + y^2$

For $f(x_1,x_2) = x_1^2 x_2 + x_1 x_2^3 \in \mathbb{R}$, the partial derivatives (i.e., the derivatives of f with respect to x_1 and x_2) are

$$\frac{\partial f(x_1,x_2)}{\partial x_1} = 2x_1x_2 + x_2^3 \tag{1.106}$$

$$\frac{\partial f(x_1,x_2)}{\partial x_2} = x_1^2 + 3x_1x_2^2 \tag{1.107}$$

where ∂ indicates that it is a partial derivative, and the gradient is then

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix}\frac{\partial f(x_1,x_2)}{\partial x_1} & \frac{\partial f(x_1,x_2)}{\partial x_2}\end{bmatrix} = \begin{bmatrix}2x_1x_2 + x_2^3 & x_1^2 + 3x_1x_2^2\end{bmatrix} \in \mathbb{R}^{1\times 2}\,. \tag{1.108}$$

Example
Consider the function $f(x,y) = x^2 + y^2$ (see Fig. 1.18). We obtain the partial derivative $\partial f/\partial x$ by treating y as a constant and computing the derivative of f with respect to x. We then obtain

$$\frac{\partial f(x,y)}{\partial x} = 2x\,. \tag{1.109}$$

Similarly, we obtain the partial derivative of f with respect to y as

$$\frac{\partial f(x,y)}{\partial y} = 2y\,. \tag{1.110}$$

Example
For $f(x, y) = (x + 2y^3)^2$, we obtain the partial derivatives

$$\frac{\partial f(x,y)}{\partial x} = 2(x + 2y^3)\frac{\partial}{\partial x}(x + 2y^3) = 2(x + 2y^3)\,, \tag{1.111}$$

$$\frac{\partial f(x,y)}{\partial y} = 2(x + 2y^3)\frac{\partial}{\partial y}(x + 2y^3) = 12(x + 2y^3)y^2\,. \tag{1.112}$$

1.4.3 Gradients of Vector-Valued Functions (Vector Fields)

Thus far, we discussed (partial) derivatives of functions $f : \mathbb{R}^n \to \mathbb{R}$. In the following, we will generalize the concept of the gradient to vector-valued functions $f : \mathbb{R}^n \to \mathbb{R}^m$, where $n, m \geq 1$.
For $f : \mathbb{R}^n \to \mathbb{R}^m$ and a vector $\boldsymbol{x} = [x_1, \dots, x_n]^\top \in \mathbb{R}^n$, the corresponding vector of function values is given as

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_m(\boldsymbol{x}) \end{bmatrix} \in \mathbb{R}^m\,. \tag{1.113}$$

Writing the vector-valued function in this way allows us to view a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$ as a vector of functions $[f_1, \dots, f_m]^\top$, $f_i : \mathbb{R}^n \to \mathbb{R}$ that map onto $\mathbb{R}$. The differentiation rules for every f_i are exactly the ones we discussed in Section 1.4.2.
Therefore, the partial derivative of a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$ with respect to $x_i \in \mathbb{R}$, $i = 1, \dots n$ is given as the vector

$$\frac{\partial f}{\partial x_i} = \begin{bmatrix} \frac{\partial f_1}{\partial x_i} \\ \vdots \\ \frac{\partial f_m}{\partial x_i} \end{bmatrix} = \begin{bmatrix} \lim_{h\to 0} \frac{f_1(x_1,\dots,x_{i-1},x_i+h,x_{i+1},\dots x_n) - f_1(x)}{h} \\ \vdots \\ \lim_{h\to 0} \frac{f_m(x_1,\dots,x_{i-1},x_i+h,x_{i+1},\dots x_n) - f_m(x)}{h} \end{bmatrix} \in \mathbb{R}^m\,. \tag{1.114}$$

From (1.105), we know that we obtain the gradient of f with respect to a vector as the row vector of the partial derivatives. In (1.114), every partial derivative is a column vector. Therefore, we obtain the gradient of $f : \mathbb{R}^n \to \mathbb{R}^m$ with respect to $\boldsymbol{x} \in \mathbb{R}^n$ as

$$\frac{\mathrm{d}f(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix} \tag{1.115}$$

$$= \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\boldsymbol{x})}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{m\times n}\,. \tag{1.116}$$

Definition 11 (Jacobian)
The matrix (or vector) of all first-order partial derivatives of a vector-valued function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is called the **Jacobian**. *The Jacobian $\boldsymbol{J}$ is an $m \times n$ matrix, which we define and arrange as follows:*

$$\boldsymbol{J} = \nabla_{\boldsymbol{x}} \boldsymbol{f} = \frac{\mathrm{d}\boldsymbol{f}(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\boldsymbol{x})}{\partial x_n} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \tag{1.117}$$

$$J(i,j) = \frac{\partial f_i}{\partial x_j}\,. \tag{1.118}$$

In particular, a function $f : \mathbb{R}^n \to \mathbb{R}^1$, which maps a vector $\boldsymbol{x} = [x_1, \ldots, x_n]^\top \in \mathbb{R}^n$ onto a scalar (e.g., $f(\boldsymbol{x}) = \sum_i x_i$), possesses a Jacobian that is a row vector (matrix of dimension $1 \times n$), see (1.105).

1.4.3.1 Gradients of Matrices

We will encounter situations where we need to take gradients of matrices with respect to vectors (or other matrices), which results in a multi-dimensional tensor. For example, if we compute the gradient of an $m \times n$ matrix with respect to a $p \times q$ matrix, the resulting Jacobian would be $(m \times n) \times (p \times q)$, i.e., a four-dimensional tensor (or array). Since matrices represent linear mappings, we can exploit the fact that there is an isomorphism (linear, invertible mapping) between the space $\mathbb{R}^{m \times n}$ of $m \times n$ matrices and the space $\mathbb{R}^{mn}$ of mn vectors. Therefore, we can re-shape[18] our matrices into vectors of lengths mn and pq, respectively. The gradient using these re-shaped matrices results in a Jacobian of size $mn \times pq$. Figure 1.19 visualizes both approaches. In practical applications, it is often desirable to re-shape the matrix into a vector and continue working with this Jacobian matrix: The chain-rule boils down to simple matrix multiplication, whereas in the case of a Jacobian tensor, we will need to pay more attention to what dimensions we need to sum out.

Example (Gradient of Matrices with Respect to Matrices)
Let us consider a matrix $\boldsymbol{L} \in \mathbb{R}^{m \times n}$ and $f : \mathbb{R}^{m \times n} \to \mathbb{R}^{n \times n}$ with

$$f(\boldsymbol{L}) = \boldsymbol{L}^\top \boldsymbol{L} =: \boldsymbol{K} \in \mathbb{R}^{n \times n}\,. \tag{1.119}$$

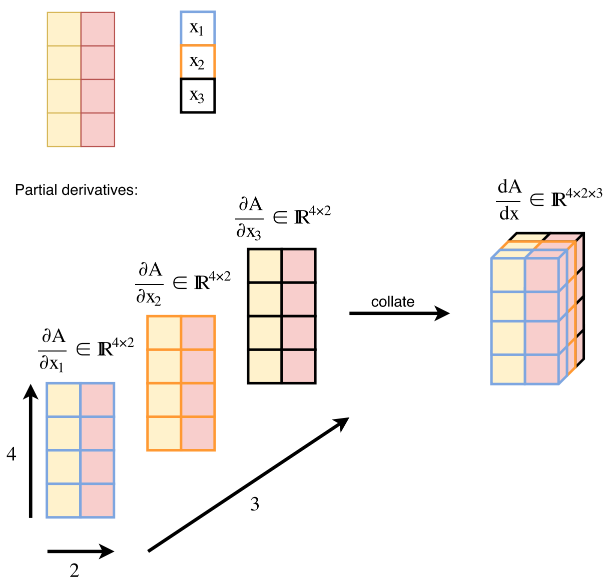
To solve this hard problem, let us first write down what we already know: We know that the gradient has the dimensions

$$\frac{\mathrm{d}\boldsymbol{K}}{\mathrm{d}\boldsymbol{L}} \in \mathbb{R}^{(n \times n) \times (m \times n)}\,, \tag{1.120}$$

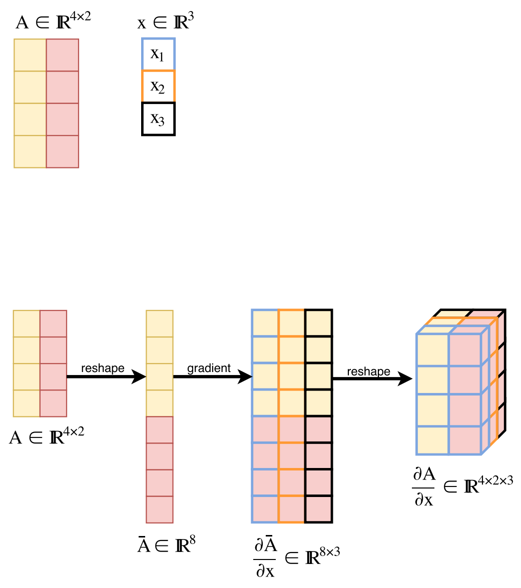
which is a tensor. If we compute the partial derivative of f with respect to a single entry L_{ij}, $i, j \in \{1, \ldots, n\}$, of $\boldsymbol{L}$, we obtain an $m \times m$-matrix

$$\frac{\partial \boldsymbol{K}}{\partial L_{ij}} \in \mathbb{R}^{n \times n}\,. \tag{1.121}$$

[18] e.g., by stacking the columns of the matrix ("flatten")



(a) Approach 1: We compute the partial derivative $\frac{\partial \boldsymbol{A}}{\partial x_1}, \frac{\partial \boldsymbol{A}}{\partial x_2}, \frac{\partial \boldsymbol{A}}{\partial x_3}$, each of which is a 4×2 matrix, and collate them in a $4 \times 2 \times 3$ tensor.



(b) Approach 2: We re-shape (flatten) $\boldsymbol{A} \in \mathbb{R}^{4\times 2}$ into a vector $\tilde{\boldsymbol{A}} \in \mathbb{R}^{8}$. Then, we compute the gradient $\frac{\mathrm{d}\tilde{\boldsymbol{A}}}{\mathrm{d}\boldsymbol{x}} \in \mathbb{R}^{8\times 3}$. We obtain the gradient tensor by re-shaping this gradient as illustrated above.

Figure 1.19: Visualization of gradient computation of a matrix with respect to a vector. We are interested in computing the gradient of $\boldsymbol{A} \in \mathbb{R}^{4\times 2}$ with respect to a vector $\boldsymbol{x} \in \mathbb{R}^3$. We know that gradient $\frac{\mathrm{d}\boldsymbol{A}}{\mathrm{d}\boldsymbol{x}} \in \mathbb{R}^{4\times 2\times 3}$. We follow two equivalent approaches to arrive there.

Furthermore, we know that

$$\frac{\mathrm{d}K_{pq}}{\mathrm{d}\boldsymbol{L}} \in \mathbb{R}^{m\times n} \tag{1.122}$$

for $p, q = 1, \dots, n$, where $K_{pq} = f_{pq}(\boldsymbol{L})$ is the (p, q)-th entry of $\boldsymbol{K} = f(\boldsymbol{L})$. Denoting the i-th column of $\boldsymbol{L}$ by $\boldsymbol{l}_i$, we see that every entry of $\boldsymbol{K}$ is given by an inner product of two columns of $\boldsymbol{L}$, i.e.,

$$K_{pq} = \boldsymbol{l}_p^\top \boldsymbol{l}_q = \sum_{k=1}^{n} L_{kp} L_{kq} \,. \tag{1.123}$$

When we now compute the partial derivative $\frac{\partial K_{pq}}{\partial L_{ij}}$, we obtain

$$\frac{\partial K_{pq}}{\partial L_{ij}} = \sum_{k=1}^{n} \frac{\partial}{\partial L_{ij}} L_{kp} L_{kq} = \partial_{pqij} \,, \tag{1.124}$$

$$\partial_{pqij} = \begin{cases} L_{iq} & \text{if } j = p,\ p \neq q \\ L_{ip} & \text{if } j = q,\ p \neq q \\ 2L_{iq} & \text{if } j = p,\ p = q \\ 0 & \text{otherwise} \end{cases} \tag{1.125}$$

From (1.120), we know that the desired gradient has the dimension $(n\times n)\times(m\times n)$, end every single entry of this tensor is given by ∂_{pqij} in (1.125), where $p,q,j = 1,\dots,n$ and $i=q,\dots,m$.

1.4.4 Basic Rules of Differentiation

In the multivariate case, where $\boldsymbol{x}\in\mathbb{R}^n$, the basic differentiation rules that we know from school (sum rule, product rule, chain rule)[19] still apply. However, we need to pay attention because now we have to deal with matrices where multiplication is no longer commutative, i.e., the order matters.

$$\text{Product Rule:}\quad \frac{\partial}{\partial \boldsymbol{x}}\big(f(\boldsymbol{x})g(\boldsymbol{x})\big) = \frac{\partial f}{\partial \boldsymbol{x}}g(\boldsymbol{x}) + f(\boldsymbol{x})\frac{\partial g}{\partial \boldsymbol{x}} \tag{1.126}$$

$$\text{Sum Rule:}\quad \frac{\partial}{\partial \boldsymbol{x}}\big(f(\boldsymbol{x})+g(\boldsymbol{x})\big) = \frac{\partial f}{\partial \boldsymbol{x}} + \frac{\partial g}{\partial \boldsymbol{x}} \tag{1.127}$$

$$\text{Chain Rule:}\quad \frac{\partial}{\partial \boldsymbol{x}}(g\circ f)(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}}\big(g(f(\boldsymbol{x}))\big) = \frac{\partial g}{\partial f}\frac{\partial f}{\partial \boldsymbol{x}} \tag{1.128}$$

Before we have a closer look at the chain rule in Section 1.4.5, we state some useful identities for computing gradients that are frequently required in a machine learning context (Petersen and Pedersen, 2012):

$$\frac{\partial}{\partial \boldsymbol{X}} f(\boldsymbol{X})^\top = \left(\frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}}\right)^\top \tag{1.129}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \mathrm{tr}(f(\boldsymbol{X})) = \mathrm{tr}\left(\frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}}\right) \tag{1.130}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \det(f(\boldsymbol{X})) = \det(f(\boldsymbol{X}))\mathrm{tr}\left(f^{-1}(\boldsymbol{X})\frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}}\right) \tag{1.131}$$

$$\frac{\partial}{\partial \boldsymbol{X}} f^{-1}(\boldsymbol{X}) = -f^{-1}(\boldsymbol{X})\frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}}f^{-1}(\boldsymbol{X}) \tag{1.132}$$

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{X}^{-1}\boldsymbol{b}}{\partial \boldsymbol{X}} = -(\boldsymbol{X}^{-1})^\top \boldsymbol{a}\boldsymbol{b}^\top (\boldsymbol{X}^{-1})^\top \tag{1.133}$$

$$\frac{\partial \boldsymbol{x}^\top \boldsymbol{a}}{\partial \boldsymbol{x}} = \boldsymbol{a}^\top \tag{1.134}$$

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{a}^\top \tag{1.135}$$

$$\frac{\partial \boldsymbol{a}^\top \boldsymbol{X}\boldsymbol{b}}{\partial \boldsymbol{X}} = \boldsymbol{a}\boldsymbol{b}^\top \tag{1.136}$$

$$\frac{\partial \boldsymbol{x}^\top \boldsymbol{B}\boldsymbol{x}}{\partial \boldsymbol{x}} = \boldsymbol{x}^\top(\boldsymbol{B}+\boldsymbol{B}^\top) \tag{1.137}$$

$$\frac{\partial}{\partial \boldsymbol{s}}(\boldsymbol{x}-\boldsymbol{A}\boldsymbol{s})^\top \boldsymbol{W}(\boldsymbol{x}-\boldsymbol{A}\boldsymbol{s}) = -2(\boldsymbol{x}-\boldsymbol{A}\boldsymbol{s})^\top \boldsymbol{W}\boldsymbol{A} \quad \text{for symmetric } \boldsymbol{W} \tag{1.138}$$

[19]product rule: $(fg)' = f'g + fg'$, sum rule: $(f+g)' = f' + g'$, chain rule: $(g\circ f)' = g'(f)f'$

1.4.5 Chain Rule

Let us have a closer look at the chain rule. Consider a function $f : \mathbb{R}^2 \to \mathbb{R}$ of two variables x_1, x_2. Furthermore, x_1 and x_2 are themselves functions of t. To compute the gradient of f with respect to t, we need to apply the chain rule (1.128) for multivariate functions as

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\mathrm{d}x_1}{\mathrm{d}t} \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} \end{bmatrix} = \frac{\partial f}{\partial x_1}\frac{\mathrm{d}x_1}{\mathrm{d}t} + \frac{\partial f}{\partial x_2}\frac{\mathrm{d}x_2}{\mathrm{d}t} \tag{1.139}$$

where d denotes the **total derivative**.

Example
Consider $f(x_1, x_2) = x_1^2 + 2x_2$, where $x_1 = \sin t$ and $x_2 = \cos t$, then

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x_1}\frac{\mathrm{d}x_1}{\mathrm{d}t} + \frac{\partial f}{\partial x_2}\frac{\mathrm{d}x_2}{\mathrm{d}t} \tag{1.140}$$

$$= 2\sin t \frac{\mathrm{d}\sin t}{\mathrm{d}t} + 2\frac{\mathrm{d}\cos t}{\mathrm{d}t} \tag{1.141}$$

$$= 2\sin t \cos t - 2\sin t = 2\sin t(\cos t - 1) \tag{1.142}$$

If $f(x_1, x_2)$ is a function of x_1 and x_2, where $x_1(s,t)$ and $x_2(s,t)$ are themselves functions of two variables s and t, the chain rule yields

$$\frac{\mathrm{d}f}{\mathrm{d}s} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial s} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial s}, \tag{1.143}$$

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t}, \tag{1.144}$$

which can be expressed as a matrix equation

$$\frac{\mathrm{d}f}{\mathrm{d}(s,t)} = \frac{\partial f}{\partial \boldsymbol{x}}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}(s,t)} = \underbrace{\begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix}}_{=\frac{\partial f}{\partial \boldsymbol{x}}} \underbrace{\begin{bmatrix} \frac{\partial x_1}{\partial s} & \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial s} & \frac{\partial x_2}{\partial t} \end{bmatrix}}_{=\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}(s,t)}} \tag{1.145}$$

Example
Consider the function $h : \mathbb{R} \to \mathbb{R}$, $h(t) = (f \circ g)(t)$ with

$$f : \mathbb{R}^2 \to \mathbb{R} \tag{1.146}$$

$$g : \mathbb{R} \to \mathbb{R}^2 \tag{1.147}$$

$$f(\boldsymbol{x}) = \exp(x_1 x_2^2), \tag{1.148}$$

$$\boldsymbol{x} = g(t) = \begin{bmatrix} t\cos t \\ t\sin t \end{bmatrix} \tag{1.149}$$

and compute the gradient of h with respect to t.
Since $f : \mathbb{R}^2 \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}^2$ we note that

$$\frac{\partial f}{\partial \boldsymbol{x}} \in \mathbb{R}^{1\times 2}, \tag{1.150}$$

$$\frac{\partial g}{\partial t} \in \mathbb{R}^{2\times 1}\,. \tag{1.151}$$

The desired gradient is computed by applying the chain-rule:

$$\begin{aligned}
\frac{\mathrm{d}h}{\mathrm{d}t} &= \frac{\partial f}{\partial \boldsymbol{x}}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} && (1.152)\\
&= \begin{bmatrix}\frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2}\end{bmatrix}\begin{bmatrix}\frac{\mathrm{d}x_1}{\mathrm{d}t}\\ \frac{\mathrm{d}x_2}{\mathrm{d}t}\end{bmatrix} && (1.153)\\
&= \begin{bmatrix}\exp(x_1x_2^2)x_2^2 & 2\exp(x_1x_2^2)x_1x_2\end{bmatrix}\begin{bmatrix}\cos t - t\sin t\\ \sin t + t\cos t\end{bmatrix} && (1.154)\\
&= \exp(x_1x_2^2)\big(x_2^2(\cos t - t\sin t) + 2x_1x_2(\sin t + t\cos t)\big) && (1.155)
\end{aligned}$$

Example (Linear Least-Squares Regression)
Let us consider the linear regression model

$$\boldsymbol{y} = \boldsymbol{\Phi\theta}\,, \qquad \boldsymbol{y}\in\mathbb{R}^N,\ \boldsymbol{\theta}\in\mathbb{R}^D\,, \tag{1.156}$$

and we define the following functions:

$$\begin{aligned}
L(\boldsymbol{e}) &:= \|\boldsymbol{e}\|^2\,, && (1.157)\\
\boldsymbol{e}(\boldsymbol{\theta}) &:= \boldsymbol{y} - \boldsymbol{\Phi\theta}\,. && (1.158)
\end{aligned}$$

We seek $\frac{\partial L}{\partial \boldsymbol{\theta}}$, and we will use the chain rule for this purpose.
Before we start any calculation, we determine the dimensionality of the gradient as

$$\frac{\partial L}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{1\times D}\,. \tag{1.159}$$

The chain rule allows us to compute the gradient as

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \frac{\partial L}{\partial \boldsymbol{e}}\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}}\,. \tag{1.160}$$

The dot product yields $\|\boldsymbol{e}\|^2 = \boldsymbol{e}^\top\boldsymbol{e}$ (see Appendix A.2), and by exploiting the product rule and (1.134) we determine

$$\frac{\partial L}{\partial \boldsymbol{e}} = 2\boldsymbol{e}^\top \in \mathbb{R}^{1\times N}\,. \tag{1.161}$$

Furthermore, we obtain

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}} = -\boldsymbol{\Phi} \in \mathbb{R}^{N\times D}\,, \tag{1.162}$$

such that our desired derivative is

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = -2\boldsymbol{e}^\top\boldsymbol{\Phi} = -\underbrace{2(\boldsymbol{y}^\top - \boldsymbol{\theta}^\top\boldsymbol{\Phi}^\top)}_{1\times N}\underbrace{\boldsymbol{\Phi}}_{N\times D} \in \mathbb{R}^{1\times D}\,. \tag{1.163}$$

Remark 9
We would have obtained the same result without using the chain rule by immediately looking at the function

$$L_2(\boldsymbol{\theta}) := \|\boldsymbol{y} - \boldsymbol{\Phi\theta}\|^2 = (\boldsymbol{y} - \boldsymbol{\Phi\theta})^\top (\boldsymbol{y} - \boldsymbol{\Phi\theta}) . \tag{1.164}$$

This approach is still practical for simple functions like L_2 but becomes impractical if consider deep function compositions.

Remark 10
When we look at deep function compositionsnested $f_K \circ f_{K-1} \circ \cdots \circ f_1$, writing out the full function is tedious. Furthermore, for programming purposes the chain rule is extremely useful: When we write functions/methods for ever f_i that return the partial derivative of its outputs with respect to its inputs, the total derivative with respect to the input is just the product of the partial derivatives returned by the individual functions. If we then decide modify f_i into $\tilde{f}_i$, we simply have to write a function that computes the partial derivative of $\tilde{f}_i$ and use this in the product of partial derivatives (instead of re-deriving the total derivative from scratch).

Remark 11 (Application of the Chain Rule in Machine Learning)
In machine learning, the chain rule plays an important role when optimizing parameters of a hierarchical model (e.g., for maximum likelihood estimation). An area where the chain rule is used to an extreme is Deep Learning where the function value y is computed as a nested/layered function

$$y = f_K(f_{K-1}(\cdots(f_1(\boldsymbol{x}))\cdots)) , \tag{1.165}$$

where $\boldsymbol{x}$ are the inputs (e.g., images), y are the observations (e.g., class labels) and every function f_i, $i = 1, \dots, K$ possesses its own parameters. In neural networks with multiple layers, we have functions $f_i(\boldsymbol{x}) = \sigma(\boldsymbol{A}_i\boldsymbol{x}_{i-1} + \boldsymbol{b}_i)$ in the ith layer, where $\boldsymbol{x}_{i-1}$ is the output of layer $i-1$ and σ an activation function, e.g., the logistic sigmoid $\frac{1}{1+e^{-x}}$, $\tanh$ or a rectified linear unit (ReLU). In order to train these models, we have to compute the gradient of a loss function with respect to the inputs of each layer (e.g., $\boldsymbol{x}_{i-1}$) to obtain the partial derivative with respect to the parameters of the previous layer (e.g., $\boldsymbol{A}_{i-1}, \boldsymbol{b}_{i-1}$). There are efficient ways of implementing this repeated application of the chain-rule using **backpropagation** *(Kelley, 1960; Bryson, 1961; Dreyfus, 1962; Rumelhart et al., 1986).*

1.4.6 Higher-Order Derivatives

Notation Consider a function $f : \mathbb{R}^2 \to \mathbb{R}$ of two variables x, y. Multiple partial derivatives (as for ordinary derivatives) are expressed as

- $\frac{\partial^2 f}{\partial x^2}$ is the second partial derivative of f with respect to x

- $\frac{\partial^n f}{\partial x^n}$ is the nth partial derivative of f with respect to x
- $\frac{\partial^2 f}{\partial x \partial y}$ is the partial derivative obtained by first partial differentiating by y and then x
- $\frac{\partial^2 f}{\partial y \partial x}$ is the partial derivative obtained by first partial differentiating by x and then y

If $f(x, y)$ is a twice (continuously) differentiable function then

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x}, \tag{1.166}$$

i.e., the order of differentiation does not matter, and the corresponding **Hessian matrix** (the matrix of second partial derivatives)

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \tag{1.167}$$

is symmetric. Generally, for $\boldsymbol{x} \in \mathbb{R}^D$, the Hessian is a $D \times D$ matrix. The Hessian measures the local geometry of curvature.

Remark 12 (Hessian of a Vector Field)
If $f : \mathbb{R}^n \to \mathbb{R}^m$ is a vector field, the Hessian is an $(m \times n \times n)$-tensor.

1.4.7 Linearization and Taylor Series

The gradient ∇f of a function f is often used for a locally linear approximation of f around $\boldsymbol{x}_0$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_0) + (\nabla_{\boldsymbol{x}} f)(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0). \tag{1.168}$$

Here $(\nabla_{\boldsymbol{x}} f)(\boldsymbol{x}_0)$ is the gradient of f with respect to $\boldsymbol{x}$, evaluated at $\boldsymbol{x}_0$. Note that (1.168) is equivalent to the first two terms in the **multi-variate Taylor-series expansion** of f at $\boldsymbol{x}_0$.

Definition 12 (Multivariate Taylor Series)
For the **multivariate Taylor series***, we consider a function*

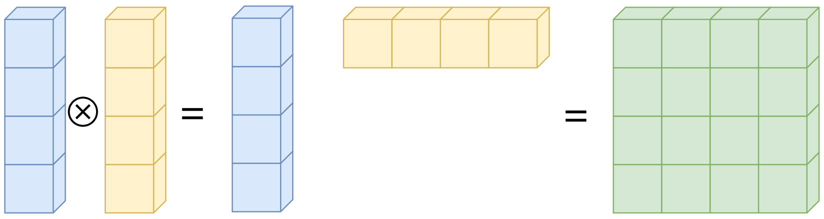
$$\begin{aligned} f : \mathbb{R}^D &\to \mathbb{R} \qquad (1.169) \\ \boldsymbol{x} &\mapsto f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^D, \qquad (1.170) \end{aligned}$$

that is smooth at $\boldsymbol{x}_0$.
When we define $\delta := \boldsymbol{x} - \boldsymbol{x}_0$, the Taylor series of f at $(\boldsymbol{x}_0)$ is defined as

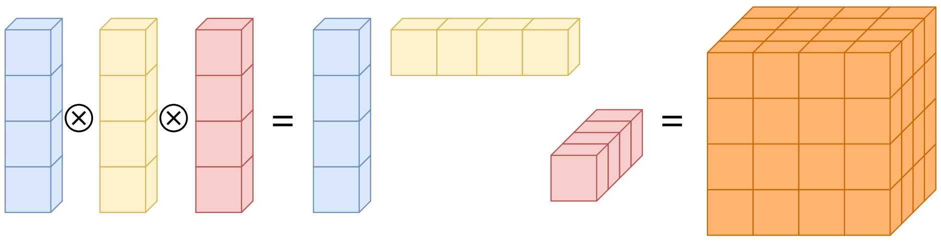
$$f(\boldsymbol{x}) = \sum_{k=0}^{\infty} \frac{D_{\boldsymbol{x}}^k f(\boldsymbol{x}_0)}{k!} \boldsymbol{\delta}^k, \tag{1.171}$$

where $D_x^k f(\boldsymbol{x}_0)$ is the k-th (total) derivative of f with respect to $\boldsymbol{x}$, evaluated at $\boldsymbol{x}_0$. The **Taylor polynomial of degree** n *of f at $\boldsymbol{x}_0$ contains the first $n+1$ components of the series in* (1.171) *and is defined as*

$$T_n = \sum_{k=0}^{n} \frac{D_{\boldsymbol{x}}^k f(\boldsymbol{x}_0)}{k!} \boldsymbol{\delta}^k \,. \tag{1.172}$$



(a) Given a vector $\boldsymbol{\delta} \in \mathbb{R}^4$, we obtain the outer product $\delta^2 := \boldsymbol{\delta} \otimes \delta = \boldsymbol{\delta}\boldsymbol{\delta}^\top \in \mathbb{R}^{4\times 4}$ as a matrix.



(b) An outer product $\delta^3 := \boldsymbol{\delta} \otimes \boldsymbol{\delta} \otimes \boldsymbol{\delta} \in \mathbb{R}^{4\times 4\times 4}$ results in a third-order tensor ("three-dimensional matrix"), i.e., an array with three indexes.

Figure 1.20: Visualizing outer products. Outer products of vectors increase the dimensionality of the array by 1 per term.

Remark 13 (Notation)

In (1.171) *and* (1.172)*, we used the slightly sloppy notation of $\boldsymbol{\delta}^k$, which is not defined for vectors $\boldsymbol{x} \in \mathbb{R}^D$, $D > 1$, and $k > 1$. Note that both $D_x^k f$ and $\boldsymbol{\delta}^k$ are k-th order tensors, i.e., k-dimensional arrays.*[20] *The k-th order tensor $\boldsymbol{\delta}_k \in \mathbb{R}^{\overbrace{D \times D \times \ldots \times D}^{k \text{ times}}}$ is obtained as a k-fold outer product, denoted by $\otimes$, of the vector $\boldsymbol{\delta} \in \mathbb{R}^D$. For example,*

$$\boldsymbol{\delta}^2 = \boldsymbol{\delta} \otimes \boldsymbol{\delta} = \boldsymbol{\delta}\boldsymbol{\delta}^\top \,, \quad \boldsymbol{\delta}^2[i,j] = \delta[i]\delta[j] \tag{1.173}$$

$$\boldsymbol{\delta}^3 = \boldsymbol{\delta} \otimes \boldsymbol{\delta} \otimes \boldsymbol{\delta} \,, \quad \boldsymbol{\delta}^3[i,j,k] = \delta[i]\delta[j]\delta[k] \,. \tag{1.174}$$

Figure 1.20 visualizes two such outer products. In general, we obtain the following terms in the Taylor series:

$$D_x^k f(\boldsymbol{x}_0)\boldsymbol{\delta}^k = \sum_a \cdots \sum_k D_x^k f(\boldsymbol{x}_0)[a,\ldots,k]\delta[a]\cdots\delta[k] \,, \tag{1.175}$$

where $D_x^k f(\boldsymbol{x}_0)\boldsymbol{\delta}^k$ contains k-th order polynomials.

[20] A vector can be implemented as a 1-dimensional array, a matrix as a 2-dimensional array.

Now that we defined the Taylor series for vector fields, let us explicitly write down the first terms $D_x^k f(\boldsymbol{x}_0)\boldsymbol{\delta}^k$ of the Taylor series expansion for $k = 0, \dots, 3$ and $\boldsymbol{\delta} := \boldsymbol{x} - \boldsymbol{x}_0$:

$$k = 0 : D_x^0 f(\boldsymbol{x}_0)\boldsymbol{\delta}^0 = f(\boldsymbol{x}_0) \in \mathbb{R} \tag{1.176}$$

$$k = 1 : D_x^1 f(\boldsymbol{x}_0)\boldsymbol{\delta}^1 = \underbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0)}_{1\times D}\underbrace{\boldsymbol{\delta}}_{D\times 1} = \sum_i \nabla_x f(\boldsymbol{x}_0)[i]\delta[i] \in \mathbb{R} \tag{1.177}$$

$$k = 2 : D_x^2 f(\boldsymbol{x}_0)\boldsymbol{\delta}^2 = tr\big(\underbrace{\boldsymbol{H}}_{D\times D}\underbrace{\boldsymbol{\delta}}_{D\times 1}\underbrace{\boldsymbol{\delta}^\top}_{1\times D}\big) = \boldsymbol{\delta}^\top \boldsymbol{H}\delta = \sum_i \sum_j H[i,j]\delta[i]\delta[j] \in \mathbb{R} \tag{1.178}$$

$$k = 3 : D_x^3 f(\boldsymbol{x}_0)\boldsymbol{\delta}^3 = \sum_i \sum_j \sum_k D_x^3 f(\boldsymbol{x}_0)[i,j,k]\delta[i]\delta[j]\delta[k] \in \mathbb{R} \tag{1.179}$$

Example (Taylor-Series Expansion of a Function with Two Variables)
Consider the function

$$f(x,y) = x^2 + 2xy + y^3\,. \tag{1.180}$$

We want to compute the Taylor series expansion of f at $(x_0, y_0) = (1,2)$. Before we start, let us discuss what to expect: The function in (1.180) is a polynomial of degree 3. We are looking for a Taylor series expansion, which itself is a linear combination of polynomials. Therefore, we do not expect the Taylor series expansion to contain terms of fourth or higher order to express a third-order polynomial. This means, it should be sufficient to determine the first four terms of (1.171) for an exact alternative representation of (1.180).
To determine the Taylor series expansion, start of with the constant term and the first-order derivatives, which are given by

$$f(1,2) = 13 \tag{1.181}$$

$$\frac{\partial f}{\partial x} = 2x + 2y \Rightarrow \frac{\partial f}{\partial x}(1,2) = 6 \tag{1.182}$$

$$\frac{\partial f}{\partial y} = 2x + 3y^2 \Rightarrow \frac{\partial f}{\partial y}(1,2) = 14\,. \tag{1.183}$$

$$\Rightarrow D_{x,y}^1 f(1,2) = \nabla_{x,y} f(1,2) = \begin{bmatrix}\frac{\partial f}{\partial x}(1,2) & \frac{\partial f}{\partial y}(1,2)\end{bmatrix} = \begin{bmatrix}6 & 14\end{bmatrix} \in \mathbb{R}^{1\times 2} \tag{1.184}$$

$$\Rightarrow \frac{D_{x,y}^1 f(1,2)}{1!}\boldsymbol{\delta} = \begin{bmatrix}6 & 14\end{bmatrix}\begin{bmatrix}x-1\\ y-2\end{bmatrix} = 6(x-1) + 14(y-2)\,. \tag{1.185}$$

Note that $D_{x,y}^1 f(1,2)\boldsymbol{\delta}$ contains only linear terms, i.e., first-order polynomials.
The second-order partial derivatives are given by

$$\frac{\partial^2 f}{\partial x^2} = 2 \Rightarrow \frac{\partial^2 f}{\partial x^2}(1,2) = 2 \tag{1.186}$$

$$\frac{\partial^2 f}{\partial y^2} = 6y \Rightarrow \frac{\partial^2 f}{\partial y^2}(1,2) = 12 \tag{1.187}$$

$$\frac{\partial f^2}{\partial x \partial y} = 2 \Rightarrow \frac{\partial f^2}{\partial x \partial y}(1,2) = 2 \tag{1.188}$$

$$\frac{\partial f^2}{\partial y \partial x} = 2 \Rightarrow \frac{\partial f^2}{\partial y \partial x}(1,2) = 2\,. \tag{1.189}$$

When we collect the second-order partial derivatives, we obtain the Hessian

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial f^2}{\partial x \partial y} \\ \frac{\partial f^2}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 2 & 6y \end{bmatrix} \tag{1.190}$$

$$\Rightarrow \boldsymbol{H}(1,2) = \begin{bmatrix} 2 & 2 \\ 2 & 12 \end{bmatrix} \in \mathbb{R}^{2\times 2}\,, \tag{1.191}$$

such that the next term of the Taylor-series expansion is given by

$$\frac{D^2_{x,y} f(1,2)}{2!}\boldsymbol{\delta}^2 = \frac{1}{2}\boldsymbol{\delta}^2 \boldsymbol{H}(1,2)\boldsymbol{\delta} \tag{1.192}$$

$$= \begin{bmatrix} x-1 & y-2 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 12 \end{bmatrix} \begin{bmatrix} x-1 \\ y-2 \end{bmatrix} \tag{1.193}$$

$$= (x-1)^2 + 2(x-1)(y-2) + 6(y-2)^2 \tag{1.194}$$

Here, $D^2_{x,y} f(1,2)\boldsymbol{\delta}^2$ contains only quadratic terms, i.e., second-order polynomials.
The third-order derivatives are obtained as

$$D^3_x f = \begin{bmatrix} \frac{\partial \boldsymbol{H}}{\partial x} & \frac{\partial \boldsymbol{H}}{\partial y} \end{bmatrix} \in \mathbb{R}^{2\times 2\times 2}\,, \tag{1.195}$$

$$D^3_x f[:,:,1] = \frac{\partial H}{\partial x} = \begin{bmatrix} \frac{\partial^3 f}{\partial x^3} & \frac{\partial^3 f}{\partial x \partial y \partial x} \\ \frac{\partial^3 f}{\partial y \partial x^2} & \frac{\partial^3 f}{\partial y^2 \partial x} \end{bmatrix}\,, \tag{1.196}$$

$$D^3_x f[:,:,2] = \frac{\partial H}{\partial y} = \begin{bmatrix} \frac{\partial^3 f}{\partial x^2 \partial y} & \frac{\partial^3 f}{\partial x \partial y^2} \\ \frac{\partial^3 f}{\partial y \partial x \partial y} & \frac{\partial^3 f}{\partial y^3}\,. \end{bmatrix}\,. \tag{1.197}$$

Since most second-order partial derivatives in the Hessian in (1.190) are constant the only non-zero third-order partial derivative is

$$\frac{\partial^3 f}{\partial y^3} = 6 \Rightarrow \frac{\partial^3 f}{\partial y^3}(1,2) = 6\,. \tag{1.198}$$

Higher-order derivatives and the mixed derivatives of degree 3 (e.g., $\frac{\partial f^3}{\partial x^2 \partial y}$) vanish, such that

$$D^3_{x,y} f[:,:,1] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\,, \quad D^3_{x,y} f[:,:,2] = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{1.199}$$

and

$$\frac{D^3_{x,y} f(1,2)}{3!}\boldsymbol{\delta}^3 = \frac{(y-2)^3}{6}\,, \tag{1.200}$$

which collects all cubic terms (third-order polynomials) of the Taylor series.
Overall, the (exact) Taylor series expansion of f at $(x_0, y_0) = (1,2)$ is

$$f(x) = f(1,2) + D^1_{x,y} f(1,2)\boldsymbol{\delta} + \frac{D^2_{x,y} f(1,2)}{2!}\boldsymbol{\delta}^2 + \frac{D^3_{x,y} f(1,2)}{3!}\boldsymbol{\delta}^3 \tag{1.201}$$

$$
\begin{aligned}
&= f(1,2) && (1.202)\\
&\quad + \frac{\partial f(1,2)}{\partial x}(x-1) + \frac{\partial f(1,2)}{\partial y}(y-2) && (1.203)\\
&\quad + \frac{1}{2!}\left(\frac{\partial^2 f(1,2)}{\partial x^2}(x-1)^2 + \frac{\partial^2 f(1,2)}{\partial y^2}(y-2)^2 + 2\frac{\partial^2 f(1,2)}{\partial x \partial y}(x-1)(y-2)\right) && (1.204)\\
&\quad + \frac{1}{6}\frac{\partial^3 f(1,2)}{\partial y^3}(y-2)^3 && (1.205)\\
&= 13 + 6(x-1) + 14(y-2) + (x-1)^2 + 6(y-2)^2 + 2(x-1)(y-2) + (y-2)^3 \,. && (1.206)
\end{aligned}
$$

In this case, we obtained an exact Taylor series expansion of the polynomial in (1.180), i.e., the polynomial in (1.206) is equivalent to the original polynomial in (1.180). In this particular example, this result is not surprising since the original function was a third-order polynomial, which we expressed through a linear combination of constant terms, first-order, second order and third-order polynomials in (1.206).

1.4.8 Gradient Checking

When we implement gradients for learning and optimization, we need to verify that our gradients are correct and correctly implemented. One way to check the gradients is to compare our analytical solution that we implemented with a finite-difference approximation. The **finite-differences** approximation of the gradient exploits the definition of the gradient (1.66)

$$f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta} \,. \tag{1.207}$$

The finite-difference approximation approximates the gradient by choosing a small value for δ (e.g., 10^{-4} and ignoring the limit, such that an approximation of the gradient is

$$f'(x) \approx \frac{f(x+\delta) - f(x)}{\delta} \tag{1.208}$$

The **central difference** approximation of the gradient is numerically more stable, but approximately twice as expensive to compute. This approximation is given by

$$f'(x) \approx \frac{f(x+\frac{1}{2}\delta) - f(x)}{\delta} + \frac{f(x) - f(x-\frac{1}{2}\delta)}{\delta} = \frac{f(x+\frac{1}{2}\delta) - f(x-\frac{1}{2}\delta)}{\delta} \tag{1.209}$$

Intuitively, central differences looks $\frac{\delta}{2}$ to the left and right of the point x_0 where we wish to evaluate the gradient at, which is also illustrated in Figure 1.21. In contrast, the finite difference approximation (1.208) computes only a one-sided gradient. However, central differences in (1.209) requires to evaluate the function f twice to obtain the gradient.

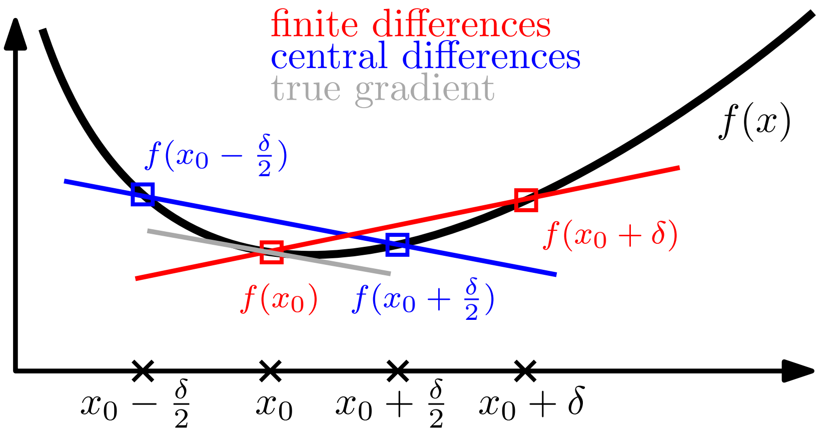


Figure 1.21: Finite and central difference approximation of the gradient. The gradient $f'(x_0)$ is shown in gray. The finite-difference approximation is shown in red, the central differences approximation is shown in blue. In this illustration, only the slope is important. Central differences yields a better approximation to the ground truth than finite differences.

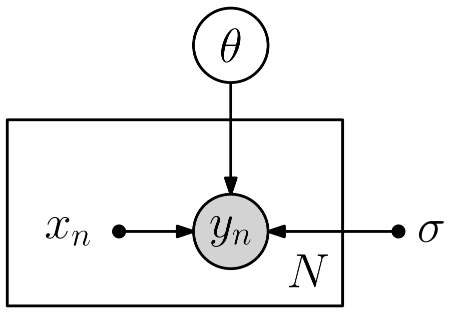


Figure 1.22: Probabilistic graphical model for linear regression.

If $\boldsymbol{x} \in \mathbb{R}^D$, we compute the finite/central difference approximation for every partial derivative. Therefore, require $2D + 1$ function evaluations for central differences compared to $D + 1$ for finite differences.

1.5 Parameter Estimation

Assume we are given a **training set** $\mathcal{D}$ consisting of N inputs $\boldsymbol{x}_i \in \mathbb{R}^D$ and corresponding observations $y_i \in \mathbb{R}$, $i = 1, \ldots, N$, where y_i and y_j are conditionally independent given $\boldsymbol{x}_i, \boldsymbol{x}_j$. Our objective is to find optimal parameters $\boldsymbol{\theta}^* \in \mathbb{R}^D$ for the linear regression model (1.2).[21] Once the parameters are found, we can predict function values by using the estimate $\boldsymbol{\theta}^*$ in the model (1.2).

[21]The corresponding graphical model is given in Fig. 1.22.

1.5.1 Maximum Likelihood Estimation

A widely used approach to finding the desired parameters $\boldsymbol{\theta}^*$ is maximum likelihood estimation where

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})\,, \qquad \boldsymbol{X} := [\boldsymbol{x}_1|\cdots|\boldsymbol{x}_N]^\top \in \mathbb{R}^{N\times D}\,, \quad \boldsymbol{y} := [y_1,\dots,y_N]^\top \in \mathbb{R}^N \tag{1.210}$$

maximizes the likelihood function $p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})$.

Remark 14
Note that the likelihood is not a probability distribution in $\boldsymbol{\theta}$: It is simply a function but does usually not integrate to 1 (i.e., it is unnormalized), and may not even be integrable with respect to $\boldsymbol{\theta}$. However, the likelihood in (1.210) *is a probability distribution in $\boldsymbol{y}$.*

To find the desired parameters $\boldsymbol{\theta}^*$ that maximize the likelihood, we typically do gradient ascent (or gradient descent on the negative likelihood). For numerical reasons, we apply the log-transformation to the problem[22] and minimize the negative log-likelihood

$$\begin{aligned}
\theta^* &\in \arg\min_{\boldsymbol{\theta}} \big(-\log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})\big) && (1.211)\\
&= \arg\min_{\boldsymbol{\theta}} \left(-\log \prod_{i=1}^N p(y_i|\boldsymbol{x}_i,\boldsymbol{\theta})\right) && (1.212)\\
&= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N -\log p(y_i|\boldsymbol{x}_i,\boldsymbol{\theta})\,, && (1.213)
\end{aligned}$$

where we exploited that the likelihood factorizes over the number of data points due to our independence assumption on the training set.
In the linear regression model (1.2) the likelihood is Gaussian (due to the Gaussian additive noise term), such that we arrive at

$$-\log p(y_i|\boldsymbol{x}_i,\boldsymbol{\theta}) = \frac{1}{2\sigma^2}(y_i - \boldsymbol{x}_i^\top\boldsymbol{\theta})^2 + \text{const} \tag{1.214}$$

where the constant includes all terms independent of $\boldsymbol{\theta}$. Using (1.214) in the negative log-likelihood (1.213) we obtain (ignoring the constant terms)

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) := -\log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) &= \frac{1}{2\sigma^2}\sum_{i=1}^N (y_i - \boldsymbol{x}_i^\top\boldsymbol{\theta})^2 && (1.215)\\
&= \frac{1}{2\sigma^2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta})^\top(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}) = \frac{1}{2\sigma^2}\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}\|^2\,, && (1.216)\\
\boldsymbol{X} &:= \begin{bmatrix}\boldsymbol{x}_1^\top\\ \vdots \\ \boldsymbol{x}_N^\top\end{bmatrix} \in \mathbb{R}^{N\times D}\,, && (1.217)
\end{aligned}$$

[22]Note that the logarithm is a (strictly) monotonically increasing function.

where $\boldsymbol{X}$ is called the **design matrix**.[23] In (1.216) we replaced the sum of squared with the squared norm[24] of the difference term $\boldsymbol{y} - \boldsymbol{X\theta}$.

Remark 15
In machine learning, the negative log likelihood function is also called an **error function**.

Now that we have a concrete form of the negative log-likelihood function we need to optimize. We will discuss two approaches, both of which are based on gradients.

1.5.1.1 Closed-Form Solution

We immediately see that (1.216) is quadratic in $\boldsymbol{\theta}$. This means that we can find a unique global solution $\boldsymbol{\theta}^*$ for minimizing the negative log-likelihood. We can find the global optimum by computing the gradient of $\mathcal{L}$, setting it to $\mathbf{0}$ and solving for $\boldsymbol{\theta}$.[25]
We compute the gradient of $\mathcal{L}$ with respect to the parameters as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} &= \frac{\partial}{\partial \boldsymbol{\theta}}\left(\frac{1}{2\sigma^2}(\boldsymbol{y}-\boldsymbol{X\theta})^\top(\boldsymbol{y}-\boldsymbol{X\theta})\right) && (1.218)\\ &= \frac{1}{2\sigma^2}\frac{\partial}{\partial \boldsymbol{\theta}}\left(\boldsymbol{y}^\top\boldsymbol{y} - 2\boldsymbol{y}^\top\boldsymbol{X\theta} + \boldsymbol{\theta}^\top\boldsymbol{X}^\top\boldsymbol{X\theta}\right) && (1.219)\\ &= \frac{1}{\sigma^2}(-\boldsymbol{y}^\top\boldsymbol{X} + \boldsymbol{\theta}^\top\boldsymbol{X}^\top\boldsymbol{X}) \in \mathbb{R}^{1\times D}\,. && (1.220)\end{aligned}$$

A necessary condition for $\boldsymbol{\theta}$ being optimal we set this gradient to $\mathbf{0}$ and obtain

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \mathbf{0} &\overset{(1.220)}{\Leftrightarrow} (\boldsymbol{\theta}^*)^\top\boldsymbol{X}^\top\boldsymbol{X} = \boldsymbol{y}^\top\boldsymbol{X} && (1.221)\\ &\Leftrightarrow (\boldsymbol{\theta}^*)^\top = \boldsymbol{y}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1} && (1.222)\\ &\Leftrightarrow \boldsymbol{\theta}^* = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} && (1.223)\end{aligned}$$

We could right-multiply the first equation by $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$ because $\boldsymbol{X}^\top\boldsymbol{X}$ is positive definite (if we do not have two identical inputs $\boldsymbol{x}_i, \boldsymbol{x}_j$ for $i \neq j$). Note that we actually do obtain a global minimum since the Hessian $\nabla^2_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \boldsymbol{X}^\top\boldsymbol{X} \in \mathbb{R}^{D\times D}$ is positive definite.

1.5.1.2 Iterative Solution

We may be interested in finding parameters on which the log-likelihood depends in a more complicated way. Then, a closed-form solution is often not available. However, we can use an iterative procedure to find a local optimum. The most straightforward procedure for this is gradient descent, which we will discuss in more detail in Section 1.6.

[23]Note that there is some notation overloading: We summarize the set of training inputs in $\boldsymbol{X}$, whereas in the design matrix we additionally assume a specific "shape".
[24]Remember that $\|\boldsymbol{x}\|^2 := \langle \boldsymbol{x}, \boldsymbol{x}\rangle$ where $\langle\cdot,\cdot\rangle$ is a scalar product (inner product).
[25]In this case, setting the gradient to $\mathbf{0}$ is a necessary and sufficient condition. As an exercise, compute the Hessian (matrix of second derivatives) and show that it is positive definite.

1.5.1.3 Maximum Likelihood Estimation with Features

When we consider the linear regression model

$$y = \boldsymbol{\phi}^\top(\boldsymbol{x})\boldsymbol{\theta} + \epsilon \tag{1.224}$$

where $\boldsymbol{\phi} : \mathbb{R}^D \to \mathbb{R}^K$ is a (nonlinear) transformation of the inputs $\boldsymbol{x}$.

Example (Polynomial Regression)
We are concerned with a regression problems $y = \boldsymbol{\phi}^\top(x)\boldsymbol{\theta} + \epsilon$, where $x \in \mathbb{R}$. A transformation that is often used in this context is

$$\boldsymbol{\phi}(x) = \begin{bmatrix} \phi_0(x) \\ \phi_1(x) \\ \vdots \\ \phi_K(x) \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \\ \vdots \\ x^K \end{bmatrix} \in \mathbb{R}^{K+1}. \tag{1.225}$$

This means, we "lift" the original 1D input space into a $K+1$-dimensional feature space consisting of monomials. With these features, we can model polynomials of degree $\leq K$ within the framework of linear regression: A polynomial of degree K is given by

$$f(x) = \sum_{i=0}^{K} \theta_i x^i = \boldsymbol{\phi}^\top(x)\boldsymbol{\theta} \tag{1.226}$$

where $\boldsymbol{\phi}$ is defined in (1.225) and $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_K]^\top$ contains the parameters θ_i.

When we consider the training data $\boldsymbol{x}_i, y_i$, $i = 1, \ldots, N$ and define the feature (design) matrix

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_0^\top(\boldsymbol{x}_1) & \phi_1^\top(\boldsymbol{x}_1) & \cdots & \phi_K^\top(\boldsymbol{x}_1) \\ \phi_0^\top(\boldsymbol{x}_2) & \phi_1^\top(\boldsymbol{x}_2) & \cdots & \phi_K^\top(\boldsymbol{x}_2) \\ \vdots & \vdots & & \vdots \\ \phi_0^\top(\boldsymbol{x}_N) & \cdots & \cdots & \phi_K^\top(\boldsymbol{x}_N) \end{bmatrix}, \quad \Phi_{ij} = \phi_j^\top(\boldsymbol{x}_i) \tag{1.227}$$

the negative log-likelihood can be written as

$$-\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{\Phi\theta})^\top(\boldsymbol{y} - \boldsymbol{\Phi\theta}) + \text{const}. \tag{1.228}$$

Comparing (1.228) with (1.216) we immediately see that $\boldsymbol{X}$ is replaced by $\boldsymbol{\Phi}$. Since both $\boldsymbol{X}$ and $\boldsymbol{\Phi}$ are independent of the parameters $\boldsymbol{\theta}$ that we wish to optimize, we therefore also arrive immediately at the maximum likelihood estimate

$$\boldsymbol{\theta}^* = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y} \tag{1.229}$$

for the linear regression problem with nonlinear features defined in (1.224).

Example (Feature Matrix for Polynomials)
For a second-order polynomial and N training points $x_i \in \mathbb{R}, i = 1, \dots, N$, the feature matrix is

$$\boldsymbol{\Phi} = \begin{bmatrix} 0 & x_1 & x_1^2 \\ 0 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 0 & x_N & x_N^2 \end{bmatrix} . \tag{1.230}$$

1.5.1.4 Properties

The maximum likelihood estimate $\boldsymbol{\theta}^*$ possesses the following properties:

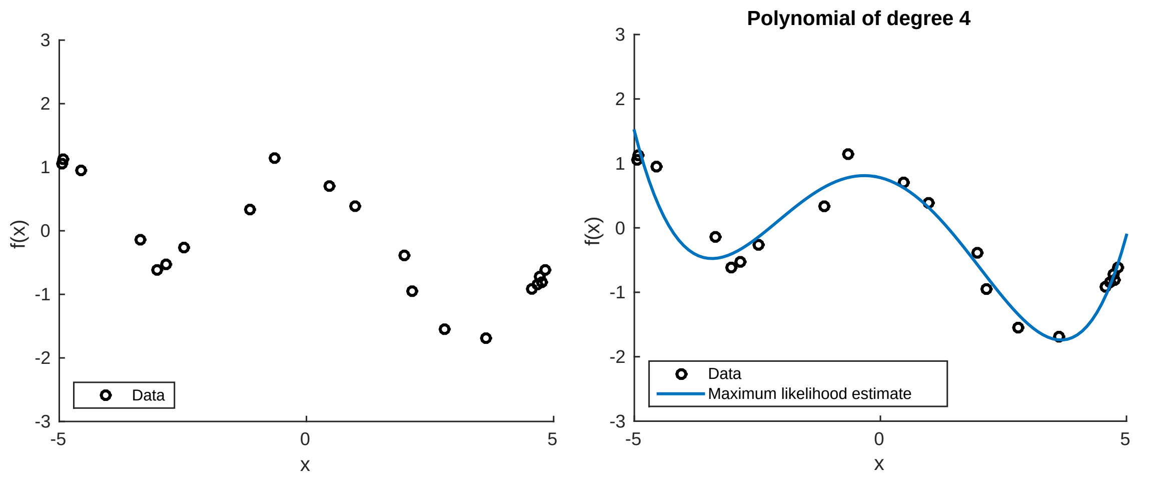
- Asymptotic consistency: The MLE converges to the true value in the limit of infinitely many observations, plus a random error that is approximately normal.
- The size of the samples necessary to achieve these properties can be quite large.
- The error's variance decays in $1/N$ where N is the number of data points.
- Especially, in the "small" data regime, maximum likelihood estimation can lead to **overfitting**.

Example (Maximum Likelihood Polynomial Fit)

Let us consider the data set in Fig. 1.23(a). The data set consists of $N = 20$ pairs (x_i, y_i), where $x_i \sim \mathcal{U}[-5, 5]$ and $y_i = -\sin(x_i/5) + \cos(x_i) + \epsilon$, where $\epsilon \sim \mathcal{N}\big(0,\, 0.2^2\big)$. We fit a polynomial of degree $M = 4$ using maximum likelihood estimation (i.e., the parameters are given in (1.229)). The maximum likelihood estimate yields an expected function value $\phi(x_*)^\top \boldsymbol{\theta}_{\text{ML}}$ at a new test location x_*. The result is shown in Fig. 1.23(b).

1.5.2 Overfitting

We have seen that we can use maximum likelihood estimation to fit linear models (e.g., polynomials) to data. We can evaluate the quality of the model by computing the error/loss incurred. One way of doing this is to compute the negative log-likelihood (1.213), which we minimized to determine the MLE. Alternatively, given that the noise parameter σ^2 is not a free parameter, we can ignore the scaling by



(a) Regression data set.

(b) Polynomial of degree 4 determined by maximum likelihood estimation.

Figure 1.23: Polynomial regression. (a) Data set consisting of (x_i, y_i) pairs, $i = 1, \ldots, 20$. (b) Maximum likelihood polynomial of degree 4.

$1/\sigma^2$, so that we end up with a squared-error-loss function $\|\boldsymbol{y} - \boldsymbol{\Phi\theta}\|^2$. Instead of using this squared loss, we often use the **root mean squared error (RMSE)**

$$\sqrt{\|\boldsymbol{y} - \boldsymbol{\Phi\theta}\|^2/N} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - \boldsymbol{\phi}^\top(\boldsymbol{x}_n)\boldsymbol{\theta})^2}\,, \tag{1.231}$$

which (a) allows us to compare errors of data sets with different sizes[26] and (b) has the same scale and the same units as the observed function values y_i.[27] Note that the division by σ^2 makes the log-likelihood “unit-free”.

We can use the RMSE (or the log-likelihood) to determine the best degree of the polynomial by finding the value M, such that the error is minimized. Given that the polynomial degree is a natural number, we can perform a brute-force search and enumerate all (reasonable) values of M.[28]

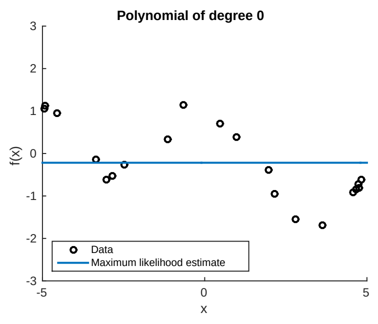
Fig. 1.24 shows a number of polynomial fits determined by maximum likelihood. We notice that polynomials of low degree (e.g., constants ($M = 0$) or linear ($M = 1$) fit the data poorly and, hence, are poor representations of the true underlying function. For degrees $M = 4, \ldots, 9$ the fits look plausible and smoothly interpolate the data. When we go to higher-degree polynomials, we notice that they fit the data better and better—in the extreme case of $M = N - 1 = 19$, the function passes through every single data point. However, these high-degree polynomials oscillate wildly and are a poor representation of the underlying function that generated the data.[29] The property of the polynomials fitting the noise structure is called **overfitting**.

[26]The RMSE is normalized.

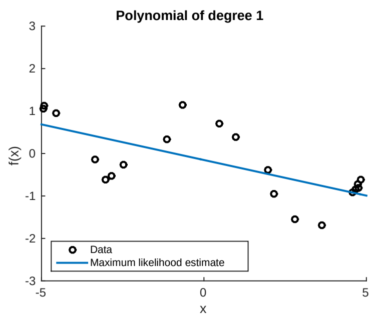
[27]Assume, we fit a model that maps post-codes ($\boldsymbol{x}$ is given in latitude,longitude) to house prices (y-values are GBP). Then, the RMSE is also measured in GBP, whereas the squared error is given in GBP^2.

[28]For a training set of size N it is sufficient to test $0 \leq M \leq N - 1$.

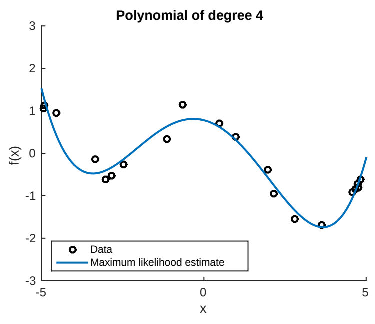
[29]Note that the noise variance $\sigma^2 > 0$.



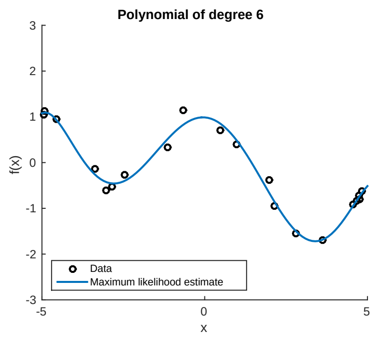
(a) $M = 0$



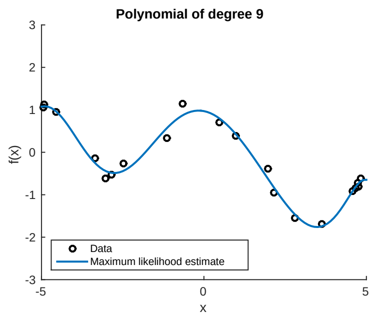
(b) $M = 1$



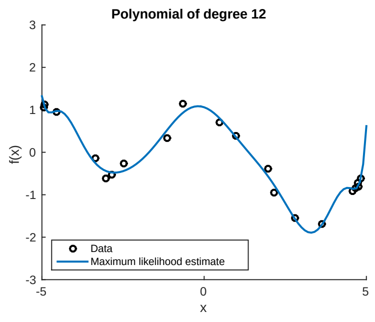
(c) $M = 4$



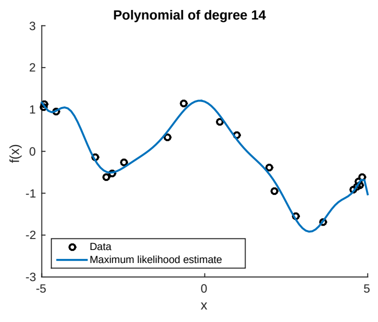
(d) $M = 6$



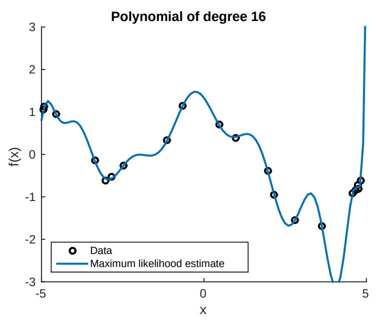
(e) $M = 9$



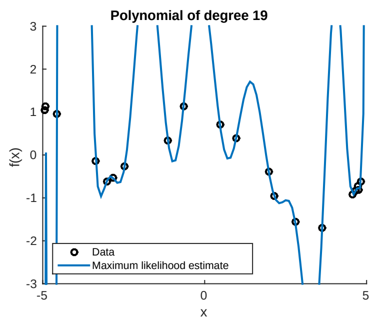
(f) $M = 12$



(g) $M = 14$



(h) $M = 16$



(i) $M = 19$

Figure 1.24: Polynomial fits for different degrees M

Remember that the goal is to achieve good generalization by making accurate predictions for new (unseen) data. We obtain some quantitative insight into the dependence of the generalization performance on the polynomial of degree M by considering a separate test set comprising 100 data points generated using exactly the same procedure used to generate the training set, but with new choices for the random noise values included in the target values. As test inputs, we chose a linear grid of 100 points in the interval of $[-5, 5]$. For each choice of M, we evaluate the RMSE (1.231) for both the training data and the test data.

Looking now at the test error, which is a qualitive measure of the generalization properties of the corresponding polynomial, we notice that initially the test error decreases, see Fig. 1.25 (red). For fourth-order polynomials the test error is relatively low and stays relatively constant up to degree 11. However, from degree 12 onward the test error increases significantly, and high-order polynomials have very

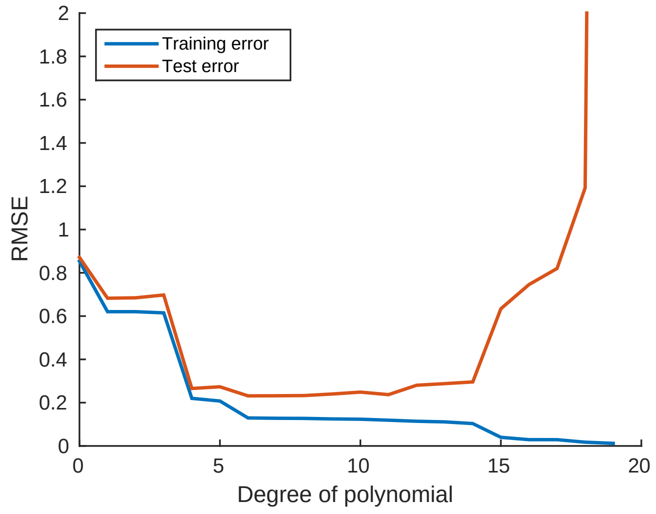


Figure 1.25: Training and test error.

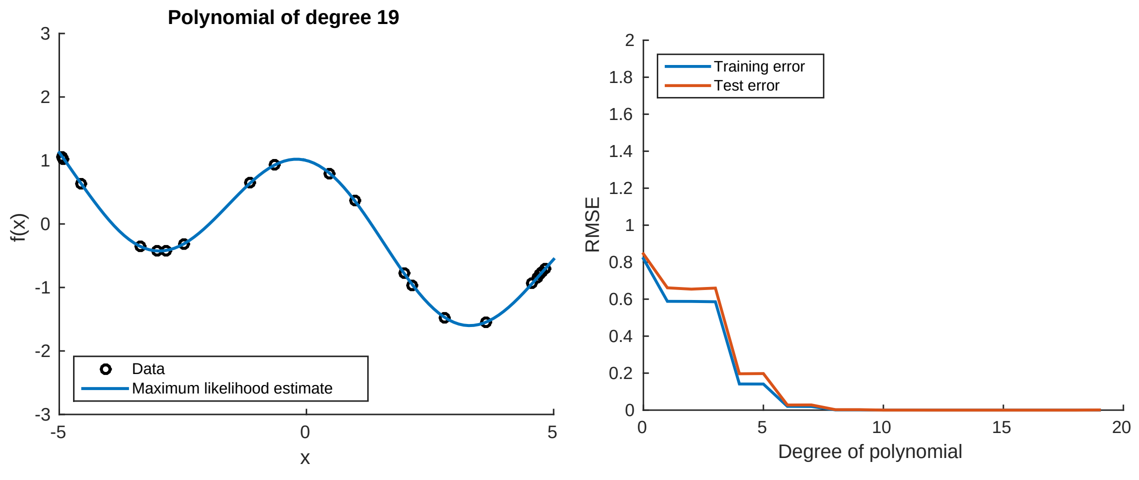
bad generalization properties. In this particular example, this also is evident from the corresponding maximum likelihood fits in Fig. 1.24. Note that the training error (blue curve in Fig. 1.25) never increases as a function of M. In our example, the best generalization (the point of the smallest test error) is achieved for a polynomial of degree $M = 6$.

It is a bit counter-intuitive that a polynomial of degree $M = 19$ is a worse approximation than a polynomial of degree $M = 4$, which is a special case of a 19th-order polynomial (by setting all higher coefficients to 0). However, a 19th-order polynomial can also describe many more functions, i.e., it is a much more flexible model. In the data set we considered, the observations y_n were noisy (i.i.d. Gaussian). A polynomial of a high degree will use its flexibility to model random disturbances as systematic/structural properties of the underlying function. Overfitting can be seen as a general problem of maximum likelihood estimation (Bishop, 2006). Assuming we had noise-free data, overfitting does not occur, which is also revealed by the test error, see Fig. 1.26.

1.5.3 Regularization

In Section 1.5.1, we saw that maximum likelihood estimation is prone to overfitting. It often happens that the parameter values become relatively big if we run into overfitting (Bishop, 2006). One way to control overfitting is to penalize big parameter values. A technique that is often used to control overfitting is **regularization**. In regularization, we add a term to the log-likelihood that penalizes the amplitude of the parameters $\boldsymbol{\theta}$. A typical example is a "loss function" of the form

$$-\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2\,, \tag{1.232}$$



(a) Maximum likelihood fit. (b) Training and test error

Figure 1.26: Noise-free maximum likelihood estimation with a 19th-degree polynomial. (a) Overfitting no longer occurs; (b) the test error declines to 0.

where the second term is the regularizer, and $\lambda \geq 0$ controls the "strictness" of the regularization.[30]

1.5.4 Maximum-A-Posterior (MAP) Estimation

From a probabilistic perspective, adding a regularizer is identical to using a prior distribution $p(\boldsymbol{\theta})$ on the parameters and then selecting the parameters that maximize the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y})$, i.e., we choose the parameters $\boldsymbol{\theta}$ that are "most probable" given the training data: In a Bayesian setting, the posterior over the parameters $\boldsymbol{\theta}$ given the training data $\boldsymbol{X}, \boldsymbol{y}$ is obtained by applying Bayes' theorem as

$$p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y}|\boldsymbol{X})}\,. \tag{1.233}$$

The parameter vector $\boldsymbol{\theta}_{\text{MAP}}$ that maximizes the posterior (1.233) is called the **maximum a-posteriori (MAP)** estimate.
To find the MAP estimate, we follow steps that are similar in flavor to maximum likelihood estimation. We start with the log-transform and compute the log-posterior as

$$\log p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}) = \log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) + \text{const}\,, \tag{1.234}$$

where the constant includes the terms independent of $\boldsymbol{\theta}$. We see that the log-posterior in (1.234) consists of the log-likelihood $p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})$ and the log-prior $\log p(\boldsymbol{\theta})$.

[30]Instead of the 2-norm, we can choose and p-norm $\|\cdot\|_p$. In practice, smaller values for p lead to sparser solutions. Here, "sparse" means that many parameter values $\theta_i = 0$, which is also useful for variable selection. For $p = 1$, the regularizer is called LASSO (least absolute shrinkage and selection operator) and was proposed by Tibshirani (1996).

Remark 16 (Relation to Regularization)
Choosing a Gaussian parameter prior $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0},\, b^2\boldsymbol{I})$, $b^2 = \frac{1}{2\lambda}$, *the (negative) log-prior term will be*

$$-\log p(\boldsymbol{\theta}) = \underbrace{\lambda\boldsymbol{\theta}^\top\boldsymbol{\theta}}_{=\lambda\|\boldsymbol{\theta}\|_2^2} + const\,, \tag{1.235}$$

and we recover exactly the regularization term in (1.232). *This means that for a quadratic regularization, the regularization parameter* λ *in* (1.232) *corresponds to twice the precision (inverse variance) of the Gaussian (isotropic) prior* $p(\boldsymbol{\theta})$. *The log-prior in* (1.234) *plays the role of a regularizer that penalizes implausible values, i.e., values that are unlikely under the prior.*

To find the MAP estimate $\boldsymbol{\theta}_{\text{MAP}}$, we minimize the negative log-posterior with respect to $\boldsymbol{\theta}$, i.e., we solve

$$\boldsymbol{\theta}_{\text{MAP}} \in \arg\min_{\boldsymbol{\theta}} -\log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})\,. \tag{1.236}$$

We compute the gradient with respect to $\boldsymbol{\theta}$ as

$$-\frac{\partial \log p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y})}{\partial\boldsymbol{\theta}} = -\frac{\partial \log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} - \frac{\partial \log p(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}\,, \tag{1.237}$$

where we identify the first term on the right-hand-side as the gradient of the negative log-likelihood given in (1.220).

1.5.4.1 MAP Estimation for Linear Regression

We consider the linear regression problem where

$$y = \boldsymbol{\phi}^\top(\boldsymbol{x})\boldsymbol{\theta} + \epsilon\,, \quad \epsilon \sim \mathcal{N}(0,\, \sigma^2)\,, \tag{1.238}$$

with a Gaussian prior $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0},\, b^2\boldsymbol{I})$ on the parameters $\boldsymbol{\theta}$.
The negative log-posterior for this model is

$$-\log p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y}) = \frac{1}{2\sigma^2}(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{\theta})^\top(\boldsymbol{y}-\boldsymbol{\Phi}\boldsymbol{\theta}) + \frac{1}{2b^2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \text{const}\,. \tag{1.239}$$

Here, the blue term corresponds to the contribution from the log-likelihood, and the red term originates from the log-prior.
The gradient of the log-posterior with respect to the parameters $\boldsymbol{\theta}$ is

$$-\frac{\partial \log p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y})}{\partial\boldsymbol{\theta}} = \frac{1}{\sigma^2}(\boldsymbol{\theta}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \boldsymbol{y}^\top\boldsymbol{\Phi}) + \frac{1}{b^2}\boldsymbol{\theta}^\top\,. \tag{1.240}$$

We will find the MAP estimate $\boldsymbol{\theta}_{\text{MAP}}$ by setting this gradient to $\mathbf{0}$:

$$\frac{1}{\sigma^2}(\boldsymbol{\theta}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \boldsymbol{y}^\top\boldsymbol{\Phi}) + \frac{1}{b^2}\boldsymbol{\theta}^\top = \mathbf{0} \tag{1.241}$$

$$\Leftrightarrow \boldsymbol{\theta}^\top\left(\frac{1}{\sigma^2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \frac{1}{b^2}\boldsymbol{I}\right) - \frac{1}{\sigma^2}\boldsymbol{y}^\top\boldsymbol{\Phi} = \mathbf{0} \tag{1.242}$$

$$\Leftrightarrow \boldsymbol{\theta}^\top \left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \frac{\sigma^2}{b^2} \boldsymbol{I} \right) = \boldsymbol{y}^\top \boldsymbol{\Phi} \tag{1.243}$$

$$\Leftrightarrow \boldsymbol{\theta}^\top = \boldsymbol{y}^\top \boldsymbol{\Phi} \left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \frac{\sigma^2}{b^2} \boldsymbol{I} \right)^{-1} \tag{1.244}$$

$$\Leftrightarrow \boldsymbol{\theta}_{\mathrm{MAP}} = \left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \frac{\sigma^2}{b^2} \boldsymbol{I} \right)^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y}\,. \tag{1.245}$$

Comparing the MAP estimate in (1.245) with the maximum likelihood estimate in (1.229) we see that the only difference between both solutions is the additional red term $\frac{\sigma^2}{b^2}\boldsymbol{I}$ in the inverse matrix.[31] This term ensures that the inverse exists and serves as a **regularizer**.

Example (MAP Estimation for Polynomial Regression)

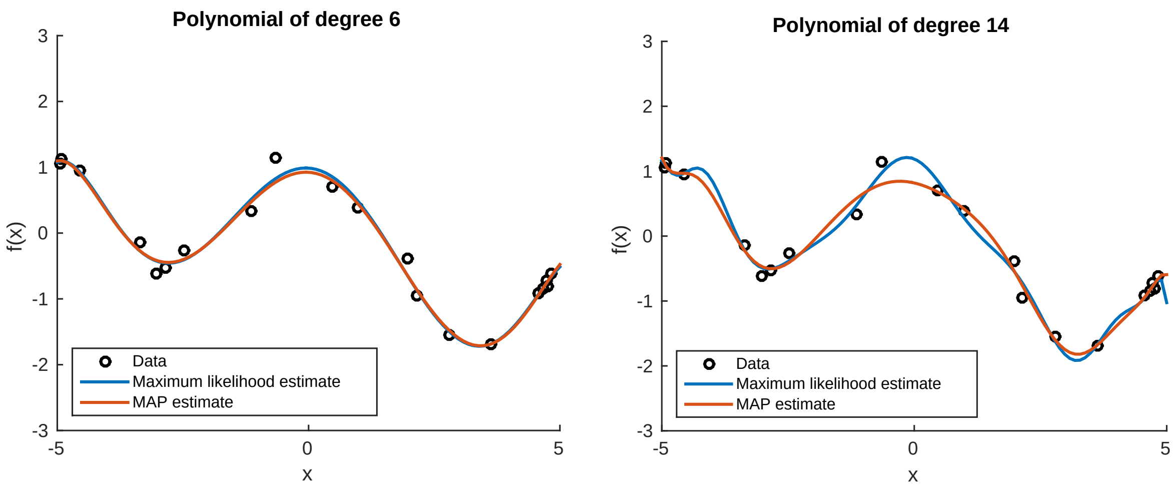


Figure 1.27: Polynomial regression: Maximum likelihood and MAP estimates.

In the polynomial regression example from Section 1.5.1, we place a Gaussian prior $p(\boldsymbol{\theta}) = \mathcal{N}(0,\, 0.1^2\boldsymbol{I})$ on the parameters $\boldsymbol{\theta}$ and determine the MAP estimates according to (1.245). In Fig. 1.27, we show both the maximum likelihood and the MAP estimates for polynomials of degree 6 (left) and degree 14 (right). The prior (regularizer) does not play a significant role for the low-degree polynomial, but keeps the function relatively smooth for higher-degree polynomials. However, the MAP estimate is only able to push the boundaries of overfitting—it is not a general solution to this problem.

[31] $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ is positive semidefinite and the additional term is strictly positive definite, such that all eigenvalues of the matrix to be inverted are positive.

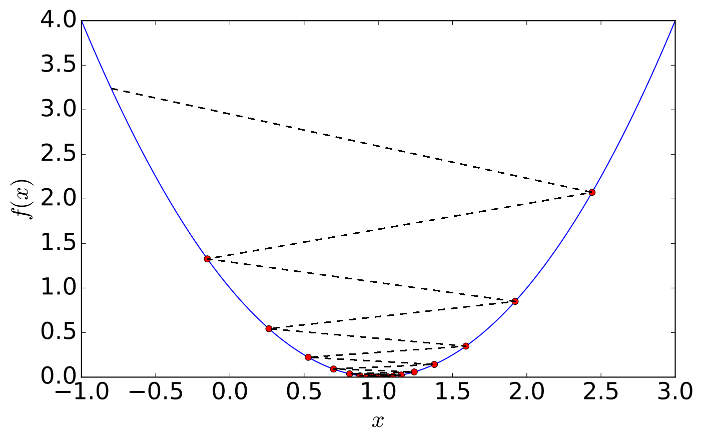


Figure 1.28: Gradient descent can lead to zigzagging and slow convergence.

1.6 Gradient Descent

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. Remember that the gradient points in the direction of the steepest ascent and it is orthogonal to the contour lines of the function we wish to optimize.
Gradient descent exploits the fact that if a multivariate function $\boldsymbol{f}(\boldsymbol{x})$ is differentiable in a neighborhood of a point $\boldsymbol{x}_0$ then $\boldsymbol{f}(\boldsymbol{x}_0)$ decreases fastest if one moves from $\boldsymbol{x}_0$ in the direction of the negative gradient $-((\nabla \boldsymbol{f})(\boldsymbol{x}_0))^\top$ of $\boldsymbol{f}$ at $\boldsymbol{x}_0$. Then, if

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \gamma((\nabla \boldsymbol{f})^\top(\boldsymbol{x}_0))^\top \tag{1.246}$$

for a small **step size** $\gamma \geq 0$ then $\boldsymbol{f}(\boldsymbol{x}_1) \leq \boldsymbol{f}(\boldsymbol{x}_0)$. Note that we use the transpose for the gradient since otherwise the dimensions will not work out.
This observation allows us to define a simple gradient-descent algorithm: If we want to find a local optimum $\boldsymbol{f}(\boldsymbol{x}_*)$ of a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m,\ \boldsymbol{x} \mapsto \boldsymbol{f}(\boldsymbol{x})$, we start with an initial guess $\boldsymbol{x}_0$ of the parameters we wish to optimize and then iterate according to

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i((\nabla \boldsymbol{f})(\boldsymbol{x}_i))^\top\,. \tag{1.247}$$

For suitable γ_i, the sequence $\boldsymbol{f}(\boldsymbol{x}_0) \geq \boldsymbol{f}(\boldsymbol{x}_1) \geq \ldots$ converges to a local minimum.

Remark 17
Gradient descent can be relatively slow close to the minimum: Its asymptotic rate of convergence is inferior to many other methods. For poorly conditioned convex problems, gradient descent increasingly 'zigzags' as the gradients point nearly orthogonally to the shortest direction to a minimum point, see Fig. 1.28.

1.6.1 Stepsize

Choosing a good stepsize is important in gradient descent: If the stepsize (also called the learning rate) is too small, gradient descent can be slow. If the stepsize is chosen too large, gradient descent can overshoot, fail to converge, or even diverge.

Robust gradient methods permanently rescale the stepsize empirically depending on local properties of the function. There are two simple heuristics (Toussaint, 2012):

- When the function value increases after a gradient step, the step size was too large. Undo the step and decrease the stepsize.
- When the function value decreases the step could have been larger. Try to increase the stepsize.

Although the "undo" step seems to be a waste of resources, using this heuristic guarantees monotonic convergence.

Example (Solving a Linear Equation System)
When we solve linear equations of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, in practice we solve $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}$ approximately by finding $\boldsymbol{x}_*$ that minimizes the the squared error

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \tag{1.248}$$

if we use the Euclidean norm. The gradient of (1.248) with respect to $\boldsymbol{x}$ is

$$\nabla_{\boldsymbol{x}} = 2(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top \boldsymbol{A}\,. \tag{1.249}$$

Remark 18
Gradient descent is rarely used for solving linear equation systems $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Instead other algorithms with better convergence properties, e.g., conjugate gradient descent, are used. The speed of convergence of gradient descent depends on the maximal and minimal eigenvalues of $\boldsymbol{A}$, while the speed of convergence of conjugate gradients has a more complex dependence on the eigenvalues, and can benefit from preconditioning. Gradient descent also benefits from preconditioning, but this is not done as commonly. For further information on gradient descent, pre-conditioning and convergence we refer to CO-477.

1.6.2 Gradient Descent with Momentum

Gradient descent with momentum (Rumelhart et al., 1986) is a method that smoothes out erratic behavior of gradient updates and dampens oscillations.[32]
The idea is to have a gradient update with a memory to implement a moving average. The momentum-based method remembers the update $\Delta\boldsymbol{x}_i$ at each iteration i and determines the next update as a linear combination of the current and previous gradients

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i((\nabla \boldsymbol{f})(\boldsymbol{x}_i))^\top + \alpha\Delta\boldsymbol{x}_i \tag{1.250}$$

$$\Delta\boldsymbol{x}_i = \boldsymbol{x}_i - \boldsymbol{x}_{i-1} = -\gamma_{i-1}((\nabla \boldsymbol{f})(\boldsymbol{x}_{i-1}))^\top\,, \tag{1.251}$$

where $\alpha \in [0,1]$. Due to the moving average of the gradient, momentum-based methods are particularly useful when the gradient itself is only a (noisy) estimate. We will discuss stochastic approximations to the gradient in the following.

[32]Goh (2017) wrote an intuitive blog post on gradient descent with momentum.

1.6.3 Stochastic Gradient Descent

Computing the gradient can be very time consuming. However, often it is possible to find a "cheap" approximation of the gradient. Approximating the gradient is still useful as long as it points in roughly the same direction as the true gradient.
Stochastic gradient descent (often shortened in SGD) is a stochastic approximation of the gradient descent method for minimizing an objective function that is written as a sum of differentiable functions.
In machine learning, we often consider objective functions of the form

$$L(\boldsymbol{\theta}) = \sum_k L_k(\boldsymbol{\theta}) \tag{1.252}$$

where $\boldsymbol{\theta}$ is the parameter vector of interest, i.e., we want to find $\boldsymbol{\theta}$ that minimizes L. An example is the negative log-likelihood

$$L(\boldsymbol{\theta}) = -\sum_k \log p(y_k|\boldsymbol{x}_k, \boldsymbol{\theta}) \tag{1.253}$$

in a regression setting, where $\boldsymbol{x}_k \in \mathbb{R}^D$ are the training inputs, y_k are the training targets and $\boldsymbol{\theta}$ are the parameters of the regression model.
Standard gradient descent, as introduced previously, is a "batch" optimization method, i.e., optimization is performed using the full training set by updating the parameters according to

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma_i(\nabla L(\boldsymbol{\theta}_i))^\top = \boldsymbol{\theta}_i - \gamma_i \sum_k (\nabla L_k(\boldsymbol{\theta}_i))^\top \tag{1.254}$$

for a suitable stepsize parameter γ_i. Evaluating the sum-gradient may require expensive evaluations of the gradients from all summand functions. When the training set is enormous (commonly the case in Deep Learning) and/or no simple formulas exist, evaluating the sums of gradients becomes very expensive: Evaluating the gradient requires evaluating the gradients of all summands. To economize on the computational cost at every iteration, **stochastic gradient descent** samples a subset of summand functions ("mini-batch") at every step. In an extreme case, the sum in (1.252) is approximated by a single summand (randomly chosen), and the parameters are updated according to

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma(\nabla L_k(\boldsymbol{\theta}_i))^\top . \tag{1.255}$$

In practice, it is good to keep the size of the mini-batch as large as possible to (a) reduce the variance in the parameter update[33] and (b) allow for the computation to take advantage of highly optimized matrix operations that should be used in a well-vectorized computation of the cost and gradient. When we choose the mini-batch size, we need to make sure it fits into CPU/GPU memory. Typical mini-batch sizes are 64, 128, 256, 512, 1024, which depends on the way computer memory is laid out and accessed.

[33]This often leads to more stable convergence since the gradient estimator is less noisy.

Remark 19
When the learning rate decreases at an appropriate rate, and subject to relatively mild assumptions, stochastic gradient descent converges almost surely to local minimum (Bottou, 1998).

If we keep the mini-batch size small, the noise in our gradient estimate will allow us to get out of some bad local optima, which we may otherwise get stuck in.

1.6.4 Further Reading

For non-differentiable functions, gradient methods are ill-defined. In these cases, **subgradient methods** can be used (Shor, 1985). For further information and algorithms for optimizing non-differentiable functions, we refer to the book by Bertsekas (1999).
Stochastic gradient descent is very effective in large-scale machine learning problems, such as training deep neural networks on millions of images (Dean et al., 2012), topic models (Hoffman et al., 2013), reinforcement learning (Mnih et al., 2015) or training large-scale Gaussian process models (Hensman et al., 2013; Gal et al., 2014). Extensions of stochastic gradient descent methods include RMSProp, AdaGrad (Duchi et al., 2011), momentum-based methods, and Adam (Kingma and Ba, 2014).

1.7 Model Selection and Cross Validation

Sometimes, we need to make high-level decisions about the model we want to use in order to increase the performance. Examples include:

- The degree of a polynomial in a regression setting
- The number of components in a mixture model
- The network architecture of a (deep) neural network
- The type of kernel in a support vector machine

The choices we make (e.g., the degree of the polynomial) influence the number of free parameters in the model and thereby also the model complexity. More complex models are more flexible in the sense that they can be used to describe more data sets. For instance, a polynomial of degree 1 (a line $a_0 + a_1 x$) can only be used to describe linear relations between inputs x and observations y. A polynomial of degree 2 can additionally describe quadratic relationships between inputs and observations.[34] Higher-order polynomials are very flexible models as we have seen already in Section 1.5 in the context of polynomial regression.
A general problem is that at training time we can only use the training set to evaluate the performance of the model. However, the performance on the training set is

[34]A polynomial $a_0 + a_1 x + a_2 x^2$ can also describe linear functions by setting $a_2 = 0$.

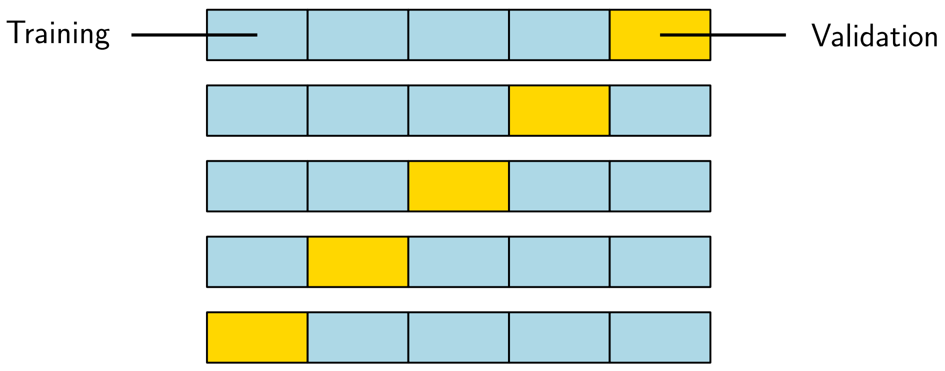


Figure 1.29: K-fold cross-validation. The data set is divided into $K = 5$ chunks, $K - 1$ of which serve as the training set (blue) and one as the validation set (yellow). This procedure is repeated for all K choices for the validation set, and the performance of the model from the K runs is averaged.

not really what we are interested in: In Section 1.5, we have seen that maximum likelihood estimation can lead to overfitting, especially when the training data set is small. Ideally, our model (also) works well on the test set (which is not available at training time). Therefore, we need some mechanisms for assessing how a model **generalizes** to unseen test data. **Model selection** is concerned with exactly this problem.

1.7.1 Cross-Validation to Assess the Generalization Performance

One way to assess the generalization performance of a model M is to use a **validation set**. The validation set a small subset of the available training set that we keep aside. A practical issue with this approach is that the amount of data is limited, and ideally we would use as much of the data available to train the model. This would require to keep our validation set $\mathcal{V}$ small, which then would lead to a noisy estimate (with high variance) of the predictive performance. One solution to these contradictive objectives (large training set, large validation set) is to use **cross-validation**. K-fold cross-validation effectively partitions the data into K chunks, $K - 1$ of which form the training set $\tilde{\mathcal{D}}$, and the last chunk serves as the validation set $\mathcal{V}$ (similar to the idea outlined above). Cross-validation iterates through (ideally) all combinations of assignments of chunks to $\tilde{\mathcal{D}}$ and $\mathcal{V}$, see Fig. 1.29.
We partition our training set $\mathcal{D} = \tilde{\mathcal{D}} \cup \mathcal{V}$, $\tilde{\mathcal{D}} \cap \mathcal{V} = \emptyset$, where $\mathcal{V}$ is the validation set, and train our model on $\tilde{\mathcal{D}}$. After training, we assess the performance of the model M on the validation set $\mathcal{V}$ (e.g., by computing RMSE values of the trained model on the validation set). We cycle through all possible partitionings of validation and training sets and compute the average generalization error of the model. Cross-validation effectively computes the expected generalization error

$$\mathbb{E}_{\mathcal{V}}[G(\mathcal{V})|M] \approx \frac{1}{K}\sum_{k=1}^{K} G(\mathcal{V}^{(k)}|M)\,, \tag{1.256}$$

where $G(\mathcal{V})$ is the generalization error (e.g., RMSE) on the validation set $\mathcal{V}$ for model

M. We repeat this procedure for all models and choose the model that performs best. Note that cross-validation does not only give us the expected generalization error, but we can also obtain high-order statistics, e.g., the standard error[35], an estimate of how uncertain the mean estimate is.

Once the model is chosen we can evaluate the final performance on the test set.

A potential disadvantage of K-fold cross-validation is the computational cost of training the model K times, which can be burdensome if the training cost is computationally expensive. In practice, it is often not sufficient to look at the direct parameters alone. For example, we need to explore multiple complexity parameters (e.g., multiple regularization parameters), which may not be direct parameters of the model. Evaluating the quality of the model, depending on these hyper-parameters may result in a number of training runs that is exponential in the number of model parameters.

However, cross-validation is an **embarrassingly parallel** problem, i.e., little effort is needed to separate the problem into a number of parallel tasks. Given sufficient computing resources (e.g., cloud computing, server farms), cross-validation does not require longer than a single performance assessment.

1.7.2 Bayesian Model Selection

There are many approaches to model selection, some of which are covered in this section. Generally, they all attempt to trade off model complexity and data fit:[36] The objective is to find the simplest model that explains the data reasonably well.[37] This concept is also known as **Occam's Razor**. One may consider placing a prior on models that favors simpler models. However, it is not necessary to do this: An "automatic Occam's Razor" is quantitatively embodied in the application of Bayesian probability (Spiegelhalter and Smith, 1980; MacKay, 1992; Jefferys and Berger, 1992). Fig. 1.30 from MacKay (2003) illustrates this property.

Now, let us phrase model selection as a hierarchical inference problem. Generally, we assume a finite number of models that we can choose from. A more efficient way than cross validation, where we have to fit each model K times (where K is the number of cross-validation folds), is to compute the **posterior distribution over models**.

Let us consider a finite number of models $M = \{M_1, \dots, M_K\}$, where each model M_k is parametrized by $\boldsymbol{\theta}_k$. In **Bayesian model selection**, we place a prior $p(M)$ on the set of models. The corresponding **generative process** is

$$M_k \sim p(M) \tag{1.257}$$

$$\boldsymbol{\theta}_k | M_k \sim p(\boldsymbol{\theta}_k) \tag{1.258}$$

$$\mathcal{D} | \boldsymbol{\theta}_k \sim p(\mathcal{D} | \boldsymbol{\theta}_k) \tag{1.259}$$

[35]The standard error is defined as $\frac{\sigma}{\sqrt{K}}$, where K is the number of experiments and σ the standard deviation.

[36]We assume that simpler models are less prone to overfitting than complex models.

[37]If we treat model selection as a hypothesis testing problem, we are looking for the simplest hypothesis that is consistent with the data (Murphy, 2012).

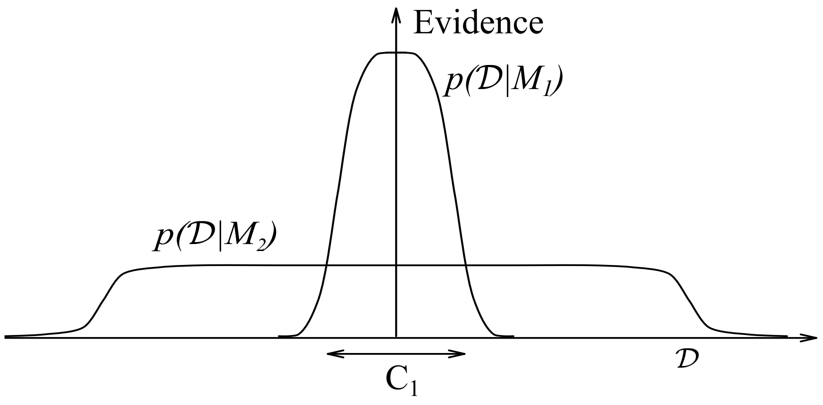


Figure 1.30: "Why Bayesian inference embodies Occam's razor. This figure gives the basic intuition for why complex models can turn out to be less probable. The horizontal axis represents the space of possible data sets $\mathcal{D}$. Bayes' theorem rewards models in proportion to how much they predicted the data that occurred. These predictions are quantified by a normalized probability distribution on $\mathcal{D}$. This probability of the data given model M_i, $p(\mathcal{D}|M_i)$, is called the evidence for M_i. A simple model M_1 makes only a limited range of predictions, shown by $p(\mathcal{D}|M_1)$; a more powerful model M_2 that has, for example, more free parameters than M_1, is able to predict a greater variety of data sets. This means, however, that M_2 does not predict the data sets in region C_1 as strongly as M_1. Suppose that equal prior probabilities have been assigned to the two models. Then, if the data set falls in region C_1, the less powerful model M_1 will be the more probable model." (MacKay, 2003)

and illustrated in Fig. 1.31. Given a training set $\mathcal{D}$, we compute the posterior over models as

$$p(M_k|\mathcal{D}) \propto p(M_k)p(\mathcal{D}|M_k)\,. \tag{1.260}$$

Note that this posterior no longer depends on the model parameters $\boldsymbol{\theta}_k$ because they have been integrated out in the Bayesian setting.
From the posterior in (1.260), we determine the MAP estimate as

$$M^* = \arg\max_{M_k} p(M_k|\mathcal{D})\,. \tag{1.261}$$

With a uniform (uninformative) prior $p(M_k) = \frac{1}{K}$, determining the MAP estimate amounts to picking the model that maximizes the **model evidence (marginal likelihood)**

$$p(\mathcal{D}|M_k) = \int p(\mathcal{D}|\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k|M_k)d\boldsymbol{\theta}_k\,, \tag{1.262}$$

where $p(\boldsymbol{\theta}_k|M_k)$ is the prior distribution of the model parameters $\boldsymbol{\theta}_k$ of model M_k.

Remark 20
There are some important differences between a likelihood and a marginal likelihood: While the likelihood is prone to overfitting, the marginal likelihood is typically not as the model parameters have been marginalized out (i.e., we no longer have to fit the parameters). Furthermore, the marginal likelihood automatically embodies a trade-off between model complexity and data fit.

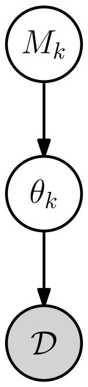


Figure 1.31: Illustration of the hierarchical generative process in Bayesian model selection. We place a prior $p(M)$ on the set of models. For each model, there is a prior $p(\boldsymbol{\theta}_k|M_k)$ on the corresponding model parameters, which are then used to generate the data $\mathcal{D}$.

1.7.3 Bayes Factors for Model Comparison

Consider the problem of comparing two probabilistic models M_1, M_2, given a data set $\mathcal{D}$. If we compute the posteriors $p(M_1|\mathcal{D})$ and $p(M_2|\mathcal{D})$, we can compute the ratio of the posteriors (**posterior odds**)

$$\underbrace{\frac{p(M_1|\mathcal{D})}{p(M_2|\mathcal{D})}}_{\text{posterior odds}} = \frac{\frac{p(\mathcal{D}|M_1)p(M_1)}{p(\mathcal{D})}}{\frac{p(\mathcal{D}|M_2)p(M_2)}{p(\mathcal{D})}} = \underbrace{\frac{p(M_1)}{p(M_2)}}_{\text{prior odds}} \underbrace{\frac{p(\mathcal{D}|M_1)}{p(\mathcal{D}|M_2)}}_{\text{Bayes factor}} \tag{1.263}$$

The first fraction on the right-hand-side (prior odds) measures how much our prior (initial) beliefs favor M_1 over M_2. The ratio of the marginal likelihoods (second fraction on the right-hand-side) is called the **Bayes factor** and measures how well the data $\mathcal{D}$ is predicted by M_1 compared to M_2.

Remark 21
The **Jeffreys-Lindley paradox** *states that the "Bayes factor always favors the simpler model since the probability of the data under a complex model with a diffuse prior will be very small" (Murphy, 2012).*

If we choose a uniform prior over models, the prior odds term in (1.263) is 1, i.e., the posterior odds is the ratio of the marginal likelihoods (Bayes factor)

$$\frac{p(\mathcal{D}|M_1)}{p(\mathcal{D}|M_2)}\,. \tag{1.264}$$

If the Bayes factor is greater than 1, we choose model M_1, otherwise model M_2.

1.7.4 Fully Bayesian Treatment

Instead of selecting a single "best" model, in a fully Bayesian treatment, we integrate out the corresponding model parameters $\boldsymbol{\theta}_k$ and average over all models M_k, $k =$

$1, \dots, K$,

$$p(\mathcal{D}) = \sum_{K=1}^{K} p(M_K) \underbrace{\int p(\mathcal{D}|\boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k|M_k)d\boldsymbol{\theta}_k}_{=p(\mathcal{D}|M_k)}, \tag{1.265}$$

where $p(\mathcal{D}|M_k)$ is the marginal likelihood of model M_k.

1.7.5 Computing the Marginal Likelihood

The marginal likelihood plays an important role in model selection: We need to compute it for Bayes factors (1.263) and in the fully Bayesian setting where we additionally integrate out the model itself (1.265).
Unfortunately, computing the marginal likelihood requires us to solve an integral. This integration is generally analytically intractable, and we will have to resort to approximation techniques, e.g., numerical integration (Stoer and Burlirsch, 2002), stochastic approximations using Monte Carlo (Murphy, 2012) or Bayesian Monte Carlo techniques (O'Hagan, 1991; Rasmussen and Ghahramani, 2003).
However, there are special cases in which we can solve it. In Section 1.2.4, we discussed conjugate models. If we choose a conjugate parameter prior $p(\boldsymbol{\theta})$, we can compute the marginal likelihood in closed form.

Example (Marginal Likelihood Computation)
We consider the linear-Gaussian model with the following generative process (the

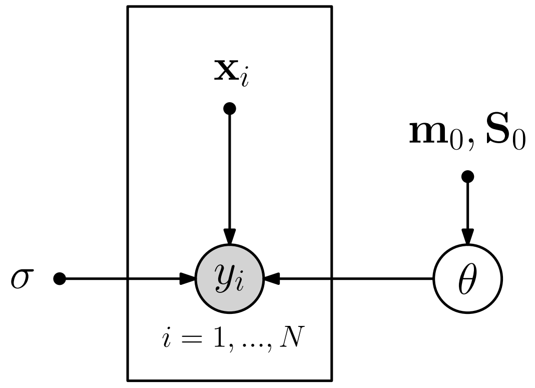


Figure 1.32: Graphical model for Bayesian linear regression.

corresponding graphical model is shown in Fig. 1.32):

$$\boldsymbol{\theta} \sim \mathcal{N}\big(\boldsymbol{m}_0,\, \boldsymbol{S}_0\big) \tag{1.266}$$
$$y_i|\boldsymbol{x}_i, \boldsymbol{\theta} \sim \mathcal{N}\big(\boldsymbol{x}_i^\top\boldsymbol{\theta},\, \sigma^2\big)\,, \tag{1.267}$$

$i = 1, \dots, N$. The marginal likelihood is given as

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} = \int \mathcal{N}\big(\boldsymbol{y}\,|\,\boldsymbol{X}\boldsymbol{\theta},\, \sigma^2\boldsymbol{I}\big)\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_0,\, \boldsymbol{S}_0\big)d\boldsymbol{\theta}\,. \tag{1.268}$$

We compute the marginal likelihood in two steps: First, we show that the marginal likelihood is Gaussian (as a distribution in $\boldsymbol{y}$); Second, we compute the mean and covariance of this Gaussian.

1. The marginal likelihood is Gaussian: From Section 1.2.3.5 we know that (i) the product of two Gaussian random variables is an (unnormalized) Gaussian distribution, (ii) a linear transformation of a Gaussian random variable is Gaussian distributed. In (1.268), we require a linear transformation to bring $\mathcal{N}\big(\boldsymbol{y}\,|\,\boldsymbol{X\theta},\,\sigma^2\boldsymbol{I}\big)$ into the form $\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{\mu},\,\Sigma\big)$ for some $\boldsymbol{\mu},\Sigma$. Once this is done, the integral can be solved in closed form. The result is the normalizing constant of the product of the two Gaussians. The normalizing constant itself has Gaussian shape, see (1.47).

2. Mean and covariance. We compute the mean and covariance matrix of the marginal likelihood by exploiting the standard results for means and covariances of affine transformations of random variables, see Section 1.2.1.2. The mean of the marginal likelihood is computed as

$$\mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{y}|\boldsymbol{X}] = \mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{X\theta}+\boldsymbol{\epsilon}] = \boldsymbol{X}\mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\theta}] = \boldsymbol{X}\boldsymbol{m}_0\,. \tag{1.269}$$

Note that $\boldsymbol{\epsilon} \sim \mathcal{N}\big(\boldsymbol{0},\,\sigma^2\boldsymbol{I}\big)$ is a vector of random variables. The covariance matrix is given as

$$\mathrm{Cov}_{\boldsymbol{\theta}}[\boldsymbol{y}] = \mathrm{Cov}[\boldsymbol{X\theta}] + \sigma^2\boldsymbol{I} = \boldsymbol{X}\,\mathrm{Cov}_{\boldsymbol{\theta}}[\boldsymbol{\theta}]\boldsymbol{X}^\top + \sigma^2\boldsymbol{I} = \boldsymbol{X}\boldsymbol{S}_0\boldsymbol{X}^\top + \sigma^2\boldsymbol{I} \tag{1.270}$$

Hence, the marginal likelihood is

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{x}) = (2\pi)^{-\frac{N}{2}}|\boldsymbol{X}\boldsymbol{S}_0\boldsymbol{X}^\top + \sigma^2\boldsymbol{I}|^{-\frac{1}{2}} \\ \times \exp\big(-\tfrac{1}{2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{m}_0)^\top(\boldsymbol{X}\boldsymbol{S}_0\boldsymbol{X}^\top+\sigma^2\boldsymbol{I})^{-1}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{m}_0)\big)\,. \end{aligned} \tag{1.271}$$

1.7.6 Further Reading

Rasmussen and Ghahramani (2001) showed that the automatic Occam's razor does not necessarily penalize the number of parameters in a model[38] but it is active in terms of the complexity of functions. They also showed that the automatic Occam's razor also holds for Bayesian non-parametric models with many parameters, e.g., Gaussian processes.
If we focus on the maximum likelihood estimate, there exist a number of heuristics for model selection that discourage overfitting. The **Akaike Information Criterion (AIC)** (Akaike, 1974)

$$\log p(\boldsymbol{x}|\boldsymbol{\theta}) - M \tag{1.272}$$

[38]In parametric models, the number of parameters is often related to the complexity of the model class.

corrects for the bias of the maximum likelihood estimator by addition of a penalty term to compensate for the overfitting of more complex models (with lots of parameters). Here, M is the number of model parameters.
The **Bayesian Information Criterion (BIC)** (Schwarz, 1978)

$$\ln p(\boldsymbol{x}) = \ln \int p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \approx \ln p(\boldsymbol{x}|\boldsymbol{\theta}) - \frac{1}{2}M \ln N \tag{1.273}$$

can be used for exponential family distributions. Here, N is the number of data points and M is the number of parameters. The BIC approximates the log-marginal likelihood $\ln p(\boldsymbol{x})$ with the log-likelihood $\ln p(\boldsymbol{x}|\boldsymbol{\theta})$ and a penalty term that depends on the number of parameters. Approximations to the marginal likelihood can be necessary if the marginal likelihood cannot be evaluated easily. The BIC is an asymptotic result derived under the assumptions that the data distribution is in an exponential family. For many data points N (1.273) can be used to approximate the log-marginal likelihood reasonably well. Chickering and Heckerman (1996) provide an overview of other approximations to the marginal likelihood.
Both AIC and BIC can be used to choose among a finite set of models, and the model with the highest AIC/BIC score should be selected. BIC penalizes model complexity more heavily than AIC.

1.8 Bayesian Linear Regression

Thus far, we looked at linear regression models where we estimated the parameters $\boldsymbol{\theta}$, e.g., by means of maximum likelihood or MAP estimation. We discovered that MLE can lead to severe overfitting, in particular, in the small-data regime. MAP estimation addresses this issue to some degree by placing a prior on the parameters that plays the role of a regularizer.
Bayesian linear regression pushes the idea of the parameter prior a step further and does not even attempt to compute a point estimate of the parameters, but instead the full posterior over the parameters is taken into account when making predictions. This means that the parameters themselves remain uncertain.

1.8.1 Model

In Bayesian linear regression, we consider the following model

$$\begin{aligned} y &= \phi^\top(\boldsymbol{x})\boldsymbol{\theta} + \epsilon \\ \epsilon &\sim \mathcal{N}\big(0,\, \sigma^2\big)\,, \\ \boldsymbol{\theta} &\sim \mathcal{N}\big(\boldsymbol{m}_0,\, \boldsymbol{S}_0\big)\,, \end{aligned} \tag{1.274}$$

where we now explicitly place a Gaussian prior $p(\boldsymbol{\theta}) = \mathcal{N}\big(\boldsymbol{m}_0,\, \boldsymbol{S}_0\big)$ on the parameter vector $\boldsymbol{\theta}$.[39] The graphical corresponding graphical model is shown in Fig. 1.33.

[39]Why is a Gaussian prior a convenient choice?

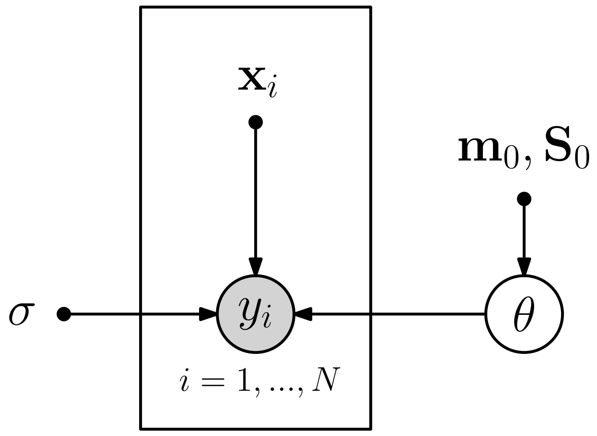


Figure 1.33: Graphical model for Bayesian linear regression.

1.8.2 Parameter Posterior

Given a training set of inputs $\boldsymbol{x}_i \in \mathbb{R}^D$ and corresponding observations $y_i \in \mathbb{R}$, $i = 1, \ldots, N$, compute the posterior over the parameters using Bayes' theorem as

$$p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y}|\boldsymbol{X})}\,, \tag{1.275}$$

where $\boldsymbol{X}$ is the collection of training inputs and $\boldsymbol{y}$ the collection of training targets. Furthermore, $p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})$ is the likelihood, $p(\boldsymbol{\theta})$ the parameter prior, and

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{1.276}$$

the **marginal likelihood/evidence**, which is independent of the parameters $\boldsymbol{\theta}$ and normalizes the posterior. The marginal likelihood is the likelihood averaged over all possible parameter settings (with respect to the prior distribution $p(\boldsymbol{\theta})$).
In our specific model (1.274), the posterior (1.275) can be computed in closed form as

$$p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y}) = \mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_N,\,\boldsymbol{S}_N\big)\,, \tag{1.277}$$

$$\boldsymbol{S}_N = (\boldsymbol{S}_0^{-1} + \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\,, \tag{1.278}$$

$$\boldsymbol{m}_N = \boldsymbol{S}_N(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y})\,, \tag{1.279}$$

where the subscript N indicates the size of the training set. In the following, we will detail how we arrive at this posterior.
Bayes' theorem tells us that the posterior $p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y})$ is proportional to the product of the likelihood $p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})$ and the prior $p(\boldsymbol{\theta})$:

$$p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y}|\boldsymbol{X})}\,, \tag{1.280}$$

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = \mathcal{N}\big(\boldsymbol{y}\,|\,\boldsymbol{\Phi\theta},\,\sigma^2\boldsymbol{I}\big)\,, \tag{1.281}$$

$$p(\boldsymbol{\theta}) = \mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_0,\,\boldsymbol{S}_0\big)\,. \tag{1.282}$$

Looking at the numerator of the posterior in (1.280), we know that the Gaussian prior times the Gaussian likelihood (where the parameters on which we place the Gaussian appear linearly in the mean) is an (unnormalized) Gaussian (see Section 1.2.3.5). If necessary, we can find the normalizing constant if we know the covariance matrix of this unnormalized Gaussian.
Before we can compute this product, e.g., by applying the results from 1.2.3.5, we need to ensure the product has the "right" form

$$\mathcal{N}\big(\boldsymbol{y}\,|\,\boldsymbol{\Phi\theta},\,\sigma^2\boldsymbol{I}\big)\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_0,\,\boldsymbol{S}_0\big) = \mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{\mu},\,\Sigma\big)\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_0,\,\boldsymbol{S}_0\big) \tag{1.283}$$

for some $\boldsymbol{\mu}, \Sigma$. With this form we determine the desired product immediately as

$$\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{\mu},\,\Sigma\big)\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_0,\,\boldsymbol{S}_0\big) \propto \mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_N,\,\boldsymbol{S}_N\big) \tag{1.284}$$

$$\boldsymbol{S}_N = (\boldsymbol{S}_0^{-1} + \Sigma^{-1})^{-1} \tag{1.285}$$

$$\boldsymbol{m}_N = \boldsymbol{S}_N(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \Sigma^{-1}\boldsymbol{\mu})\,. \tag{1.286}$$

In the following, we discuss two ways of finding $\boldsymbol{\mu}$ and Σ.

1.8.2.1 Linear Transformation of Gaussian Random Variables

To find $\boldsymbol{\mu}$ and Σ, we use some basic identities for linearly transforming Gaussian random variables (see Section 1.2.3.5), such that we transform $\boldsymbol{y} = \Phi\boldsymbol{\theta}$ into $\boldsymbol{B}\boldsymbol{y} = \boldsymbol{\theta}$ for a suitable $\boldsymbol{B}$. In the following, we will perform these steps directly on the Gaussian distribution[40] and obtain

$$\mathcal{N}\big(\boldsymbol{y}\,|\,\boldsymbol{\Phi\theta},\,\sigma^2\boldsymbol{I}\big) \overset{\text{scale by }\boldsymbol{\Phi}^\top}{\rightsquigarrow} \mathcal{N}\big(\boldsymbol{\Phi y}\,|\,\boldsymbol{\Phi}^\top\boldsymbol{\Phi\theta},\,\sigma^2\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\big) \tag{1.287}$$

$$\overset{\text{scale by }(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}}{\rightsquigarrow} \mathcal{N}\big((\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y}\,|\,\boldsymbol{\theta},\,\sigma^2(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\big) \tag{1.288}$$

$$= \mathcal{N}\big(\boldsymbol{\theta}\,|\,(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y},\,\sigma^2(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\big)\,. \tag{1.289}$$

If we now use the re-arranged likelihood (1.289) and define its mean as $\boldsymbol{\mu}$ and covariance matrix as Σ in (1.286) and (1.285), respectively, we obtain

$$\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{\mu},\,\Sigma\big)\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_0,\,\boldsymbol{S}_0\big) \propto \mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_N,\,\boldsymbol{S}_N\big) \tag{1.290}$$

$$\boldsymbol{S}_N = (\boldsymbol{S}_0^{-1} + \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1} \tag{1.291}$$

$$\boldsymbol{m}_N = \boldsymbol{S}_N(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \underbrace{\sigma^{-2}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})}_{\boldsymbol{\Sigma}^{-1}}\underbrace{(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y}}_{\boldsymbol{\mu}}) = \boldsymbol{S}_N(\boldsymbol{S}_0^{-1}\boldsymbol{m}_0 + \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y})\,. \tag{1.292}$$

1.8.2.2 Completing the Squares

Instead of looking at the product of the prior and the likelihood, we can transform the problem into log-space and solve for $\boldsymbol{\mu}, \Sigma$ by completing the squares.

$$\log\mathcal{N}\big(\boldsymbol{y}\,|\,\boldsymbol{\Phi\theta},\,\sigma^2\boldsymbol{I}\big) + \log\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_0,\,\boldsymbol{S}_0\big) \tag{1.293}$$

[40] Pay attention to the transformation of the mean and covariance.

$$= -\frac{1}{2}\big(\sigma^{-2}(\boldsymbol{y}-\boldsymbol{\Phi\theta})^\top(\boldsymbol{y}-\boldsymbol{\Phi\theta}) + (\boldsymbol{\theta}-\boldsymbol{m}_0)^\top \boldsymbol{S}_0^{-1}(\boldsymbol{\theta}-\boldsymbol{m}_0)\big) + \text{const} \tag{1.294}$$

where the constant contains terms independent of $\boldsymbol{\theta}$. We will ignore the constant in the following. We now factorize (1.294), which yields

$$-\frac{1}{2}\big(\sigma^{-2}\boldsymbol{y}^\top\boldsymbol{y} - 2\sigma^{-2}\boldsymbol{y}^\top\boldsymbol{\Phi\theta} + \boldsymbol{\theta}^\top\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi\theta} + \boldsymbol{\theta}^\top\boldsymbol{S}_0^{-1}\boldsymbol{\theta} - 2\boldsymbol{m}_0^\top\boldsymbol{S}_0^{-1}\boldsymbol{\theta} + \boldsymbol{m}_0^\top\boldsymbol{S}_0^{-1}\boldsymbol{m}_0\big) \tag{1.295}$$

$$= -\frac{1}{2}\big(\boldsymbol{\theta}^\top(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{S}_0^{-1})\boldsymbol{\theta} - 2(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0)^\top\boldsymbol{\theta}\big) + \text{const}\,, \tag{1.296}$$

where the constant contains the black terms in (1.295), which are independent of $\boldsymbol{\theta}$. By inspecting (1.296), we find that this equation is quadratic in $\boldsymbol{\theta}$. The fact that the unnormalized log-posterior distribution is a (negative) quadratic form implies that the posterior is Gaussian, i.e.,

$$p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y}) = \exp(\log p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y})) \propto \exp(\log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})) \tag{1.297}$$

$$\propto \exp\Big(-\frac{1}{2}\big(\boldsymbol{\theta}^\top(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{S}_0^{-1})\boldsymbol{\theta} - 2(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0)^\top\boldsymbol{\theta}\big)\Big) \tag{1.298}$$

where we just used (1.296) in the last transformation.
The remaining task is it to bring this (unnormalized) Gaussian into the form that is proportional to $\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_N,\,\boldsymbol{S}_N\big)$ for a scaling factor, i.e., we need to identify the mean $\boldsymbol{m}_N$ and the covariance matrix $\boldsymbol{S}_N$.
Here, we use the idea of **completing the squares**. The desired log-posterior is

$$\log\mathcal{N}\big(\boldsymbol{\theta}\,|\,\boldsymbol{m}_N,\,\boldsymbol{S}_N\big) = -\frac{1}{2}\big((\boldsymbol{\theta}-\boldsymbol{m}_N)^\top\boldsymbol{S}_N^{-1}(\boldsymbol{\theta}-\boldsymbol{m}_N)\big) + \text{const} \tag{1.299}$$

$$= -\frac{1}{2}\big(\boldsymbol{\theta}^\top\boldsymbol{S}_N^{-1}\boldsymbol{\theta} - 2\boldsymbol{m}_N^\top\boldsymbol{S}_N^{-1}\boldsymbol{\theta} + \boldsymbol{m}_N\boldsymbol{S}_N^{-1}\boldsymbol{m}_N\big)\,. \tag{1.300}$$

Here, we factorized the quadratic form $(\boldsymbol{\theta}-\boldsymbol{m}_N)^\top\boldsymbol{S}_N^{-1}(\boldsymbol{\theta}-\boldsymbol{m}_N)$ into a term that is quadratic in $\boldsymbol{\theta}$ alone (blue), a term that is linear in $\boldsymbol{\theta}$ (red), and a constant term (black). This allows us now to find $\boldsymbol{S}_N$ and $\boldsymbol{m}_N$ by matching the colored expressions in (1.296) and (1.300), which yields

$$\boldsymbol{S}_N^{-1} = \boldsymbol{\Phi}^\top\sigma^{-2}\boldsymbol{I\Phi} + \boldsymbol{S}_0^{-1} \Leftrightarrow \boldsymbol{S}_N = (\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{S}_0^{-1})^{-1}\,, \tag{1.301}$$

$$\boldsymbol{m}_N^\top\boldsymbol{S}_N^{-1} = (\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0)^\top \Leftrightarrow \boldsymbol{m}_N = \boldsymbol{S}_N(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0)\,. \tag{1.302}$$

This is identical to the solution in (1.291)–(1.292), which we obtained by repeated linear transformation of Gaussian random variables.

Remark 22 (Completing the Squares—General Approach)
If we are given an equation

$$\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{a}^\top\boldsymbol{x} + \mathit{const}_1\,, \tag{1.303}$$

where $\boldsymbol{A}$ *is symmetric and positive definite, which we wish to bring into the form*

$$(\boldsymbol{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}(\boldsymbol{x}-\boldsymbol{\mu}) + \mathit{const}_2\,, \tag{1.304}$$

we can do this by setting

$$\boldsymbol{\Sigma} = \boldsymbol{A} \tag{1.305}$$

$$\boldsymbol{\mu} = \boldsymbol{\Sigma}^{-1}\boldsymbol{a} \tag{1.306}$$

and const$_2$ = *const*$_1 - \boldsymbol{\mu}^\top\boldsymbol{\Sigma}\boldsymbol{\mu}$.

We can see that the terms inside the exponential in (1.298) are of the form (1.303) with

$$\boldsymbol{A} = \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{S}_0^{-1}\,, \tag{1.307}$$

$$\boldsymbol{a} = \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0\,. \tag{1.308}$$

Since $\boldsymbol{A}, \boldsymbol{a}$ can be difficult to identify in equations like (1.295), it is often helpful to bring these equations into the form (1.303) that decouples quadratic term, linear terms and constants, which simplifies finding the desired solution.

Remark 23
The posterior precision (inverse covariance) of the parameters (see (1.301)*, for example)*

$$\boldsymbol{S}_N^{-1} = \boldsymbol{S}_0^{-1} + \frac{1}{\sigma^2}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\,, \tag{1.309}$$

contains two terms: $\boldsymbol{S}_0^{-1}$ *is the prior precision and* $\frac{1}{\sigma^2}\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ *is a data-dependent (precision) term. Both terms (matrices) are symmetric and positive definite. The data-dependent term* $\frac{1}{\sigma^2}\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ *grows as more data is taken into account.*[41] *This means (at least) two things:*

- *The posterior precision grows as more and more data is taken into account (therefore, the covariance shrinks).*
- *The (relative) influence of the parameter prior vanishes for large* N.

1.8.3 Prediction and Inference

In practice, we are usually not so much interested in the parameter values $\boldsymbol{\theta}$. Instead, our focus often lies in the predictions we make with those parameter values. In a fully Bayesian setting, we take the full posterior distribution and average over all plausible parameter settings when we make predictions, instead of finding the maximum a-posteriori (point) estimate of the parameters (the setting $\boldsymbol{\theta}_{\text{MAP}}$ at which the posterior attains its maximum value). To predict the distribution of

$$y_* = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta} + \epsilon\,, \quad \epsilon \sim \mathcal{N}\big(0,\, \sigma^2\big)\,, \tag{1.310}$$

at a test location $\boldsymbol{x}_*$, we compute

$$p(y_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*) = \int p(y_*|\boldsymbol{x}_*, \boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y})d\boldsymbol{\theta} \tag{1.311}$$

[41] $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ is accumulating contributions from the data, not averaging.

$$= \int \mathcal{N}\big(y_* \,|\, \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta},\, \sigma^2\big)\mathcal{N}\big(\boldsymbol{\theta} \,|\, \boldsymbol{m}_N,\, \boldsymbol{S}_N\big)d\boldsymbol{\theta} \tag{1.312}$$

$$= \mathcal{N}\big(y_* \,|\, \boldsymbol{m}_N^\top\boldsymbol{\phi}(\boldsymbol{x}_*),\, \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_*) + \sigma^2\big) \tag{1.313}$$

The term $\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_*)$ reflects the uncertainty associated with the parameters $\boldsymbol{\theta}$, whereas σ^2 is the noise variance. Note that $\boldsymbol{S}_N$ depends on the training inputs $\boldsymbol{X}$, see (1.291). The predictive mean coincides with the MAP estimate.

Remark 24 (Mean and Variance of Noise-Free Function Values)
In many cases, we are not interested in the predictive distribution $p(y_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*)$ of a (noisy) observation. Instead, we would like to obtain the distribution of the (noise-free) latent function values $f(\boldsymbol{x}_*) = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}$. We determine the corresponding moments by exploiting the properties of means and variances, which yields

$$\mathbb{E}[f(\boldsymbol{x}_*)|\boldsymbol{X}, \boldsymbol{y}] = \mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}] = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}] = \boldsymbol{m}_N^\top\boldsymbol{\phi}(\boldsymbol{x}_*)\,, \tag{1.314}$$

$$\mathbb{V}_{\boldsymbol{\theta}}[f(\boldsymbol{x}_*)|\boldsymbol{X}, \boldsymbol{y}] = \mathbb{V}_{\boldsymbol{\theta}}[\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}] = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\mathbb{V}_{\boldsymbol{\theta}}[\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}]\boldsymbol{\phi}(\boldsymbol{x}_*) = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_*) \tag{1.315}$$

Remark 25 (Distribution over Functions)
The fact that we integrate out the parameters $\boldsymbol{\theta}$ induces a distribution over functions: If we sample $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y})$ from the parameter posterior, we obtain a single function realization $\boldsymbol{\theta}_i^\top\boldsymbol{\phi}(\cdot)$. The **mean function***, i.e., the set of all expected function values $\mathbb{E}_{\boldsymbol{\theta}}[f(\cdot)|\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}]$, of this distribution over functions is $\boldsymbol{m}_N^\top\boldsymbol{\phi}(\cdot)$. The (marginal) variances, i.e., the variance of the function $f(\cdot)$, are given by $\boldsymbol{\phi}^\top(\cdot)\boldsymbol{S}_N\boldsymbol{\phi}(\cdot)$.*

1.8.3.1 Derivation

The predictive distribution $p(y_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*)$ is Gaussian: In (1.312), we multiply two Gaussians (in $\boldsymbol{\theta}$), which results in another (unnormalized) Gaussian.[42] When integrating out $\boldsymbol{\theta}$, we are left with the normalization constant, which itself is Gaussian shaped (see Section 1.2.3.5). Therefore, it suffices to determine the mean and the (co)variance of the predictive distribution, which we will do by applying the standard rules for computing means and (co)variances (see Section 1.2.1). In the following, we will use the shorthand notation $\boldsymbol{\phi}_* = \boldsymbol{\phi}(\boldsymbol{x}_*)$.
The mean of the posterior predictive distribution $p(y_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*)$ is

$$\mathbb{E}_{\boldsymbol{\theta},\epsilon}[y_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*] = \mathbb{E}_{\boldsymbol{\theta},\epsilon}[\boldsymbol{\phi}_*^\top\boldsymbol{\theta} + \epsilon|\boldsymbol{X}, \boldsymbol{y}] \tag{1.316}$$

$$= \boldsymbol{\phi}_*^\top\mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}] + 0 \tag{1.317}$$

$$= \boldsymbol{\phi}_*^\top\boldsymbol{m}_N\,. \tag{1.318}$$

Here, we exploited that the noise is i.i.d. and that its mean is 0.
The corresponding posterior predictive variance is

$$\mathbb{V}_{\boldsymbol{\theta},\epsilon}[y_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*] = \mathbb{V}_{\boldsymbol{\theta},\epsilon}[\boldsymbol{\phi}_*^\top\boldsymbol{\theta} + \epsilon|\boldsymbol{X}, \boldsymbol{y}] \tag{1.319}$$

[42]To be precise: We multiply the posterior $p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y})$ with a distribution of a linearly transformed $\boldsymbol{\theta}$. Note that a linear transformation of a Gaussian random variable preserves Gaussianity (see Section 1.2.3.5).

$$\overset{i.i.d.}{=} \mathbb{V}_{\boldsymbol{\theta}}[\boldsymbol{\phi}_*^\top \boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}] + \mathbb{V}_\epsilon[\epsilon] \tag{1.320}$$

$$= \boldsymbol{\phi}_*^\top \mathbb{V}_{\boldsymbol{\theta}}[\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}]\boldsymbol{\phi}_* + \sigma^2 \tag{1.321}$$

$$= \boldsymbol{\phi}_*^\top \boldsymbol{S}_N \boldsymbol{\phi}_* + \sigma^2 \,. \tag{1.322}$$

The blue terms in the variance expression are the terms that are due to the inherent (posterior) uncertainty of the parameters, which induces the uncertainty about the latent function f. The additive green term is the variance of the i.i.d. noise variable.

Example

Fig. 1.34 shows some examples of the posterior distribution over functions, induced by the parameter posterior. The left panels show the maximum likelihood estimate, the MAP estimate (which is identical to the posterior mean function) and the 95% predictive confidence bounds, represented by the shaded area. The right panels show samples from the posterior over functions: Here, we sampled parameters $\boldsymbol{\theta}_i$ from the parameter posterior and computed the function $\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}_i$, which is a single realization of a function under the posterior distribution over functions. For low-order polynomials, the parameter posterior does not allow the parameters to vary much: The sampled functions are nearly identical. When we make the model more flexible by adding more parameters (i.e., we end up with a higher-order polynomial), these parameters are not sufficiently constrained by the posterior, and the sampled functions can be easily visually separated. We also see in the corresponding panels on the left how the uncertainty increases, especially at the boundaries. Although for a 10th-order polynomial the MAP estimate yields a good fit, the Bayesian linear regression model additionally tells us that the posterior uncertainty is huge. This information can be critical, if we use these predictions in a decision-making system, where bad decisions can have significant consequences (e.g., in reinforcement learning or robotics).

Remark 26
Bayesian linear regression naturally equips the estimate with uncertainty induced by the (posterior) distribution on the parameters. In maximum likelihood (or MAP) estimation, we can obtain an estimate of the uncertainty by looking at the point-wise squared distances between the observed values y_i in the training data and the function values $\phi(\boldsymbol{x}_i)^\top\boldsymbol{\theta}$. The variance estimate would then be itself a maximum likelihood estimate and given by

$$\sigma^2_{ML} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \phi(\boldsymbol{x}_i)^\top\boldsymbol{\theta})^2 \,, \tag{1.323}$$

where $\boldsymbol{\theta}$ is a point estimate of the parameters (e.g., maximum likelihood or MAP) and N is the size of the training data set.

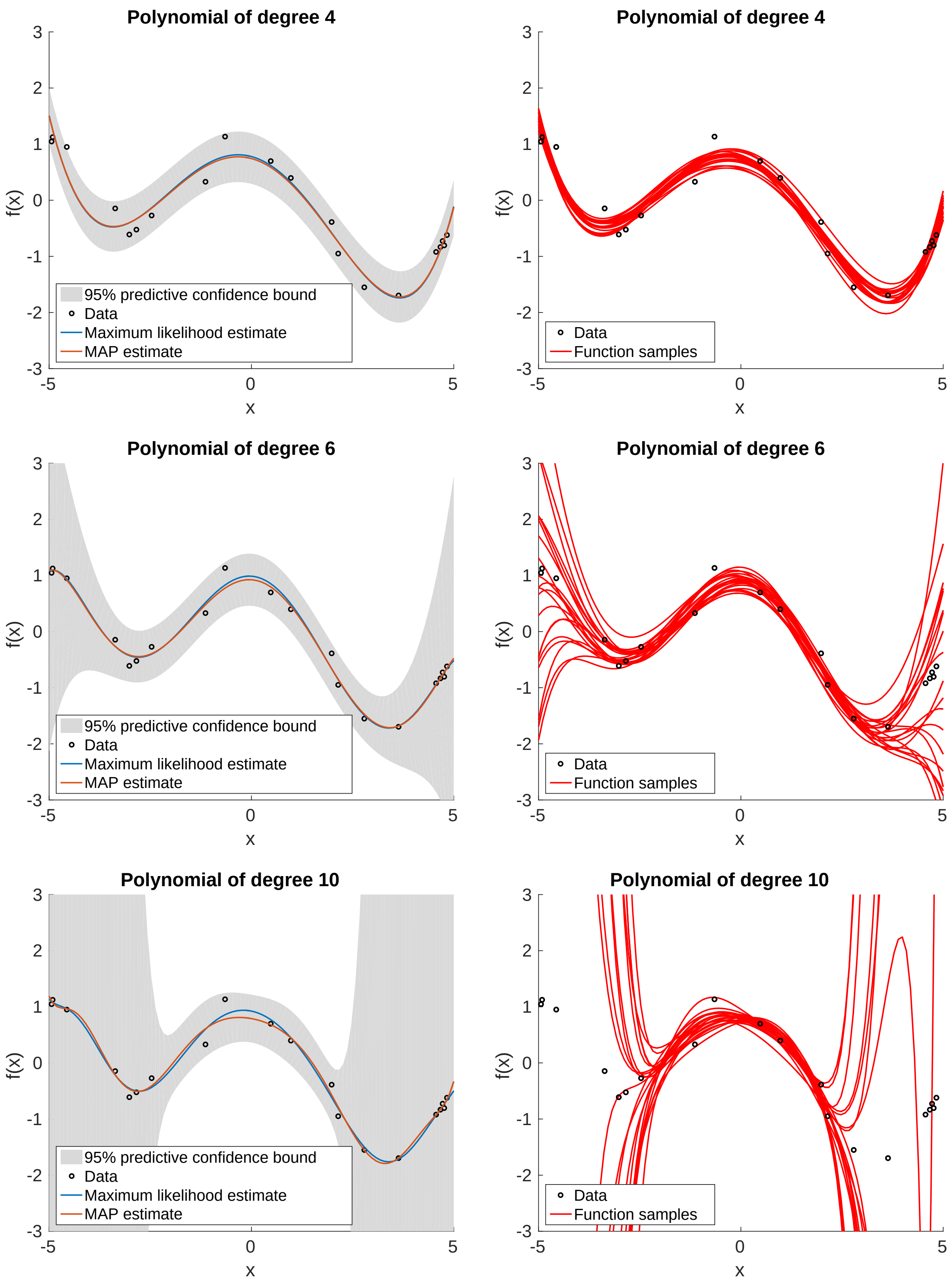


Figure 1.34: Bayesian linear regression. Left panels: The shaded area indicates the 95% predictive confidence bounds. The mean of the Bayesian linear regression model coincides with the MAP estimate. The predictive uncertainty is the sum of the noise term and the posterior parameter uncertainty, which depends on the location of the test input. Right panels: Sampled functions from the posterior distribution.

Chapter 2

Feature Extraction

2.1 Decompositions

In this chapter we will discuss about the use of linear algebra of vectors and matrices in order to define basic feature extraction and dimensionality reduction methodologies. In this context, we will study particular linear decompositions, such as eigen-decomposition and diagonalisations, QR decomposition, Singular Value Decompositions (SVD), etc. The above algebraic decompositions will be used to formulate popular linear feature extraction methodologies such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). We will show that many of these decompositions arise naturally by formulating and solving trace optimisation problems with constraints. We will also study a Maximum Likelihood (ML) solution of a Probabilistic PCA (PPCA) formulation. Finally, we will study some non-linear feature extraction methodologies, using kernel methods. In the following we will be using elements of linear algebra that can be found in Appendix A.1.

2.1.1 Eigen-decomposition

Assume square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ with n linearly independent eigenvectors $\mathbf{q}_i, i = 1,\dots,n$ and n eigenvalues $\lambda_1,\dots,\lambda_n$. Then $\mathbf{A}$ can be factorised as

$$\mathbf{A} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1} \tag{2.1}$$

where $\mathbf{Q} = [\mathbf{q}_1 \dots \mathbf{q}_n]$ and $\boldsymbol{\Lambda}$ is a diagonal matrix whose diagonal elements are the corresponding eigenvalues, i.e. $\boldsymbol{\Lambda} = \mathrm{diag}\{\lambda_1,\dots,\lambda_n\}$.
Using the eigen-decomposition we can compute various powers of $\mathbf{A}$ as

$$\mathbf{A}^k = \mathbf{Q}\boldsymbol{\Lambda}^k\mathbf{Q}^{-1}. \tag{2.2}$$

We can easily verify the above for $k = 2$ as $\mathbf{A}^2 = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1}\mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1} = \mathbf{Q}\boldsymbol{\Lambda}^2\mathbf{Q}^{-1}$. Then, we can easily prove the general case using induction.
In case $k = -1$ we can compute the inverse as

$$\mathbf{A}^{-1} = \mathbf{Q}\boldsymbol{\Lambda}^{-1}\mathbf{Q}^{-1}. \tag{2.3}$$

2.1.1.1 Symmetric Matrices

The eigen-decomposition of symmetric matrices are of particular interest in machine learning. Symmetric matrices have always real eigendecomposition (i.e., all eigenvalues and eigenvectors are real). Furthermore, $\mathbf{Q}$ is an orthogonal matrix (i.e., $\mathbf{Q}^T = \mathbf{Q}^{-1}$). Hence, if $\mathbf{A}$ is symmetric

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T. \tag{2.4}$$

In case that $\mathbf{A}$ is singular (i.e., rank$(\mathbf{A}) = r < n$) then there are $n - r$ eigenvectors that correspond to zero eigenvalues.

$$\mathbf{A} = \mathbf{Q}\begin{bmatrix} \lambda_1 & \dots & 0 & 0 \\ 0 & \ddots & 0 & 0 \\ \mathbf{0} & \dots & \lambda_r & \mathbf{0} \\ 0 & \dots & 0 & \mathbf{0} \end{bmatrix}\mathbf{Q}^T = \mathbf{Q}\begin{bmatrix} \mathbf{\Lambda}_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\mathbf{Q}^T = \mathbf{Q}_r\mathbf{\Lambda}_r\mathbf{Q}_r^T \tag{2.5}$$

where $\mathbf{Q}_r$ is an $n \times r$ matrix of the eigenvectors that correspond to non-zero eigenvalues and $\mathbf{\Lambda}_r$ is $r \times r$ diagonal matrix of the r non-zero eigenvalues.

2.1.2 QR decomposition

Any real square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ can be decomposed as

$$\mathbf{A} = \mathbf{Q}\mathbf{R} \tag{2.6}$$

where $\mathbf{Q}$ is an orthogonal matrix (i.e., $\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}$) and $\mathbf{R}$ is an upper-triangular matrix.
Some important properties of QR decomposition

- If $\mathbf{A}$ is non-singular then its QR decomposition is unique if we require the diagonal elements of $\mathbf{R}$ to be positive.
- If $\mathbf{A}$ is singular then QR decomposition still exists but yields a singular upper triangular $\mathbf{R}$.
- If $\mathbf{A}$ has r linearly independent columns, then the first r columns of $\mathbf{Q}$ form an orthonormal basis for the column space of A.
- $|\det(\mathbf{A})| = |\prod_i r_{ii}|$ (see Appendix A.1.2.12 for a proof).

QR has important applications in solving linear systems, in producing an orthogonal basis for eigenvalue computations etc. In the following we will see some examples.

Example (Solving Linear System)
Assume the following linear system

$$\begin{array}{ccccccc} r_{11}x_1 & +r_{12}x_2 & +r_{13}x_3 & +\cdots & +r_{1(m-1)}x_{m-1} & +r_{1m}x_m & = b_1 \\ 0x_1 & +r_{22}x_2 & +r_{23}x_3 & +\cdots & +r_{2(m-1)}x_{m-1} & +r_{2m}x_m & = b_2 \\ & & & \vdots & & & \\ 0x_1 & +0x_2 & +0x_3 & +\cdots & +r_{(m-1)(m-1)}x_{m-1} & +r_{(m-1)m}x_m & = b_{m-1} \\ 0x_1 & +0x_2 & +0x_3 & +\cdots & +0x_{m-1} & +r_{mm}x_m & = b_m \end{array} \tag{2.7}$$

The above linear system can be written as

$$\mathbf{R}\mathbf{x} = \mathbf{b} \tag{2.8}$$

where $\mathbf{R}$ is an upper triangular matrix, $\mathbf{x} = [x_1, \ldots, x_m]^T$ and $\mathbf{b} = [b_1, \ldots, b_m]^T$.
The above system can be easily solved using the following set of rules (backward elimination)

$$\begin{aligned} x_m &= \frac{b_m}{r_{mm}} \\ x_{m-1} &= \frac{b_{m-1} - r_{(m-1)m}x_m}{r_{(m-1)(m-1)}} \\ &\vdots \\ x_1 &= \frac{b_1 - \sum_{i=2}^{m} r_{1i}x_i}{r_{11}}. \end{aligned} \tag{2.9}$$

The QR decomposition can be used in order to reduce any linear $m \times m$ system $\mathbf{A}\mathbf{x} = \mathbf{b}$ (with $\mathbf{A}$ of full rank) to a system as in (2.8). That is, assume that the QR decomposition of $\mathbf{A}$ is $\mathbf{Q}\mathbf{R}$ then

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \Leftrightarrow \\ \mathbf{Q}\mathbf{R}\mathbf{x} &= \mathbf{b} \Leftrightarrow \\ \mathbf{R}\mathbf{x} &= \mathbf{Q}^T\mathbf{b}. \end{aligned} \tag{2.10}$$

A very important application of the QR decomposition is the QR algorithm for eigenvalue computation.

Example (The QR algorithm for eigenvalue computation)

Assume matrix $\mathbf{A}$. We create the series $\mathbf{A}_k$ by choosing for $k = 0, \mathbf{A}_0 = \mathbf{A}$ and then

$$\begin{aligned} \mathbf{A}_k &= \mathbf{Q}_k\mathbf{R}_k \\ \mathbf{A}_{k+1} &= \mathbf{R}_k\mathbf{Q}_k \end{aligned} \tag{2.11}$$

As can be seen

$$\mathbf{A}_{k+1} = \mathbf{R}_k\mathbf{Q}_k = \mathbf{Q}_k^T\mathbf{Q}_k\mathbf{R}_k\mathbf{Q}_k = \mathbf{Q}_k^T\mathbf{A}_k\mathbf{Q}_k \tag{2.12}$$

so all $\mathbf{A}_k$ are similar and hence they have the same eigenvalues. The matrices $\mathbf{A}_k$ converge to a triangular matrix (Golub and Van Loan (2012)). Hence, the main diagonal of this matrix contains the eigenvalues of $\mathbf{A}$.
As an exercise, run the above algorithm in Matlab for a symmetric matrix and compare the results with the function eig of Matlab. In particular, try to run the code

```
A = randn(100,100);
B = A*A';
```

```
B = 1/2*(B+B'); %This is for stability (making a truly symmetric matrix)
Ao = B;
for i=1:100
    [Q, R] = qr(Ao);
    Ao     = R*Q;
end
```

and compare the eigenvalues computed by `eig(B)`.

2.1.2.1 Gram-Schmidt Process

In the following we will discuss a practical and well-known algorithm for computing the QR decomposition. The algorithm can also be used in order to compute an orthogonal base from a matrix. The algorithm is the so-called Gram-Schmidt (GS) process.
GS process answers the following question: If the columns of $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$ define a basis (not orthonormal) for an inner product space, is there a way to convert it to an orthonormal basis?
We start by geometrically describing the process for two vectors $\mathbf{a}_1$ and $\mathbf{a}_2$. In the first step we normalize the norm of $\mathbf{a}_1$ as $\mathbf{q}_1 = \frac{\mathbf{a}_1}{||\mathbf{a}_1||_2}$. Then, we compute the projection of $\mathbf{a}_2$ onto $\mathbf{q}_1$ as in (A.6)

$$\text{proj}_{\mathbf{q}_1}\mathbf{a}_2 = \frac{\mathbf{q}_1^T\mathbf{a}_2}{||\mathbf{q}_1||_2}\mathbf{q}_1 = (\mathbf{q}_1^T\mathbf{a}_2)\mathbf{q}_1. \tag{2.13}$$

Then, we can compute the vector orthogonal to $\mathbf{q}_1$ as

$$\tilde{\mathbf{q}}_2 = \mathbf{a}_2 - (\mathbf{q}_1^T\mathbf{a}_2)\mathbf{q}_1 \tag{2.14}$$

and normalize it so that it has norm one as $\mathbf{q}_2 = \frac{\tilde{\mathbf{q}}_2}{||\tilde{\mathbf{q}}_2||_2}$.
The general algorithm for computing the orthogonal basis $\mathbf{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_n]$ as

$$\begin{aligned}
\tilde{\mathbf{q}}_1 &= \mathbf{a}_1, \quad \mathbf{q}_1 = \frac{\tilde{\mathbf{q}}_1}{||\tilde{\mathbf{q}}_1||_2} \\
\tilde{\mathbf{q}}_2 &= \mathbf{a}_2 - \text{proj}_{\mathbf{q}_1}\mathbf{a}_2, \quad \mathbf{q}_2 = \frac{\tilde{\mathbf{q}}_2}{||\tilde{\mathbf{q}}_2||_2} \\
\tilde{\mathbf{q}}_3 &= \mathbf{a}_3 - \text{proj}_{\mathbf{q}_1}\mathbf{a}_3 - \text{proj}_{\mathbf{q}_2}\mathbf{a}_3, \quad \mathbf{q}_3 = \frac{\tilde{\mathbf{q}}_3}{||\tilde{\mathbf{q}}_3||_2} \\
\tilde{\mathbf{q}}_n &= \mathbf{a}_n - \sum_{i=1}^{n-1}\text{proj}_{\mathbf{q}_i}\mathbf{a}_n, \quad \mathbf{q}_n = \frac{\tilde{\mathbf{q}}_n}{||\tilde{\mathbf{q}}_n||_2}
\end{aligned} \tag{2.15}$$

The upper triangular matrix $\mathbf{R}$ an be computed by observing that

$$\begin{aligned} \mathbf{a}_1 &= (\mathbf{q}_1^T\mathbf{a}_1)\mathbf{q}_1 \\ \mathbf{a}_2 &= (\mathbf{q}_1^T\mathbf{a}_2)\mathbf{q}_1 + (\mathbf{q}_2^T\mathbf{a}_2)\mathbf{q}_2 \\ &\vdots \\ \mathbf{a}_n &= \sum_{i=1}^{n}(\mathbf{q}_i^T\mathbf{a}_n)\mathbf{q}_i. \end{aligned} \tag{2.16}$$

Hence, $\mathbf{R}$ can be computed as

$$\mathbf{R} = \begin{bmatrix} \mathbf{q}_1^T\mathbf{a}_1 & \mathbf{q}_1^T\mathbf{a}_2 & \cdots & \mathbf{q}_1^T\mathbf{a}_n \\ 0 & \mathbf{q}_2^T\mathbf{a}_2 & \cdots & \mathbf{q}_2^T\mathbf{a}_n \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \mathbf{q}_n^T\mathbf{a}_n \end{bmatrix}. \tag{2.17}$$

Example (The GS procedure)
Let matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ -2 & -3 & 1 \end{bmatrix}$$

perform QR decomposition on $\mathbf{A}$.
$\mathbf{A} = [\mathbf{a}_1 \ \ \mathbf{a}_2 \ \ \mathbf{a}_3]$, then $\mathbf{a}_1 = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix}$, $\mathbf{a}_2 = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix}$, $\mathbf{a}_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$

Let $\mathbf{Q} = [\mathbf{q}_1 \ \ \mathbf{q}_2 \ \ \mathbf{q}_3]$, then since $||\mathbf{a}_1||_2 = \sqrt{6}$, $\mathbf{q}_1 = \mathbf{a}_1/||\mathbf{a}_1||_2 = \begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \\ -\frac{2}{\sqrt{6}} \end{bmatrix}$.

First we compute $\mathbf{q}_1^\mathsf{T}\mathbf{a}_2 = 9/\sqrt{6}$

$$\mathbf{q}_2 = \mathbf{a}_2 - \mathbf{q}_1^T\mathbf{a}_2\mathbf{q}_1 = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix} - \frac{9}{6}\begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix}$$

and $||\mathbf{q}_2||_2 = \sqrt{1/2}$. Then, $\mathbf{q}_2 = \frac{\mathbf{q}_2}{||\mathbf{q}_2||_2} = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}$

$\mathbf{a}_3^T\mathbf{q}_1 = -\frac{1}{\sqrt{6}}$ and $\mathbf{a}_3^T\mathbf{q}_2 = \frac{1}{\sqrt{2}}$. Then,

$$\begin{aligned} \mathbf{q}_3 &= \mathbf{a}_3 - (\mathbf{a}_3^T\mathbf{q}_1)\mathbf{q}_1 - (\mathbf{a}_3^T\mathbf{q}_2)\mathbf{q}_2 \\ &= \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + \frac{1}{6}\begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix} \end{aligned} \tag{2.18}$$

$$||\mathbf{q}_3||_2 = 2/\sqrt{3} \text{ hence } \mathbf{q}_3 = \frac{\mathbf{q}_3}{||\mathbf{q}_3||_2} = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{bmatrix} \tag{2.19}$$

Hence,

$$\mathbf{Q} = \begin{bmatrix} \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ -\frac{2}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{3}} \end{bmatrix}$$

and

$$\mathbf{R} = \begin{bmatrix} \mathbf{q}_1^T\mathbf{a}_1 & \mathbf{q}_1^T\mathbf{a}_2 & \mathbf{q}_1^T\mathbf{a}_3 \\ 0 & \mathbf{q}_2^T\mathbf{a}_2 & \mathbf{q}_2^T\mathbf{a}_3 \\ 0 & 0 & \mathbf{q}_3^T\mathbf{a}_3 \end{bmatrix} = \begin{bmatrix} \sqrt{6} & \frac{9}{\sqrt{6}} & -\frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{2}{\sqrt{3}} \end{bmatrix}$$

2.1.3 Singular Value Decomposition

A very useful matrix decomposition is the Singular Value Decomposition (SVD). If $\mathbf{A} \in \mathbb{R}^{m\times n}$, then there exist orthogonal matrices

$$\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_m] \in \mathbb{R}^{m\times m} \text{ and } \mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_n] \in \mathbb{R}^{n\times n}. \tag{2.20}$$

such that

$$\begin{aligned} \mathbf{U}^T\mathbf{A}\mathbf{V} &= \mathbf{\Sigma} = \text{diag}\{\sigma_1, \dots, \sigma_p\} \in \mathbb{R}^{m\times n}, \ p = \min\{m, n\} \\ \mathbf{A} &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \end{aligned} \tag{2.21}$$

where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$. σ_i are called singular values of $\mathbf{A}$ and the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ are the corresponding i-th left and right singular vectors, respectively.
SVD reveals a lot of information regarding the structure of a matrix. Assume the SVD of matrix $\mathbf{A}$, then we define r by

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq \sigma_{r+1} = \cdots = \sigma_p = 0, \tag{2.22}$$

then

$$\begin{aligned} \text{rank}(\mathbf{A}) &= r \\ \text{null}(\mathbf{A}) &= \text{span}\{\mathbf{v}_{r+1}, \dots, \mathbf{v}_n\} \\ \text{ran}(\mathbf{A}) &= \text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\} \end{aligned} \tag{2.23}$$

and the SVD expansion can be written as

$$\mathbf{A} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T. \tag{2.24}$$

Furthermore, regarding the 2-norm and Frobenius norm we have the following,

$$\begin{aligned} ||\mathbf{A}||_F^2 &= \sigma_1^2 + \ldots + \sigma_p^2, \ p = \min\{m, n\} \\ ||\mathbf{A}||_2 &= \sigma_1. \end{aligned} \tag{2.25}$$

2.1.3.1 Thin SVD

Assume $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V} \in \mathbb{R}^{m\times n}$ is the SVD of $\mathbf{A}$ and $m \geq n$, then

$$\mathbf{A} = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V} \tag{2.26}$$

where

$$\mathbf{U}_1 = [\mathbf{u}_1, \ldots, \mathbf{u}_n] \in \mathbb{R}^{m\times n} \tag{2.27}$$

and

$$\mathbf{\Sigma}_1 = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n\times n}. \tag{2.28}$$

The above is the thin SVD of matrix $\mathbf{A}$.
In thin SVD we have $\mathbf{U}_1^T\mathbf{U}_1 = \mathbf{I}_n$ but $\mathbf{U}_1\mathbf{U}_1^T \neq \mathbf{I}_m$. Furthermore, $\mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{V}^T = \mathbf{I}_m$.

2.1.3.2 Dimensionality Reduction and SVD

A very important application of SVD is the reduction of the rank of a matrix. Rank reduction can be used for dimensionality reduction on the data (e.g., the smallest singular values and the corresponding singular vectors may correspond to noise).

Theorem 1
Let the SVD of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. *If* $k < r = rank(\mathbf{A})$ *and*

$$\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T, \tag{2.29}$$

then

$$\min_{rank(\mathbf{B})=k} ||\mathbf{A} - \mathbf{B}||_2 = ||\mathbf{A} - \mathbf{A}_k||_2 = \sigma_{k+1}. \tag{2.30}$$

Proof. Since, $\mathbf{U}^T\mathbf{A}_k\mathbf{V} = \text{diag}(\sigma_1, \ldots, \sigma_k, 0, \ldots, 0)$ then $\text{rank}(\mathbf{A}_k) = k$ and $\mathbf{U}^T(\mathbf{A} - \mathbf{A}_k)\mathbf{V} = \text{diag}(0, \ldots, 0, \sigma_{k+1}, \ldots, \sigma_p)$. Hence, $||\mathbf{A} - \mathbf{A}_k||_2 = \sigma_{k+1}$.
Now suppose $\text{rank}(\mathbf{B}) = k$ for a $\mathbf{B} \in \mathbb{R}^{m\times n}$. It follows that there is an orthonormal basis $\{\mathbf{q}_1, \ldots, \mathbf{q}_{n-k}\}$ so that $\text{null}(\mathbf{B}) = \text{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_{n-k}\}$. It follows that

$$\text{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_{n-k}\} \cap \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}\} \neq \{0\}. \tag{2.31}$$

That is, there exist unit 2-norm vectors so that $\mathbf{B}\mathbf{z} = \mathbf{0}$ and which can be written as a linear combination $\mathbf{z} = \sum_{i=1}^{k+1} \alpha_i \mathbf{v}_i$ with $\sum_{i=1}^{k+1} \alpha_i^2 = 1$ and $\alpha_i = \mathbf{v}_i^T\mathbf{z}$. Since $\mathbf{B}\mathbf{z} = \mathbf{0}$ and

$$\mathbf{A}\mathbf{z} = \sum_{i=1}^{k+1} \sigma_i(\mathbf{v}_i^T\mathbf{z})\mathbf{u}_i \tag{2.32}$$

we have that

$$||\mathbf{A} - \mathbf{B}||_2^2 \geq ||(\mathbf{A} - \mathbf{B})\mathbf{z}||_2^2 = ||\mathbf{A}\mathbf{z}||_2^2 = \sum_{i=1}^{k+1} \sigma_i^2(\mathbf{v}_i^T\mathbf{z})^2 \geq \sigma_{k+1} \sum_{i=1}^{k+1} \alpha_i^2 = \sigma_{k+1}^2 \tag{2.33}$$

which completes the proof.

The above theorem provides us with a way for dimensionality reduction through rank-reduction. That is, assume that we have a collection of n data samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$. We stack the samples as columns of matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$. Then, the k low-rank representation of the data is

$$\mathbf{X}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T. \tag{2.34}$$

keeping the first k singular values and the corresponding singular vectors.

2.1.4 Principal Component Analysis

In the following we describe one the most well-studied and popular methods for feature extraction and dimensionality reduction. That is, we will describe Principal Component Analysis (PCA). PCA has a very long history in mathematical statistics and engineering. It was first invented by Karl Pearson in 1901 and independently developed by Harold Hotelling. In the following we will introduce PCA under two perspectives. One is the statistical one and the second is based on reconstruction.
We assume that we are given a collection of n d-dimensional samples stored in the columns of matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$. For convenience we will assume that data are centered (i.e., $\sum_{i=1}^n \mathbf{x}_i = \mathbf{0}$). We can always center our data by computing and subtracting the mean of the data $\mathbf{m} = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i$. Data centering can be done efficiently using matrix multiplication (see the example below).

Example (Data Centering using Matrix Multiplication)
In order to compute data centering using matrix multiplication first we need to see how we can create a matrix that contains in all n columns the same vector $\mathbf{a} \in \mathbb{R}^d$

$$\mathbf{A} = [\mathbf{a}, \ldots, \mathbf{a}] = \mathbf{a}\mathbf{1}_n^T. \tag{2.35}$$

where $\mathbf{1}_n \in \mathbf{R}^n$ is a vector.
Using the above, a matrix M that contains in all columns the mean of the data ($\mathbf{m} = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i = \frac{1}{n}\mathbf{X}\mathbf{1}_n$) can be computed as

$$\mathbf{M} = \mathbf{m}\mathbf{1}^T = \mathbf{X}(\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T). \tag{2.36}$$

Hence, data centering can be computed as

$$[\mathbf{x}_1 - \mathbf{m}, \ldots, \mathbf{x}_n - \mathbf{m}] = \mathbf{X} - \mathbf{M} = \mathbf{X}(\mathbf{I} - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T). \tag{2.37}$$

2.1.4.1 Statistical Perspective

We want to find a set of features, via linear projections onto some basis, that maximise the variance of the sample data on that basis. We will start by assuming that we want to find only one vector $\mathbf{w}$ so that the variance of the projected features $y_i = \mathbf{w}_i^T \mathbf{x}_i$ is maximised. The variance can be defined as

$$\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mu_y)^2. \tag{2.38}$$

where $\mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i$. But since we assumed centered data it is easy to verify that $\mu_y = 0$. Hence, the optimal features $\{y_1^o, \dots, y_n^o\}$

$$\begin{aligned}
\{y_1^o, \dots, y_n^o\} &= \arg\max \frac{1}{2}\sigma_y^2 \Rightarrow \\
\mathbf{w}_o &= \arg\max_{\mathbf{w}} \frac{1}{2n}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i)^2 = \arg\max_{\mathbf{w}} \frac{1}{2n}\sum_{i=1}^{n}\mathbf{w}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{w} \\
&= \arg\max_{\mathbf{w}} \frac{1}{2n}\sum_{i=1}^{n}\mathbf{w}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{w} = \arg\max_{\mathbf{w}} \mathbf{w}^T\frac{1}{2n}\mathbf{X}\mathbf{X}^T\mathbf{w} \\
&= \arg\max_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{S}_t\mathbf{w}
\end{aligned} \tag{2.39}$$

where $\mathbf{S}_t = \frac{1}{n}\mathbf{X}\mathbf{X}^T$[1] is a very important matrix called covariance matrix or total-scatter matrix.

The above optimisation problem has a trivial solution which is $\mathbf{w} = \infty$. In order to avoid the trivial solution we incorporate the constraint $||\mathbf{w}||_2 = 1$. Hence optimisation problem (2.39) is now re-formulated as

$$\begin{aligned}
\mathbf{w}_o &= \arg\max_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{S}_t\mathbf{w} \\
\text{subject to } & \ \mathbf{w}^T\mathbf{w} = 1.
\end{aligned} \tag{2.40}$$

In order to solve the above constrained optimisation problem we need to formulate the Lagrangian as

$$L(\mathbf{w}, \lambda) = \frac{1}{2}\mathbf{w}^T\mathbf{S}_t\mathbf{w} - \lambda(\mathbf{w}^T\mathbf{w} - 1). \tag{2.41}$$

We can compute the partial derivatives (using the results in Appendix A.33) as

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \mathbf{S}_t\mathbf{w} - \lambda\mathbf{w} \tag{2.42}$$

forcing $\nabla_{\mathbf{w}} L(\mathbf{w}) = \mathbf{0}$ we end up to

$$\mathbf{S}_t\mathbf{w} = \lambda\mathbf{w} \tag{2.43}$$

which means that $\mathbf{w}$ is an eigenvector of $\mathbf{S}_t$ and the Lagrangian multiplier λ is the corresponding eigenvalue. $\mathbf{S}_t$ is a symmetric positive semi-definite matrix, hence all of its eigenvalues are non-negative.

Now, the question is what pair of eigenvalue and eigenvector, should we choose? Intuitively, someone would answer the eigenvector that corresponds to the largest

[1]We will drop the $\frac{1}{n}$ in the following for convenience.

eigenvalue. But let's see that mathematically, by replacing the solution to the optimisation problem (2.39), we get

$$\lambda_o = \arg\max_{\lambda} \lambda. \tag{2.44}$$

Hence, we have to choose λ to be the largest eigenvalue and $\mathbf{w}$ to be the eigenvector that corresponds to the largest eigenvalue. This vector is also called the principal axis of the data.
Now assume that we want to extract more than one dimension. That is, we want to find $\mathbf{y}_i \in \mathbb{R}^d$ so that

$$\mathbf{y}_i = \begin{bmatrix} y_{1i} \\ \vdots \\ y_{di} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T\mathbf{x}_i \\ \vdots \\ \mathbf{w}_d^T\mathbf{x}_i \end{bmatrix} = \mathbf{W}^T\mathbf{x}_i \tag{2.45}$$

where $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_d]$. We also assume that $\mathbf{W}^T\mathbf{W} = \mathbf{I}$.
The optimisation problem is now written as

$$\begin{aligned} \mathbf{W}_o &= \arg\max_{\mathbf{W}} \frac{1}{2n}\sum_{k=1}^{d}\sum_{i=1}^{n} y_{ki}^2 = \arg\max_{\mathbf{W}} \frac{1}{2n}\sum_{i=1}^{n}\sum_{k=1}^{d}(\mathbf{w}_k^T\mathbf{x}_i)^2 \\ &= \arg\max_{\mathbf{W}} \frac{1}{2n}\sum_{i=1}^{n}\sum_{k=1}^{d}\mathbf{w}_k^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{w}_k \\ &= \arg\max_{\mathbf{W}} \frac{1}{2n}\sum_{k=1}^{d}\mathbf{w}_k^T(\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^T)\mathbf{w}_k = \arg\max_{\mathbf{W}} \frac{1}{2}\sum_{k=1}^{d}\mathbf{w}_k^T\mathbf{S}_t\mathbf{w}_k \\ &= \arg\max_{\mathbf{W}} \text{tr}(\mathbf{W}^T\mathbf{S}_t\mathbf{W}) \\ &\text{subject to}\ \ \mathbf{W}^T\mathbf{W} = \mathbf{I}. \end{aligned} \tag{2.46}$$

The Lagrangian of the above optimisation problem is

$$L(\mathbf{W}, \mathbf{\Lambda}) = \text{tr}(\mathbf{W}^T\mathbf{S}_t\mathbf{W}) - \text{tr}(\mathbf{\Lambda}(\mathbf{W}^T\mathbf{W} - \mathbf{I})) \tag{2.47}$$

where $\mathbf{\Lambda}$ is a matrix with Lagrangian multipliers. Then, we need to estimate $\nabla_{\mathbf{W}} L(\mathbf{W}) = \mathbf{0}$ which leads to

$$\mathbf{S}_t\mathbf{W} = \mathbf{W}\mathbf{\Lambda}. \tag{2.48}$$

which gives

$$\mathbf{S}_t\mathbf{w}_k = \lambda_k\mathbf{w}_k \tag{2.49}$$

for $i = 1, \dots, k$. Hence, $\mathbf{W}$ contains as columns the eigenvectors of $\mathbf{S}_t$.
Now, let's discuss: what d eigenvectors should we keep?
Assume the eigendecomposition of $\mathbf{S}_t$ as

$$\mathbf{S}_t = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \tag{2.50}$$

then $\mathbf{W} = \mathbf{U}_d = [\mathbf{u}_1, \dots, \mathbf{u}_d]$. The cost function in (2.46) can written as

$$\text{tr}(\mathbf{W}^T\mathbf{S}_t\mathbf{W}) = \text{tr}(\mathbf{W}^T\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{W}) = \text{tr}(\mathbf{\Lambda}_d) = \sum_{k=1}^{d}\lambda_k. \tag{2.51}$$

Since $\lambda_k > 0$ maximisation of the above cost function boils down to keeping the eigenvectors that correspond to the k largest eigenvalues.
To conclude, PCA transform looks for d orthogonal direction vectors (known as the principal axes) such that the projection of input sample vectors onto the principal directions has the maximal spread, or equivalently that the variance of the output coordinates is maximal. The principal directions are the first (with respect to descending eigenvalues) k eigenvectors of the covariance matrix $\mathbf{X}\mathbf{X}^T$.

2.1.4.2 Reconstruction Perspective

We will also have another perspective of PCA using the notion of optimal linear reconstruction. Assume again the same set of centered data $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$. We assume we want to find an optimal way to reconstruct the data using a small set of vectors and produce $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_n]$. We want to find the optimal vectors by minimizing the least squares form as

$$\mathbf{W}_o = \arg\min_{\mathbf{W}} \frac{1}{2}\sum_{i=1}^{n} ||\mathbf{x}_i - \tilde{\mathbf{x}}_i||_2^2 = \arg\min_{\mathbf{W}} \frac{1}{2}||\mathbf{X} - \tilde{\mathbf{X}}||_F^2. \tag{2.52}$$

The reconstruction operator using the vectors stored in the columns of $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_d]$ can be written as

$$\mathbf{x}_i = \mathbf{W}\mathbf{y}_i \tag{2.53}$$

which gives $\mathbf{y}_i = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T$.
Replacing it back we get that the optimal reconstruction is

$$\tilde{\mathbf{x}}_i = \mathbf{W}(\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\mathbf{x}_i. \tag{2.54}$$

Imposing that $\mathbf{W}^T\mathbf{W} = \mathbf{I}$ we get

$$\begin{aligned}
\mathbf{W}_o &= \arg\min_{\mathbf{W}} \frac{1}{2}||\mathbf{X} - \tilde{\mathbf{X}}||_F^2 = \arg\min_{\mathbf{W}} ||\mathbf{X} - \mathbf{W}\mathbf{W}^T\mathbf{X}||_F^2 \\
&= \arg\min_{\mathbf{W}} \text{tr}((\mathbf{X} - \mathbf{W}\mathbf{W}^T\mathbf{X})^T(\mathbf{X} - \mathbf{W}\mathbf{W}^T\mathbf{X})) \\
&= \arg\min_{\mathbf{W}} \text{tr}(\mathbf{X}^T\mathbf{X}) - \text{tr}(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}) \\
&= \arg\max_{\mathbf{W}} \text{tr}(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}). \\
\text{subject to } & \ \mathbf{W}^T\mathbf{W} = \mathbf{I}
\end{aligned} \tag{2.55}$$

which is equivalent to the optimisation problem (2.46).

2.1.4.3 Computing PCA

In this section we will discuss how to compute PCA in Small Sampled Size (SSS) problems (i.e., where the original dimensionality of the observations, F, is larger – and in majority of cases is much larger – than the number of samples.
If we want to keep d principal components, the computational cost of the eigen-analysis of an $F \times F$ matrix in order to find d eigenvalues/eigenvectors is $\mathcal{O}(dF^2)$. If

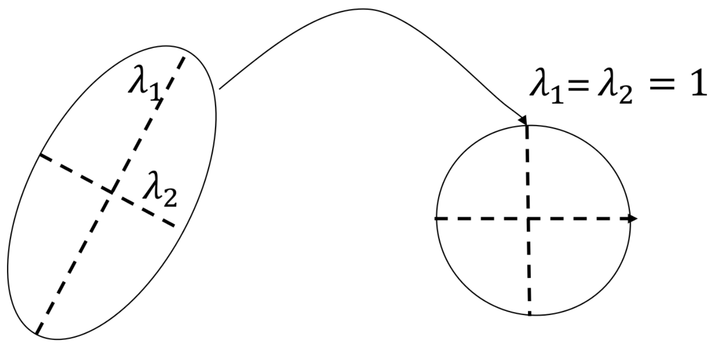


Figure 2.1: Example of data whitening using the PCA projection matrix $\mathbf{W} = \mathbf{U}\mathbf{\Lambda}^{-\frac{1}{2}}$.

F is large, this computation can be quite expensive. We will show how to expedite this procedure when $n \ll F$. We firstly have to introduce the following Lemma.
Lemma 1
Let us assume that $\mathbf{B} = \mathbf{X}\mathbf{X}^\mathbf{T}$ and $\mathbf{C} = \mathbf{X}^\mathbf{T}\mathbf{X}$. It can be proven that $\mathbf{B}$ and $\mathbf{C}$ have the same positive eigenvalues $\mathbf{\Lambda}$ and, assuming that $n < F$, then the eigenvectors $\mathbf{U}$ of $\mathbf{B}$ and the eigenvectors $\mathbf{V}$ of $\mathbf{C}$ are related as $\mathbf{U} = \mathbf{X}\mathbf{V}\mathbf{\Lambda}^{-\frac{1}{2}}$.

Using Lemma 1 we can compute the eigenvectors $\mathbf{U}$ of $\mathbf{S_t} = \mathbf{X}\mathbf{X}^T$ in $\mathcal{O}(n^3)$. The eigenanalysis of $\mathbf{X}^\mathbf{T}\mathbf{X}$ is denoted by

$$\mathbf{X}^\mathbf{T}\mathbf{X} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\mathbf{T} \tag{2.56}$$

where $\mathbf{V}$ is a $n \times (n-1)$ matrix with the eigenvectors as columns and $\mathbf{\Lambda}$ is a $(n-1) \times (n-1)$ diagonal matrix with the eigenvalues in the main diagonal. Given that $\mathbf{V}^\mathbf{T}\mathbf{V} = \mathbf{I}$ and $\mathbf{V}\mathbf{V}^\mathbf{T} \neq \mathbf{I}$ we have

$$\left.\begin{array}{l}\mathbf{X}^\mathbf{T}\mathbf{X} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\mathbf{T} \\ \mathbf{U} = \mathbf{X}\mathbf{V}\mathbf{\Lambda}^{-\frac{1}{2}}\end{array}\right\} \Rightarrow \begin{aligned}\mathbf{U}^\mathbf{T}\mathbf{X}\mathbf{X}^\mathbf{T}\mathbf{U} &= \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}^\mathbf{T}\mathbf{X}^\mathbf{T}\mathbf{X}\mathbf{X}^\mathbf{T}\mathbf{X}\mathbf{V}\mathbf{\Lambda}^{-\frac{1}{2}} = \\ &= \mathbf{\Lambda}^{-\frac{1}{2}}\underbrace{\mathbf{V}^\mathbf{T}\mathbf{V}}_{\mathbf{I}}\mathbf{\Lambda}\underbrace{\mathbf{V}^\mathbf{T}\mathbf{V}}_{\mathbf{I}}\mathbf{\Lambda}\underbrace{\mathbf{V}^\mathbf{T}\mathbf{V}}_{\mathbf{I}}\mathbf{\Lambda}^{-\frac{1}{2}} = \\ &= \mathbf{\Lambda}\end{aligned} \tag{2.57}$$

The pseudocode for computing PCA in SSS problems is

Algorithm 1 Principal Component Analysis

```
1: procedure PCA
2:     Compute dot product matrix: $\mathbf{X}^\mathbf{T}\mathbf{X} = [(\mathbf{x}_i - \mathbf{m})^T(\mathbf{x}_i - \mathbf{m})]$
3:     Eigenanalysis: $\mathbf{X}^\mathbf{T}\mathbf{X} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\mathbf{T}$
4:     Compute eigenvectors: $\mathbf{U} = \mathbf{X}\mathbf{V}\mathbf{\Lambda}^{-\frac{1}{2}}$
5:     Keep specific number of first components: $\mathbf{U_d} = [\mathbf{u}_1, \ldots, \mathbf{u}_d]$
6:     Compute $d$ features: $\mathbf{Y} = \mathbf{U_d}^\mathbf{T}\mathbf{X}$
```

Let's inspect the covariance matrix of low-dimensional features stored in matrix $\mathbf{Y}$. The covariance matrix of $\mathbf{Y}$ is

$$\mathbf{Y}\mathbf{Y}^\mathbf{T} = \mathbf{U}^\mathbf{T}\mathbf{X}\mathbf{X}^\mathbf{T}\mathbf{U} = \mathbf{\Lambda}.$$

From the above it is evident that the features in $\mathbf{Y}$ are un-correlated (i.e., the covariance matrix $\mathbf{YY^T}$ is diagonal, with off-diagonal elements being zero) and the variance in each dimension is equal to the positive eigenvalues of $\mathbf{XX^T}$.
Assume further that we want to make the low-dimensional covariance matrix of the data equal to the identity matrix. This procedure is called whitening (or sphering) which is an important normalisation of the data.
The whitening tranformation is given by the projection matrix

$$\mathbf{W} = \mathbf{U}\mathbf{\Lambda}^{-\frac{1}{2}} \tag{2.58}$$

which normalises the data to have unit variance (Figure 2.1).

2.1.4.4 Link between SVD and PCA

Here we will investigate the relationship between PCA with SVD on a set of centralised data stored in matrix $\mathbf{X}$. The project basis of PCA is given by the r eigenvectors of the the covariance matrix that do not correspond to zero eigenvectors

$$\mathbf{XX}^T = \mathbf{W}_r\mathbf{\Lambda}_r\mathbf{W}_r^T. \tag{2.59}$$

Furthermore, assume the SVD decomposition of

$$\mathbf{X} = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^T. \tag{2.60}$$

Using the above SVD decomposition we can write the covariance matrix as

$$\mathbf{XX}^T = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^T\mathbf{V}_r\mathbf{\Sigma}_r\mathbf{U}_r^T = \mathbf{U}_r\mathbf{\Sigma}_r^2\mathbf{U}_r^T. \tag{2.61}$$

Comparing now the above with the eigen-decomposition in 2.59, we get that (1) the basis of PCA is given by the left orthogonal space of SVD and (2) the eigenvalues are the singular values squared.
Furthermore, the low-dimensional features of PCA are given by

$$\mathbf{Y}_r = \mathbf{U}_r^T\mathbf{X} = \mathbf{U}_r^T\mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^T = \mathbf{\Sigma}_r\mathbf{V}_r^T. \tag{2.62}$$

Hence, the normalised features stacked in matrix $\mathbf{\Sigma}_r^{-1}\mathbf{Y}_r$ are equal to the right orthogonal space of SVD, $\mathbf{V}_r$. That is, the right orthogonal space of SVD is equal to the whitened features.

2.1.5 Linear Discriminant Analysis

In this section we will try to derive a dimensionality reduction methodology that exploits the availability of discrete labels. That is we assume the following scenario: Say we are given a set of data $\mathbf{x}_1, \dots, \mathbf{x}_n$ and a vector $\mathbf{l} = [l_1, \dots, l_n]$ with labels (a label l_i per sample $\mathbf{x}_i$) with $l_i \in 1, \dots, C$ where C is the number of classes we have. Some of the pros and cons of PCA for the above scenario are

- PCA: Unsupervised approach, good for compression of data and data reconstruction. Good statistical prior.

- PCA: Not explicitly defined for classification problems (i.e., in case that data come with labels)
- How do we define a latent space it this case? (i.e., that helps in data classification).

In order to capitalise on the availability of class labels we need to properly define relevant statistical properties which may help us in classification. Intuition: We want to find a space in which:

1. data consisting each class look more like each other, while,
2. data of separate classes look more dissimilar.

The relevant questions we need to answer are:

1. How do I make my data in each class look more similar? (Answer: Minimise the variability in each class (i.e., minimize the variance)).
2. How do I make the data between classes look dissimilar? (Answer: I move the data from different classes further away from each other (i.e., increase the distance between their means)).

2.1.5.1 The two class case

In the following, we will discuss the two-class case. That is, we assume that $C = 2$ and that we have two means $\mu_y(c_1), \mu_y(c_2)$ and two variances $\sigma_y^2(c_1), \sigma_y^2(c_2)$ (one for each class). We want a latent space of y_i such that:

$$\sigma_y^2(c_1) + \sigma_y^2(c_2) \text{ is minimum}$$

$$(\mu_y(c_1) - \mu_y(c_2))^2 \text{ is maximum}$$

How do I combine them together?

$$\text{minimize} \left\{ \frac{\sigma_y^2(c_1) + \sigma_y^2(c_2)}{(\mu_y(c_1) - \mu_y(c_2))^2} \right\}$$

$$\text{Or maximize} \left\{ \frac{(\mu_y(c_1) - \mu_y(c_2))^2}{\sigma_y^2(c_1) + \sigma_y^2(c_2)} \right\}.$$

Assuming again that the low-dimensional features are found through a projection to a vector $\mathbf{w}$ as $y_i = \mathbf{w}^T \mathbf{x}_i$, the within-class variance in the first class c_1

$$\begin{aligned}
\sigma_y^2(c_1) &= \frac{1}{N_{c_1}} \sum_{\mathbf{x}_i \in c_1} (y_i - \mu_y(c_1))^2 \\
&= \frac{1}{N_{c_1}} \sum_{\mathbf{x}_i \in c_1} (\mathbf{w}^\mathsf{T}(\mathbf{x}_i - \boldsymbol{\mu}(c_1)))^2 \\
&= \frac{1}{N_{c_1}} \sum_{\mathbf{x}_i \in c_1} \mathbf{w}^\mathsf{T}(\mathbf{x}_i - \boldsymbol{\mu}(c_1))(\mathbf{x}_i - \boldsymbol{\mu}(c_1))^\mathsf{T}\mathbf{w}
\end{aligned}$$

$$
\begin{aligned}
&= \mathbf{w}^\mathsf{T} \frac{1}{N_{c_1}} \sum_{\mathbf{x}_i \in c_1} (\mathbf{x}_i - \boldsymbol{\mu}(c_1))(\mathbf{x}_i - \boldsymbol{\mu}(c_1))^\mathsf{T} \mathbf{w} \\
&= \mathbf{w}^\mathsf{T} \mathbf{S}_1 \mathbf{w}
\end{aligned}
$$

where $\boldsymbol{\mu}(c_1) = \frac{1}{N_{c_1}} \sum_{\mathbf{x}_i \in c_1} \mathbf{x}_i$ is the mean of the first class.
Similarly, the within-class variance in the second class c_2 is given by

$$\sigma_y^2(c_2) = \mathbf{w}^\mathsf{T} \mathbf{S}_2 \mathbf{w}$$

where $\mathbf{S}_2 = \frac{1}{N_{c_2}} \sum_{\mathbf{x}_i \in c_2} (\mathbf{x}_i - \boldsymbol{\mu}(c_2))(\mathbf{x}_i - \boldsymbol{\mu}(c_2))^T$ and $\boldsymbol{\mu}(c_2)$ is the mean of the second class.
Now, the sum of the two variances can be written as

$$\sigma_y^2(c_1) + \sigma_y^2(c_2) = \mathbf{w}^T(\mathbf{S}_1 + \mathbf{S}_2)\mathbf{w} = \mathbf{w}^T \mathbf{S}_w \mathbf{w}$$

where $\mathbf{S}_w = \mathbf{S}_1 + \mathbf{S}_2$ is the within class scatter matrix.
The distance between the two means can be written as

$$(\mu_y(c_1) - \mu_y(c_2))^2 = \mathbf{w}^\mathsf{T} \underbrace{(\boldsymbol{\mu}(c_1) - \boldsymbol{\mu}(c_2))(\boldsymbol{\mu}(c_1) - \boldsymbol{\mu}(c_2))^T}_{\mathbf{S}_b \text{ between class scatter matrix}} \mathbf{w}$$

The quantity we need to maximise is

$$\frac{(\mu_y(c_1) - \mu_y(c_2))^2}{\sigma_y^2(c_1) + \sigma_y^2(c_2)} = \frac{\mathbf{w}^\mathsf{T} \mathbf{S}_b \mathbf{w}}{\mathbf{w}^\mathsf{T} \mathbf{S}_w \mathbf{w}}$$

The equivalent optimisation problem is

$$\max \mathbf{w}^\mathsf{T} \mathbf{S}_b \mathbf{w} \text{ s.t. } \mathbf{w}^\mathsf{T} \mathbf{S}_w \mathbf{w} = 1$$

In order to solve the optimisation problem we need to formulate the Lagrangian

$$\text{Langrangian: } L(\mathbf{w}, \lambda) = \mathbf{w}^\mathsf{T} \mathbf{S}_b \mathbf{w} - \lambda(\mathbf{w}^\mathsf{T} \mathbf{S}_w \mathbf{w} - 1)$$

$$\frac{\partial \mathbf{w}^\mathsf{T} \mathbf{S}_w \mathbf{w}}{\partial \mathbf{w}} = 2\mathbf{S}_w \mathbf{w} \quad \frac{\partial \mathbf{w}^\mathsf{T} \mathbf{S}_t \mathbf{w}}{\partial \mathbf{w}} = 2\mathbf{S}_t \mathbf{w}$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{0} \Rightarrow \lambda \mathbf{S}_w \mathbf{w} = \mathbf{S}_b \mathbf{w}.$$

Hence the optimal $\mathbf{w}$ is given by the eigenvector that corresponds to the eigenvalue of $\mathbf{S}_w^{-1}\mathbf{S}_b$ (assuming that $\mathbf{S}_w$ is invertible).
In this special case (where $C = 2$) the optimal $\mathbf{w}$ is

$$\mathbf{w} \propto \mathbf{S}_w^{-1}(\boldsymbol{\mu}(c_1) - \boldsymbol{\mu}(c_2)).$$

In the following we will compute the LDA projection for the following 2D dataset.

$$c_1 = \{(4,1), (2,4), (2,3), (3,6), (4,4)\}$$

$$c_2 = \{(9,10),(6,8),(9,5),(8,7),(10,8)\}$$

Solution (by hand)
The class statistics are

$$\mathbf{S}_1 = \begin{bmatrix} 0.8 & -0.4 \\ -0.4 & 2.64 \end{bmatrix}, \ \mathbf{S}_2 = \begin{bmatrix} 1.84 & -0.04 \\ -0.04 & 2.64 \end{bmatrix}$$

The within and between class scatter matrices are

$$\boldsymbol{\mu}_1 = [3.0\ 3.6]^{\mathsf{T}} \quad \boldsymbol{\mu}_2 = [8.4\ 7.6]^{\mathsf{T}}$$

$$\mathbf{S}_b = \begin{bmatrix} 29.16 & 21.6 \\ 21.6 & 16.0 \end{bmatrix}, \ \mathbf{S}_w = \begin{bmatrix} 2.64 & -0.44 \\ -0.44 & 5.28 \end{bmatrix}$$

The LDA projection is then obtained as the solution of the generalized eigenvalue problem

$$\mathbf{S}_w^{-1}\mathbf{S}_b\mathbf{w} = \lambda\mathbf{w} \rightarrow |\mathbf{S}_w^{-1}\mathbf{S}_b - \lambda\mathbf{I}| = 0 \rightarrow$$
$$\begin{vmatrix} 11.89-\lambda & 8.81 \\ 5.08 & 3.76-\lambda \end{vmatrix} = 0 \rightarrow \lambda = 15.65$$

$$\begin{bmatrix} 11.89 & 8.81 \\ 5.08 & 3.76 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = 15.65\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \rightarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0.91 \\ 0.39 \end{bmatrix}.$$

Or directly by

$$\mathbf{w}^* = \mathbf{S}_w^{-1}(\mu_1 - \mu_2) = [-0.91\ \ -0.39]^{\mathsf{T}}.$$

2.1.5.2 Multi-class Case

In the general case, where C classes are available, the within-class scatter matrix is defined as

$$\mathbf{S}_w = \sum_{j=1}^{C}\mathbf{S}_j = \sum_{j=1}^{C}\frac{1}{N_{c_j}}\sum_{\mathbf{x}_i\in c_j}(\mathbf{x}_i - \boldsymbol{\mu}(c_j))(\mathbf{x}_i - \boldsymbol{\mu}(c_j))^{\mathsf{T}}$$

and the between-class scatter matrix as

$$\mathbf{S}_b = \sum_{j=1}^{C}(\boldsymbol{\mu}(c_j) - \mathbf{m})(\boldsymbol{\mu}(c_j) - \mathbf{m})^{\mathsf{T}}$$

Now, we have to find a matrix $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_d]$ by solving the following optimisation problem

$$\max \mathrm{tr}[\mathbf{W}^{\mathsf{T}}\mathbf{S}_b\mathbf{W}] \text{ s.t. } \mathbf{W}^{\mathsf{T}}\mathbf{S}_w\mathbf{W} = \mathbf{I}$$

The Lagrangian of the problem is defined as

$$L(\mathbf{W}, \boldsymbol{\Lambda}) = \mathrm{tr}[\mathbf{W}^{\mathsf{T}}\mathbf{S}_b\mathbf{W}] - \mathrm{tr}[\boldsymbol{\Lambda}(\mathbf{W}^{\mathsf{T}}\mathbf{S}_w\mathbf{W} - \mathbf{I})]$$

$$\frac{\partial \text{tr}[\mathbf{W}^{\mathrm{T}}\mathbf{S}_b\mathbf{W}]}{\partial \mathbf{W}} = 2\mathbf{S}_b\mathbf{W} \quad \frac{\partial \text{tr}[\mathbf{\Lambda}(\mathbf{W}^{\mathrm{T}}\mathbf{S}_w\mathbf{W} - \mathbf{I})]}{\partial \mathbf{W}} = 2\mathbf{S}_w\mathbf{W}\mathbf{\Lambda}$$

$$\frac{\partial L(\mathbf{W}, \mathbf{\Lambda})}{\partial \mathbf{W}} = \mathbf{0} \Rightarrow \mathbf{S}_b\mathbf{W} = \mathbf{S}_w\mathbf{W}\mathbf{\Lambda}.$$

As a result, the columns of $\mathbf{W}$ are the eigenvectors of $\mathbf{S}_w^{-1}\mathbf{S}_b$ (assuming $\mathbf{S}_w$ is not singular) that correspond to its largest eigenvalues (if d are the eigenvectors, then the following must hold: $d \leq C - 1$).

2.2 Computing Linear Discriminant Analysis

In the following we will show how to solve the general LDA optimisation problem in SSS problems (i.e., without having to assume that $\mathbf{S}_w$ is invertible)

$$\begin{aligned} \mathbf{W}_0 = \arg\max_{\mathbf{W}} \quad & \text{tr}(\mathbf{W}^{\mathbf{T}}\mathbf{S}_b\mathbf{W}) \\ \text{subject to} \quad & \mathbf{W}^{\mathbf{T}}\mathbf{S}_w\mathbf{W} = \mathbf{I} \end{aligned} \tag{2.63}$$

Assume that we have C classes in total. We assume that each class has $N_{\mathbf{c}_i}$ samples, stored in matrix $\mathbf{c}_i = [\mathbf{x}_1, \dots, \mathbf{x}_{N_{\mathbf{c}_i}}], i = 1, \dots, N_{\mathbf{c}_i}$, where each $\mathbf{x}_j$ has F dimensions and $\boldsymbol{\mu}(\mathbf{c}_i)$ is the mean vector of the class i. Thus, the overall data matrix $\mathbf{X} = [\mathbf{c}_1, \dots, \mathbf{c}_C]$ has size of $F \times n$ ($n = \sum_{i=1}^{C} N_{\mathbf{c}_i}$). If $\mathbf{m}$ is the overall mean, then the within-class scatter matrix, $\mathbf{S_w}$, is defined as

$$\mathbf{S_w} = \sum_{j=1}^{C} \mathbf{S}_j = \sum_{j=1}^{C} \sum_{\mathbf{x}_i \in \mathbf{c}_j} (\mathbf{x}_i - \boldsymbol{\mu}(\mathbf{c}_j))(\mathbf{x}_i - \boldsymbol{\mu}(\mathbf{c}_j))^T \tag{2.64}$$

and has rank($\mathbf{S_w}$) $= \min(F, n - (C+1))$. Moreover, the between-class scatter matrix, $\mathbf{S_b}$, is defined as

$$\mathbf{S_b} = \sum_{j=1}^{C} N_{\mathbf{c}_j}(\boldsymbol{\mu}(\mathbf{c}_j) - \mathbf{m})(\boldsymbol{\mu}(\mathbf{c}_j) - \mathbf{m})^T \tag{2.65}$$

and has rank($\mathbf{S_b}$) $= \min(F, C - 1)$.
The solution of Eq. 2.63 is given from the generalised eigenvalue problem

$$\mathbf{S_b}\mathbf{W} = \mathbf{S_w}\mathbf{W}\mathbf{\Lambda} \tag{2.66}$$

thus the optimal $\mathbf{W}_o$ corresponds to the eigenvectors of $\mathbf{S}_w^{-1}\mathbf{S}_b$ that correspond to the largest eigenvalues. In order to deal with the singularity of $\mathbf{S}_w$, we can do the following steps:

1. Perform PCA on our data matrix $\mathbf{X}$ to reduce the dimensions to $n - (C+1)$ using the eigenvectors $\mathbf{U}$

2. Solve LDA on this reduced space (i.e., in the space of $\mathbf{Y} = \mathbf{U}^T\mathbf{X}$) and get optimal matrix $\mathbf{Q}$ that has $C - 1$ columns.

3. Compute the total transformation as $\mathbf{W} = \mathbf{U}\mathbf{Q}$.

Unfortunately, if you follow the above procedure it is possible that important information is lost. In the following, we show how the components of LDA can be computed by applying a simultaneous diagonalisation procedure. Before we continue, we need to write some properties regarding the between and within class matrices $\mathbf{S}_w$ and $\mathbf{S}_b$.

Properties
The scatter matrices have some interesting properties. Let us denote

$$\mathbf{M} = \begin{bmatrix} \mathbf{E}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{E}_C \end{bmatrix} = \mathrm{diag}\left\{\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_C\right\} \tag{2.67}$$

where

$$\mathbf{E}_i = \begin{bmatrix} \frac{1}{N_{\mathbf{c}_i}} & \cdots & \frac{1}{N_{\mathbf{c}_i}} \\ \vdots & \ddots & \vdots \\ \frac{1}{N_{\mathbf{c}_i}} & \cdots & \frac{1}{N_{\mathbf{c}_i}} \end{bmatrix}_{N_{\mathbf{c}_i} \times N_{\mathbf{c}_i}} \tag{2.68}$$

Note that M is idempotent, thus MM = M. Given that the data covariance matrix is $\mathbf{S}_t = \mathbf{X}\mathbf{X}^\mathbf{T}$, the between-class scatter matrix can be written as

$$\mathbf{S}_b = \mathbf{X}\mathbf{M}\mathbf{M}\mathbf{X}^\mathbf{T} = \mathbf{X}\mathbf{M}\mathbf{X}^\mathbf{T} \tag{2.69}$$

and the within-class scatter matrix as

$$\mathbf{S}_w = \underbrace{\mathbf{X}\mathbf{X}^\mathbf{T}}_{\mathbf{S_t}} - \underbrace{\mathbf{X}\mathbf{M}\mathbf{X}^\mathbf{T}}_{\mathbf{S_b}} = \mathbf{X}(\mathbf{I} - \mathbf{M})\mathbf{X}^\mathbf{T} \tag{2.70}$$

Thus, we have that $\mathbf{S}_t = \mathbf{S}_w + \mathbf{S}_b$. Note that since $\mathbf{M}$ is idempotent, $\mathbf{I} - \mathbf{M}$ is also idempotent.

Given the above properties, the objective function of Eq. 2.63 can be expressed as

$$\begin{aligned} \mathbf{W}_o = \arg\max_{\mathbf{W}} \quad & \mathrm{tr}(\mathbf{W}^\mathbf{T}\mathbf{X}\mathbf{M}\mathbf{M}\mathbf{X}^\mathbf{T}\mathbf{W}) \\ \text{subject to} \quad & \mathbf{W}^\mathbf{T}\mathbf{X}(\mathbf{I} - \mathbf{M})(\mathbf{I} - \mathbf{M})\mathbf{X}^\mathbf{T}\mathbf{W} = \mathbf{I} \end{aligned} \tag{2.71}$$

The optimisation procedure of this problem involves a procedure called Simultaneous Diagonalisation. Let's assume that the final transformation matrix has the form

$$\mathbf{W} = \mathbf{U}\mathbf{Q} \tag{2.72}$$

We aim to find the matrix $\mathbf{U}$ that diagonalises $\mathbf{S}_w = \mathbf{X}(\mathbf{I} - \mathbf{M})(\mathbf{I} - \mathbf{M})\mathbf{X}^\mathbf{T}$. This practically means that, given the constraint of Eq. 2.71, we want

$$\begin{aligned} &\mathbf{W}^\mathbf{T}\mathbf{X}(\mathbf{I} - \mathbf{M})(\mathbf{I} - \mathbf{M})\mathbf{X}^\mathbf{T}\mathbf{W} = \mathbf{I} \Rightarrow \\ \Rightarrow &\mathbf{Q}^\mathbf{T} \underbrace{\mathbf{U}^\mathbf{T}\mathbf{X}(\mathbf{I} - \mathbf{M})(\mathbf{I} - \mathbf{M})\mathbf{X}^\mathbf{T}\mathbf{U}}_{\mathbf{I}} \mathbf{Q} = \mathbf{I} \end{aligned} \tag{2.73}$$

Consequently, using Eqs. 2.72 and 2.73, the objective function of Eq. 2.71 can be further expressed as

$$\begin{aligned} \mathbf{Q}_o = \arg\max_{\mathbf{Q}} \quad & \operatorname{tr}(\mathbf{Q^T U^T X M M X^T U Q}) \\ \text{subject to} \quad & \mathbf{Q^T Q} = \mathbf{I} \end{aligned} \tag{2.74}$$

where the constraint $\mathbf{W^T X(I-M)(I-M)X^T W} = \mathbf{I}$ now has the form $\mathbf{Q^T Q} = \mathbf{I}$.
Lemma 2
Assume the matrix $\mathbf{X(I-M)(I-M)X^T} = \mathbf{X_w X_w}^{\mathbf{T}}$, where $\mathbf{X_w}$ is the $F \times n$ matrix $\mathbf{X_w} = \mathbf{X(I-M)}$. By performing eigenanalysis on $\mathbf{X_w}^{\mathbf{T}}\mathbf{X_w}$ as $\mathbf{X_w}^{\mathbf{T}}\mathbf{X_w} = \mathbf{V_w \Lambda V_w}^{\mathbf{T}}$, we get $n-(C+1)$ positive eigenvalues, thus $\mathbf{V_w}$ is a $n \times (n-(C+1)$ matrix.

The optimisation problem of Eq. 2.74 can be solved in two steps

1. Find $\mathbf{U}$ such that $\mathbf{U^T X(I-M)(I-M)X^T U} = \mathbf{I}$. By applying Lemma 2, we get $\mathbf{U} = \mathbf{X_w V_w \Lambda}_{\boldsymbol{w}}{}^{-1}$. Note that $\mathbf{U}$ has size $F \times (n-(C+1))$.

2. Find $\mathbf{Q_0}$. By denoting
$$\tilde{\mathbf{X}}_{\mathbf{b}} = \mathbf{U^T X M}$$
the $(n-(C+1)) \times n$ matrix of projected class means, Eq. 2.74 becomes
$$\begin{aligned} \mathbf{Q}_o = \arg\max_{\mathbf{Q}} \quad & \operatorname{tr}(\mathbf{Q^T}\tilde{\mathbf{X}}_{\mathbf{b}}\tilde{\mathbf{X}}_{\mathbf{b}}^{\mathbf{T}}\mathbf{Q}) \\ \text{subject to} \quad & \mathbf{Q^T Q} = \mathbf{I} \end{aligned} \tag{2.75}$$
which is equivalent to applying PCA on the matrix of projected class means. The final $\mathbf{Q_0}$ is a matrix with columns the d eigenvectors of $\tilde{\mathbf{X}}_{\mathbf{b}}\tilde{\mathbf{X}}_{\mathbf{b}}^{\mathbf{T}}$ that correspond to the d largest eigenvalues ($d \leq C-1$).

The final projection matrix is given by

$$\mathbf{W_0} = \mathbf{U Q_0} \tag{2.76}$$

Based on the above, the pseudocode for computing LDA is

Algorithm 2 Linear Discriminant Analysis

```
1: procedure LDA
2:     Find the eigenvectors of $\mathbf{S_w}$ that correspond to the non-zero eigenvalues (usually $n-(C+1)$), i.e. $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_{n-(C+1)}]$ by performing eigen-analysis to $\mathbf{(I-M)X}^T\mathbf{X(I-M)} = \mathbf{V}_w\mathbf{\Lambda}_w\mathbf{V}_w^T$ and computing $\mathbf{U} = \mathbf{X(I-M)V_w\Lambda}_{\boldsymbol{w}}{}^{-1}$ (performing whitening on $\mathbf{S}_w$).
3:     Project the data as $\tilde{\mathbf{X}}_{\mathbf{b}} = \mathbf{U^T X M}$.
4:     Perform PCA on $\tilde{\mathbf{X}}_{\mathbf{b}}$ to find $\mathbf{Q}$ (i.e., compute the eigenanalysis of $\tilde{\mathbf{X}}_{\mathbf{b}}\tilde{\mathbf{X}}_{\mathbf{b}}^T = \mathbf{Q\Lambda}_b\mathbf{Q}^T$).
5:     The total transform is $\mathbf{W} = \mathbf{UQ}$
```

2.2.1 Kernel PCA and Kernel LDA

All the above techniques are linear. But in many cases it would be beneficial to design non-linear feature extraction methods (for examples see the slides). Assume again that we have a set of observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$. We further assume that we have a non-linear mapping ϕ

$$\mathbf{x}_i \in \mathbb{R}^F \rightarrow \phi(\mathbf{x}_i) \in \mathcal{H}$$

$\mathcal{H}$ can be of arbitrary dimensionality space (could be even infinite).
$\phi(.)$ may not be explicitly known or is extremely expensive to compute and store. What is explicitly known is the dot product in $\mathcal{H}$ (also known as kernel k)

$$\phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$$
$$(\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}^{F\times F} \rightarrow \ k(.,.) \in \mathbb{R}$$

All positive (semi)-definite functions can be used as kernels.
Given a training population of n samples $[\mathbf{x}_1, \ldots, \mathbf{x}_n]$, we compute the training kernel matrix (also called Gram matrix)

$$\mathbf{K} = [\phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{x}_j)] = [k(\mathbf{x}_i, \mathbf{x}_j)]$$

All the computations are performed via the use of the kernel or the centralized kernel matrix

$$\bar{\mathbf{K}} = (\phi(\mathbf{x}_i) - \mathbf{m}^\Phi)^\mathsf{T}(\phi(\mathbf{x}_j) - \mathbf{m}^\Phi), \quad \mathbf{m}^\Phi = \frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i)$$

Some popular kernel functions include: Gaussian Radial Basis Function (RBF) kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{r^2}}$$

Polynomial kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\mathsf{T}\mathbf{x}_j + b)^n$$

Hyperbolic Tangent kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\mathbf{x}_i^\mathsf{T}\mathbf{x}_j + b)$$

In kernel literature the original observation space

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$$

is called input space.
While the non-linear space

$$\mathbf{X}^\Phi = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)]$$

is called feature space. Using the feature space the kernel matrix is defined as

$$\mathbf{K} = [\phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{x}_j)] = [k(\mathbf{x}_i, \mathbf{x}_j)] = \mathbf{X}^{\Phi^\mathsf{T}}\mathbf{X}^\Phi.$$

We can define the centralised matrix of features as

$$\begin{aligned}\bar{\mathbf{X}}^{\Phi} &= [\phi(\mathbf{x}_1) - \mathbf{m}^{\Phi}, \ldots, \phi(\mathbf{x}_n) - \mathbf{m}^{\Phi}] \\ &= \mathbf{X}^{\Phi}(\mathbf{I} - \mathbf{E}) = \mathbf{X}^{\Phi}\mathbf{M},\ \mathbf{E} = \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}.\end{aligned}$$

Using the centralised matrix of features the centralised kernel matrix as

$$\begin{aligned}\bar{\mathbf{K}} =&[(\phi(\mathbf{x}_i) - \mathbf{m}^{\Phi})^{\mathsf{T}}(\phi(\mathbf{x}_j) - \mathbf{m}^{\Phi})] = (\mathbf{I} - \mathbf{E})\mathbf{X}^{\Phi^{\mathsf{T}}}\mathbf{X}^{\Phi}(\mathbf{I} - \mathbf{E}) \\ =&(\mathbf{I} - \mathbf{E})\mathbf{K}(\mathbf{I} - \mathbf{E}) = \mathbf{K} - \mathbf{E}\mathbf{K} - \mathbf{K}\mathbf{E} + \mathbf{E}\mathbf{K}\mathbf{E}.\end{aligned}$$

We can now define PCA in the features space or, as it is known, Kernel PCA (KPCA)

$$\begin{aligned}\mathbf{U}_o^{\Phi} =&\arg\max_{\mathbf{U}^{\Phi}} \operatorname{tr}[\mathbf{U}^{\Phi^{\mathsf{T}}}\mathbf{S}_t^{\Phi}\mathbf{U}^{\Phi}] \\ =&\arg\max_{\mathbf{U}^{\Phi}} \operatorname{tr}[\mathbf{U}^{\Phi^{\mathsf{T}}}\overline{\mathbf{X}}^{\Phi^{\mathsf{T}}}\overline{\mathbf{X}}^{\Phi}\mathbf{U}^{\Phi}] \\ \text{subject to } &\mathbf{U}^{\Phi^{\mathsf{T}}}\mathbf{U}^{\Phi} = \mathbf{I}.\end{aligned}$$

The solution is given by the d eigenvectors that correspond to the d largest eigenvalues

$$\mathbf{S}_t^{\Phi}\mathbf{U}_o^{\Phi} = \mathbf{U}_o^{\Phi}\mathbf{\Lambda}$$

Do you see any problem with that? How can we compute the eigenvectors of $\mathbf{S}_t^{\Phi}$. We do not even know ϕ!
Remember our Lemma that links the eigenvectors and eigenvalues of matrices of the form $\mathbf{A}\mathbf{A}^{\mathsf{T}}$ and $\mathbf{A}^{\mathsf{T}}\mathbf{A}$

$$\overline{\mathbf{K}} = \overline{\mathbf{X}}^{\Phi^{\mathsf{T}}}\overline{\mathbf{X}}^{\Phi} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\mathsf{T}} \quad \text{then } \mathbf{U}_o^{\Phi} = \overline{\mathbf{X}}^{\Phi}\mathbf{V}\mathbf{\Lambda}^{-\frac{1}{2}}$$

All computations are performed via the use of $\overline{\mathbf{K}}$ (so called kernel trick)
Still $\mathbf{U}_o^{\Phi} = \overline{\mathbf{X}}^{\Phi}\mathbf{V}\mathbf{\Lambda}^{-\frac{1}{2}}$ cannot be analytically computed. But we do not want to compute $\mathbf{U}_o^{\Phi}$. What we want is to compute latent features. That is, given a test sample $\mathbf{x}_i$, we want to compute $\mathbf{y} = \mathbf{U}_o^{\Phi^{\mathsf{T}}}\phi(\mathbf{x}_t)$ (this can be performed via the kernel trick)

$$\begin{aligned}\mathbf{y} =&\mathbf{U}_o^{\Phi^{\mathsf{T}}}(\phi(\mathbf{x}_t) - \mathbf{m}^{\Phi}) \\ =&\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}^{\mathsf{T}}\overline{\mathbf{X}}^{\Phi^{\mathsf{T}}}(\phi(\mathbf{x}_t) - \mathbf{m}^{\Phi}) \\ =&\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}^{\mathsf{T}}(\mathbf{I} - \mathbf{E})\mathbf{X}^{\Phi^{\mathsf{T}}}\left(\phi(\mathbf{x}_t) - \frac{1}{n}\mathbf{X}^{\Phi}\mathbf{1}\right) \\ =&\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}^{\mathsf{T}}(\mathbf{I} - \mathbf{E})\left(\mathbf{X}^{\Phi^{\mathsf{T}}}\phi(\mathbf{x}_t) - \frac{1}{n}\mathbf{X}^{\Phi^{\mathsf{T}}}\mathbf{X}^{\Phi}\mathbf{1}\right) \\ =&\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}^{\mathsf{T}}(\mathbf{I} - \mathbf{E})\left(g(\mathbf{x}_t) - \frac{1}{n}\mathbf{K}\mathbf{1}\right)\end{aligned}$$

where

$$g(\mathbf{x}_t) = \mathbf{X}^{\Phi^{\mathsf{T}}}\phi(\mathbf{x}_t) = \begin{bmatrix} \phi(\mathbf{x}_1)^{\mathsf{T}}\phi(\mathbf{x}_t) \\ \cdots \\ \phi(\mathbf{x}_n)^{\mathsf{T}}\phi(\mathbf{x}_t) \end{bmatrix} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_t) \\ \cdots \\ k(\mathbf{x}_n, \mathbf{x}_t) \end{bmatrix}.$$

Kernel LDA is a tutorial exercise.

2.2.1.1 Maximum Likelihood for Probabilistic PCA

PCA, as defined above, is a deterministic procedure. In the following, we will discuss about a probabilistic counterpart of PCA, the so-called Probabilistic PCA (PPCA). PPCA forms the basis for a Bayesian treatment of PCA in which the dimensionality of the principal subspace can be found automatically from the data. Furthermore, PPCA can be used to model class-conditional densities and hence be applied to classification problems. Finally, PPCA model can be run generatively to provide samples from the distribution.
The probabilistic model of PPCA can be written as

$$\begin{aligned} \mathbf{x}_i &= \mathbf{W}\mathbf{y}_i + \mathbf{m} + \mathbf{e}_i \\ \mathbf{e}_i &\sim \mathcal{N}(\mathbf{e}_i|\mathbf{0}, \sigma^2\mathbf{I}) \\ \mathbf{y}_i &\sim \mathcal{N}(\mathbf{y}_i|\mathbf{0}, \mathbf{I}) \end{aligned} \tag{2.77}$$

or equivalently

$$\begin{aligned} p(\mathbf{x}_i|\mathbf{y}_i, \mathbf{W}, \sigma) = \mathcal{N}(\mathbf{x}_i|\mathbf{W}\mathbf{y}_i + \mathbf{m}, \sigma^2\mathbf{I}) &= \frac{1}{\sqrt{(2\pi)^F\sigma^F}} \exp\left(-\frac{1}{2\sigma^2}(\mathbf{x}_i - \mathbf{m} - \mathbf{W}\mathbf{y}_i)^T(\mathbf{x}_i - \mathbf{m} - \mathbf{W}\mathbf{y}_i)\right) \\ p(\mathbf{y}_i) = \mathcal{N}(\mathbf{y}_i|\mathbf{0}, \mathbf{I}) &= \frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2}\mathbf{y}_i^T\mathbf{y}_i\right). \end{aligned} \tag{2.78}$$

Given the conditional probability $p(\mathbf{x}_i|\mathbf{y}_i, \mathbf{W}, \sigma)$ and prior $p(\mathbf{y})$ we can compute the two following important distributions

$$\begin{aligned} p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{W}, \sigma),\ &\text{posterior} \\ p(\mathbf{x}_i|\mathbf{W}, \sigma),\ &\text{marginal.} \end{aligned} \tag{2.79}$$

We apply the technique called "completing the square" in order to do so.

$$p(\mathbf{x}_i|\mathbf{W}, \sigma) = \int_{\mathbf{y}_i} p(\mathbf{x}_i, \mathbf{y}_i|\mathbf{W}, \sigma)d\mathbf{y}_i = \int_{\mathbf{y}_i} p(\mathbf{x}_i|\mathbf{y}_i, \mathbf{W}, \sigma)p(\mathbf{y}_i)d\mathbf{y}_i. \tag{2.80}$$

$$p(\mathbf{x}_i|\mathbf{y}_i, \mathbf{W}, \sigma)p(\mathbf{y}_i) = \frac{1}{\sqrt{(2\pi)^F\sigma^F}\sqrt{(2\pi)^d}} \exp\left(-\frac{1}{2\sigma^2}\left((\mathbf{x}_i - \mathbf{m} - \mathbf{W}\mathbf{y}_i)^T(\mathbf{x}_i - \mathbf{m} - \mathbf{W}\mathbf{y}_i) + \sigma^2\mathbf{y}_i^T\mathbf{y}_i\right)\right) \tag{2.81}$$

Now, in order to compute the marginal, as well as the posterior, we will restructure the exponent of the exponential. The aim of the restructure is to reveal a term that does not depend on $\mathbf{y}_i$, so that it can be safely go out of the integral in (2.80). This term is used to produce the marginal. The other term is used to produce the posterior. Let's now see how to restructure the exponent (for convenience let's assume $\bar{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{m}$)

$$\begin{aligned} &(\bar{\mathbf{x}}_i - \mathbf{W}\mathbf{y}_i)^T(\bar{\mathbf{x}}_i - \mathbf{W}\mathbf{y}_i) + \sigma^2\mathbf{y}_i^T\mathbf{y}_i \\ &= \bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_i - 2\bar{\mathbf{x}}_i^T\mathbf{W}\mathbf{y}_i + \mathbf{y}_i^T\mathbf{W}^T\mathbf{W}\mathbf{y}_i + \sigma^2\mathbf{y}_i^T\mathbf{y}_i \\ &= \bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_i - 2\bar{\mathbf{x}}_i^T\mathbf{W}\mathbf{y}_i + \mathbf{y}_i^T(\mathbf{W}^T\mathbf{W} + \sigma^2\mathbf{I})\mathbf{y}_i \\ &= \bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_i - 2\bar{\mathbf{x}}_i^T\mathbf{W}\mathbf{y}_i + \mathbf{y}_i^T\mathbf{M}\mathbf{y}_i \end{aligned} \tag{2.82}$$

where $\mathbf{M} = \mathbf{W}^T\mathbf{W} + \sigma^2\mathbf{I}$.
We observe that we have a quadratic term $\mathbf{y}_i^T\mathbf{M}\mathbf{y}_i$ and we need some extra terms in order to complete its quadratic form. We can do so as follows

$$\begin{aligned}
&\bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_i - 2\bar{\mathbf{x}}_i^T\mathbf{W}\mathbf{y}_i + \mathbf{y}_i^T\mathbf{M}\mathbf{y}_i \\
&= \bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_i - 2(\mathbf{M}^{-1}\mathbf{W}^T\bar{\mathbf{x}}_i)^T\mathbf{M}\mathbf{y}_i + \mathbf{y}_i^T\mathbf{M}\mathbf{y}_i \\
&= \bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_i - 2(\mathbf{M}^{-1}\mathbf{W}^T\bar{\mathbf{x}}_i)^T\mathbf{M}\mathbf{y}_i + \mathbf{y}_i^T\mathbf{M}\mathbf{y}_i + (\mathbf{M}^{-1}\mathbf{W}^T\bar{\mathbf{x}}_i)^T\mathbf{M}(\mathbf{M}^{-1}\mathbf{W}^T\bar{\mathbf{x}}_i) - (\mathbf{M}^{-1}\mathbf{W}^T\bar{\mathbf{x}}_i)^T\mathbf{M}(\mathbf{M}^{-1}\mathbf{W}^T \\
&= (\mathbf{y}_i - \mathbf{M}^{-1}\mathbf{W}^T\bar{\mathbf{x}}_i)^T\mathbf{M}(\mathbf{y}_i - \mathbf{M}^{-1}\mathbf{W}^T\bar{\mathbf{x}}_i) + \bar{\mathbf{x}}_i^T(\mathbf{I} - \mathbf{W}\mathbf{M}^{-1}\mathbf{W}^T)\bar{\mathbf{x}}_i.
\end{aligned} \tag{2.83}$$

Hence, after straightforward computations (i.e., putting the exponent back to the integral), we have that

$$\begin{aligned}
p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{W}, \sigma) &= \mathcal{N}(\mathbf{y}_i|\mathbf{M}^{-1}\mathbf{W}^T(\mathbf{x}_i - \mathbf{m}), \sigma^2\mathbf{M}^{-1}) \\
p(\mathbf{x}_i|\mathbf{W}, \sigma) &= \mathcal{N}(\mathbf{x}_i|\mathbf{m}, (\sigma^{-2}\mathbf{I} - \sigma^{-2}\mathbf{W}\mathbf{M}^{-1}\mathbf{W}^T)^{-1})
\end{aligned} \tag{2.84}$$

In order to simplify the marginal we will make use of the Woodbury identity

$$(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{V}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{A}^{-1}. \tag{2.85}$$

Using the above we can easily verify that if $\mathbf{D} = \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}$ then

$$\mathbf{D}^{-1} = \sigma^{-2}\mathbf{I} - \sigma^{-2}\mathbf{W}\mathbf{M}^{-1}\mathbf{W}^T = \sigma^{-2}\mathbf{I} - \sigma^{-2}\mathbf{W}(\sigma^2\mathbf{I} + \mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T. \tag{2.86}$$

Hence, the marginal can be written as

$$p(\mathbf{x}_i|\mathbf{W}, \sigma) = \mathcal{N}(\mathbf{x}_i|\mathbf{m}, \sigma^2\mathbf{I} + \mathbf{W}\mathbf{W}^T). \tag{2.87}$$

Having computed the marginal, we are ready to formulate a maximum likelihood framework in order to estimate the parameters of the model, i.e., $\theta = \{\mathbf{W}, \mathbf{m}, \sigma\}$, given a set of training data samples $\mathbf{x}_1, \dots, \mathbf{x}_n$

$$\begin{aligned}
\theta_o &= \arg\max_\theta \ln p(\mathbf{x}_1, \dots, \mathbf{x}_n|\theta) \\
&= \arg\max_\theta \ln \prod_{i=1}^n p(\mathbf{x}_i|\theta) \\
&= \arg\max_\theta \left\{ -\frac{nd}{2}\ln(2\pi) - \frac{n}{2}\ln\det(\mathbf{D}) - \frac{1}{2}\sum_{i=1}^n (\mathbf{x}_i - \mathbf{m})^T\mathbf{D}^{-1}(\mathbf{x}_i - \mathbf{m}) \right\}.
\end{aligned} \tag{2.88}$$

Hence, by removing the constant terms, the function we want to optimise with regards to the parameters is

$$\begin{aligned}
L(\mathbf{W}, \sigma, \mathbf{m}) &= \frac{n}{2}\ln\det(\mathbf{D}) - \frac{1}{2}\sum_{i=1}^n (\mathbf{x}_i - \mathbf{m})^T\mathbf{D}^{-1}(\mathbf{x}_i - \mathbf{m}) \\
&= \frac{n}{2}\ln\det(\mathbf{D}) - \frac{1}{2}\sum_{i=1}^n \mathrm{tr}(\mathbf{D}^{-1}(\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T) \\
&= \frac{n}{2}\ln\det(\mathbf{D}) - \frac{n}{2}\mathrm{tr}(\mathbf{D}^{-1}\mathbf{S}_t)
\end{aligned} \tag{2.89}$$

We will now take the derivative of the function with regards to the parameters

$$\begin{aligned} \nabla_{\mathbf{m}} L = \mathbf{0} &\Rightarrow \mathbf{m} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i \\ \nabla_{\mathbf{W}} L = \mathbf{0} &\Rightarrow \mathbf{D}^{-1}\mathbf{S}_t\mathbf{D}^{-1}\mathbf{W} - \mathbf{D}^{-1}\mathbf{W} \Rightarrow \mathbf{S}_t\mathbf{D}^{-1}\mathbf{W} = \mathbf{W} \end{aligned} \quad . \tag{2.90}$$

There are three different solutions. The first is $\mathbf{W} = \mathbf{0}$ (not useful). The second is $\mathbf{D} = \mathbf{S}_t$. In this case, if $\mathbf{S}_t = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ is the covariance matrix eigendecomposition, then $\mathbf{W} = \mathbf{U}(\mathbf{\Lambda} - \sigma^2\mathbf{I})^{\frac{1}{2}}\mathbf{V}^T$ for an arbitrary rotation matrix $\mathbf{V}$ (i.e., $\mathbf{V}^T\mathbf{V} = \mathbf{I}$), $\mathbf{D} \neq \mathbf{S}_t$ and $\mathbf{W} \neq 0$ $d < q = \text{rank}(\mathbf{S}_t)$.
Assume the SVD of $\mathbf{W} = \mathbf{U}\mathbf{L}\mathbf{V}^{\mathsf{T}}$

$$\mathbf{U} = [\mathbf{u}_1 \dots \mathbf{u}_d] \; F \times d \text{ matrix}$$
$$\mathbf{U}^{\mathsf{T}}\mathbf{U} = \mathbf{I}, \; \mathbf{V}^{\mathsf{T}}\mathbf{V} = \mathbf{V}\mathbf{V}^{\mathsf{T}} = \mathbf{I}$$
$$\mathbf{L} = \begin{bmatrix} l_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & l_d \end{bmatrix}$$

$$\mathbf{S}_t\mathbf{D}^{-1}\mathbf{U}\mathbf{L}\mathbf{V}^{\mathsf{T}} = \mathbf{U}\mathbf{L}\mathbf{V}^{\mathsf{T}}$$

Let's study $\mathbf{D}^{-1}\mathbf{U}$.

$$\begin{aligned} \mathbf{D}^{-1} &= (\mathbf{W}\mathbf{W}^{\mathsf{T}} + \sigma^2\mathbf{I})^{-1} \xRightarrow{\mathbf{W}=\mathbf{U}\mathbf{L}\mathbf{V}^{\mathsf{T}}} \\ &= (\mathbf{U}\mathbf{L}^2\mathbf{U}^{\mathsf{T}} + \sigma^2\mathbf{I})^{-1} \end{aligned}$$

Assume a set of bases $\mathbf{U}_{F-d}$ such that $\mathbf{U}_{F-d}^{\mathsf{T}}\mathbf{U} = 0$ and $\mathbf{U}_{F-d}^{\mathsf{T}}\mathbf{U}_{F-d}^{\mathsf{T}} = \mathbf{I}$. We then have

$$\begin{aligned} \mathbf{D}^{-1} &= \left(\mathbf{U}\mathbf{L}\mathbf{U}^T + \sigma^2\mathbf{I}\right)^{-1} \\ &= \left([\mathbf{U}\;\mathbf{U}_{F-d}]\begin{bmatrix} \mathbf{L}^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}[\mathbf{U}\;\mathbf{U}_{F-d}]^{\mathsf{T}} + [\mathbf{U}\;\mathbf{U}_{F-d}]\sigma^2\mathbf{I}[\mathbf{U}\;\mathbf{U}_{F-d}]^{\mathsf{T}}\right)^{-1} \\ &=[\mathbf{U}\;\mathbf{U}_{F-d}]\begin{bmatrix} \mathbf{L}^2 + \sigma^2\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \sigma^2\mathbf{I} \end{bmatrix}^{-1}[\mathbf{U}\;\mathbf{U}_{F-d}]^{\mathsf{T}} \\ &=[\mathbf{U}\;\mathbf{U}_{F-d}]\begin{bmatrix} (\mathbf{L}^2 + \sigma^2\mathbf{I})^{-1} & \mathbf{0} \\ \mathbf{0} & \sigma^{-2}\mathbf{I} \end{bmatrix}[\mathbf{U}\;\mathbf{U}_{F-d}]^T \end{aligned}$$

And subsequently

$$\begin{aligned} \mathbf{D}^{-1}\mathbf{U} &= [\mathbf{U}\;\mathbf{U}_{F-d}]\begin{bmatrix} (\mathbf{L}^2 + \sigma^2\mathbf{I})^{-1} & \mathbf{0} \\ \mathbf{0} & \sigma^{-2}\mathbf{I} \end{bmatrix}[\mathbf{U}\;\mathbf{U}_{F-d}]^{\mathsf{T}}\mathbf{U} \\ &= [\mathbf{U}\;\mathbf{U}_{F-d}]\begin{bmatrix} (\mathbf{L}^2 + \sigma^2\mathbf{I})^{-1} & \mathbf{0} \\ \mathbf{0} & \sigma^{-2}\mathbf{I} \end{bmatrix}[\mathbf{I}\;\mathbf{0}]^{\mathsf{T}} \\ &= [\mathbf{U}\;\mathbf{U}_{F-d}]\begin{bmatrix} (\mathbf{L}^2 + \sigma^2\mathbf{I})^{-1} \\ \mathbf{0} \end{bmatrix} \end{aligned}$$

$$= \mathbf{U}(\mathbf{L}^2 + \sigma^2\mathbf{I})^{-1}$$

As a result, we have

$$\begin{aligned} \mathbf{S}_t\mathbf{D}^{-1}\mathbf{U}\mathbf{L}\mathbf{V}^{\mathsf{T}} &= \mathbf{U}\mathbf{L}\mathbf{V}^{\mathsf{T}} \\ \mathbf{S}_t\mathbf{U}(\mathbf{L}^2 + \sigma^2\mathbf{I})^{-1} &= \mathbf{U} \\ \mathbf{S}_t\mathbf{U} &= \mathbf{U}(\mathbf{L}^2 + \sigma^2\mathbf{I}) \end{aligned}$$

Hence we have that $\mathbf{S}_t\mathbf{u}_i = (l_i^2 + \sigma^2)\mathbf{u}_i$. For $\mathbf{S}_t = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathsf{T}}$, $\mathbf{u}_i$ are the eigenvectors of $\mathbf{S}_t$ and $\lambda_i = l_i^2 + \sigma^2 \Rightarrow l = \sqrt{\lambda - \sigma^2}$. Unfortunately, $\mathbf{V}$ cannot be determined thus there is a rotation ambiguity.
Concluding, the optimum $\mathbf{W}_d$ is given by (keeping d eigenvectors)

$$\mathbf{W}_d = \mathbf{U}_d(\mathbf{\Lambda}_d - \sigma^2\mathbf{I})^{\frac{1}{2}}\mathbf{V}^{\mathsf{T}}$$

Having computed $\mathbf{W}$, we need to compute the optimum σ^2

$$\begin{aligned} L(\mathbf{W}, \sigma^2, \boldsymbol{\mu}) &= -\frac{NF}{2}\ln(2\pi) - \frac{N}{2}\ln(|\mathbf{D}|) - \frac{N}{2}\text{tr}[\boldsymbol{D}^{-1}\boldsymbol{S}_t] \\ = -\frac{NF}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{W}\mathbf{W}^{\mathsf{T}} + \sigma^2\mathbf{I}| &- \frac{N}{2}\text{tr}[(\mathbf{W}\mathbf{W}^{\mathsf{T}} + \sigma^2\mathbf{I})^{-1}\mathbf{S}_t] \end{aligned}$$

$$\begin{aligned} \mathbf{W}_d\mathbf{W}_d^{\mathsf{T}} + \sigma^2\mathbf{I} &= [\mathbf{U}_d\ \mathbf{U}_{F-d}]\begin{bmatrix} \mathbf{\Lambda}_d - \sigma^2\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}[\mathbf{U}_d\ \mathbf{U}_{F-d}]^{\mathsf{T}} \\ &+ [\mathbf{U}_d\ \mathbf{U}_{F-d}]\begin{bmatrix} \sigma^2 & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \sigma^2 \end{bmatrix}[\mathbf{U}_d\ \mathbf{U}_{F-d}]^{\mathsf{T}} \\ &= [\mathbf{U}_d\ \mathbf{U}_{F-d}]\begin{bmatrix} \mathbf{\Lambda}_d & \mathbf{0} \\ \mathbf{0} & \sigma^2\mathbf{I} \end{bmatrix}[\mathbf{U}_d\ \mathbf{U}_{F-d}]^{\mathsf{T}} \end{aligned}$$

Hence

$$|\mathbf{W}_d\mathbf{W}_d^{\mathsf{T}} + \sigma^2\mathbf{I}| = \prod_{i=1}^{d}\lambda_i \prod_{i=d+1}^{F}\sigma^2$$

And thus the log of the determinant is given by

$$\ln|\mathbf{W}_d\mathbf{W}_d^{\mathsf{T}} + \sigma^2\mathbf{I}| = (F-d)\ln\sigma^2 + \sum_{i=1}^{d}\ln\lambda_i$$

We also have that

$$\mathbf{D}^{-1}\mathbf{S}_t = [\mathbf{U}_d\ \mathbf{U}_{F-d}]\begin{bmatrix} \mathbf{\Lambda}_d & \mathbf{0} \\ \mathbf{0} & \sigma^2\mathbf{I} \end{bmatrix}^{-1}[\mathbf{U}_d\ \mathbf{U}_{F-d}]^{\mathsf{T}}[\mathbf{U}_d\ \mathbf{U}_{F-d}]\begin{bmatrix} \mathbf{\Lambda}_d & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}_{q-d} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

$$\begin{aligned}
&[\mathbf{U}_d\ \mathbf{U}_{F-d}]^\mathsf{T} \\
&= [\mathbf{U}_d\ \mathbf{U}_{F-d}] \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{1}{\sigma^2}\mathbf{\Lambda}_{q-d} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} [\mathbf{U}_d\ \mathbf{U}_{F-d}]^\mathsf{T} \\
&\Rightarrow \text{tr}(\mathbf{D}^{-1}\mathbf{S}_t) = \frac{1}{\sigma^2}\sum_{i=d+1}^{q}\lambda_i + d
\end{aligned}$$

$$L(\sigma^2) = -\frac{N}{2}\left\{F\ln 2\pi + \sum_{j=1}^{d}\ln\lambda_j + (F-d)\ln\sigma^2 + \frac{1}{\sigma^2}\sum_{j=d+1}^{q}\lambda_j + d\right\}$$

$$\frac{\partial L}{\partial \sigma} = 0 \Rightarrow -2\sigma^{-3}\sum_{j=d+1}^{q}\lambda_j + \frac{2(F-d)}{\sigma} = 0 \Rightarrow \sigma^2 = \frac{1}{F-d}\sum_{j=d+1}^{q}\lambda_j$$

Putting the solution back, we have

$$L(\sigma^2) = -\frac{N}{2}\left\{\sum_{j=1}^{d}\ln\lambda_j + (F-d)\ln\frac{1}{F-d}\sum_{j=d+1}^{q}\lambda_j + F\ln 2\pi + F\right\}$$

$$\begin{aligned}
L(\sigma^2) = -\frac{N}{2}\{\underbrace{\sum_{j=1}^{d}\ln\lambda_j + \sum_{j=d}^{q}\ln\lambda_j}_{\ln|\mathbf{S}_t|} - \sum_{j=d}^{q}\ln\lambda_j \\
+(F-d)\ln\frac{1}{F-d}\sum_{j=d+1}^{q}\lambda_j + F\ln 2\pi + F\}
\end{aligned}$$

$$\max\frac{N}{2}\left\{\frac{1}{F-d}\ln|\mathbf{S}_t| - \frac{1}{F-d}\sum_{j=d}^{q}\ln\lambda_j + \ln\left(\frac{1}{F-d}\sum_{j=d+1}^{q}\ln\lambda_j\right) + \text{const.}\right\}$$

$$\Rightarrow \min\left\{\ln\left(\frac{1}{F-d}\sum_{j=d}^{q}\ln\lambda_j\right) - \frac{1}{F-d}\sum_{j=d}^{q}\ln\lambda_j\right\}$$

Taking into account Jensen inequality

$$\ln\left(\frac{\sum_{i=1}^{n} r_i}{n}\right) \geq \frac{1}{n}\sum_{i=1}^{n}\ln r_i$$

we have that

$$\ln\left(\frac{1}{F-d}\sum_{j=d+1}^{q}\lambda_j\right) \geq \frac{1}{F-d}\sum_{j=d+1}^{q}\ln\lambda_j$$

Hence

$$\Rightarrow \ln\left(\frac{1}{F-d}\sum_{j=d}^{q}\ln\lambda_j\right) - \frac{1}{F-d}\sum_{j=d}^{q}\ln\lambda_j \geq 0$$

Therefore, the function is minimised when the discarded eigenvectors are the ones that correspond to the $q-d$ eigenvalues.
A brief summary:

$$\sigma^2 = \frac{1}{F-d}\sum_{j=d+1}^{q}\lambda_j$$
$$\mathbf{W}_d = \mathbf{U}_d(\mathbf{\Lambda}_d - \sigma^2\mathbf{I})^{\frac{1}{2}}\mathbf{V}^{\mathsf{T}}$$
$$\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$$

We no longer have a projection but: $\mathbb{E}_{p(\mathbf{y}_i|\mathbf{x}_i)}\{\mathbf{y}_i\} = \mathbf{M}^{-1}\mathbf{W}^{\mathsf{T}}(\mathbf{x}_i - \boldsymbol{\mu})$. We also have a reconstruction $\hat{\mathbf{x}}_i = \mathbf{W}\mathbb{E}_{p(\mathbf{y}_i|\mathbf{x}_i)}\{\mathbf{y}_i\} + \boldsymbol{\mu}$. We can notice that

$$\lim_{\sigma^2\to 0}\mathbf{W}_d = \mathbf{U}_d\mathbf{\Lambda}_d^{\frac{1}{2}}$$

$$\lim_{\sigma^2\to 0}\mathbf{M} = \mathbf{W}_d^{\mathsf{T}}\mathbf{W}_d$$

Hence,

$$\begin{aligned}\lim_{\sigma^2\to 0}\mathbb{E}_{p(\mathbf{y}_i|\mathbf{x}_i)}\{\mathbf{y}_i\} &= \mathbf{M}^{-1}\mathbf{W}_d^{\mathsf{T}}(\mathbf{x}_i - \boldsymbol{\mu}) \\ &= \mathbf{\Lambda}_d^{-\frac{1}{2}}\mathbf{U}_d(\mathbf{x}_i - \boldsymbol{\mu})\end{aligned}$$

which gives PCA.

Chapter 3

Support Vector Machines

3.1 Support Vector Classification

In the following, we will touch upon quadratic optimisation problems with constraints in order to see in more details how the methods of Lagrangian multipliers work. Furthermore, we will study how the dual optimisation problem is formulated and solved. We will study this in the context of Support Vector Machines (SVMs) for classification and regression.

3.1.1 Linear Separating Hyperplane with Maximal Margin

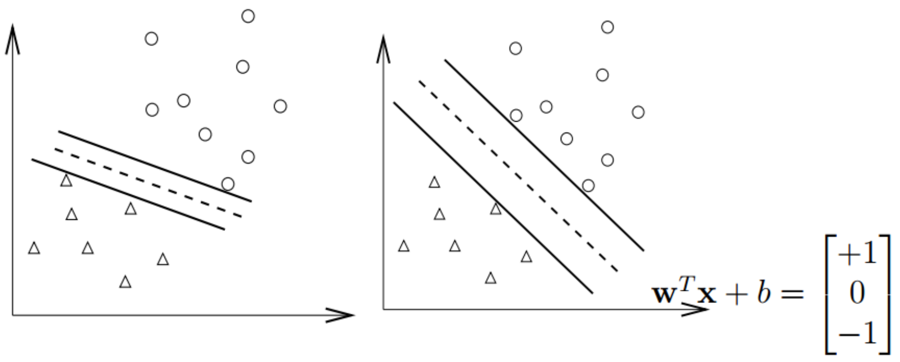
The original idea of SVM classification is to use a linear separating hyperplane to create a classifier. Given training vectors $\mathbf{x}_i,\ i = 1, \dots, n$ with $\mathbf{x}_i \in \mathbb{R}^F$, a vector $\mathbf{y}$ is defined as follows

$$y_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ in class } 1 \\ -1 & \text{if } \mathbf{x}_i \text{ in class } 2 \end{cases}$$

The SVM technique tries to find the separating hyperplane with the largest margin between two classes, measured along a line perpendicular to the hyperplane. For example, in Figure 3.1, the two classes could be fully separated by a dotted line $\mathbf{w}^{\mathsf{T}}\mathbf{x} + b = 0$. We would like to decide the line with the largest margin. In other words, intuitively we think that the distance between two classes of training data should be as large as possible. That means we find a line with parameters $\mathbf{w}$ and b such that the distance between $\mathbf{w}^{\mathsf{T}}\mathbf{x} + b = \pm 1$ is maximised.
The distance between $\mathbf{w}^{\mathsf{T}}\mathbf{x} + b = 1$ and -1 can be calculated by the following way. Consider a point $\tilde{\mathbf{x}}$ on $\mathbf{w}^{\mathsf{T}}\mathbf{x} + b = -1$ (see Figure 3.2). As $\mathbf{w}$ is the "normal vector" of the line $\mathbf{w}^{\mathsf{T}}\mathbf{x} + b = -1$, $\mathbf{w}$ and the line are perpendicular to each other. Starting from $\tilde{\mathbf{x}}$ and moving along the direction $\mathbf{w}$, we assume $\tilde{\mathbf{x}} + t\mathbf{w}$ touches line $\mathbf{w}^{\mathsf{T}}\mathbf{x} + b = 1$. Therefore,

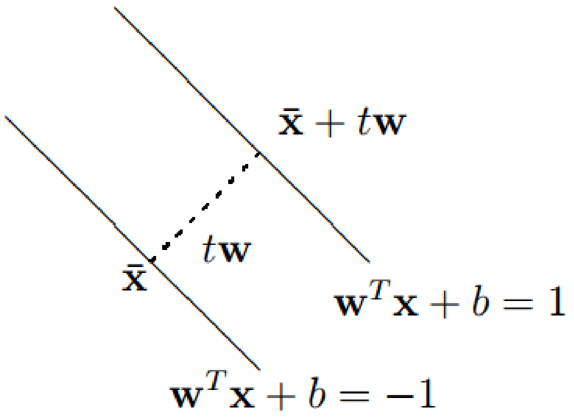
$$\mathbf{w}^{\mathsf{T}}(\tilde{\mathbf{x}} + t\mathbf{w}) + b = 1 \text{ and } \mathbf{w}^{\mathsf{T}}\tilde{\mathbf{x}} + b = -1$$

We then have $t\mathbf{w}^{\mathsf{T}}\mathbf{w} = 2$, so the distance (i.e., the length of $t\mathbf{w}$) is $||t\mathbf{w}||_2 = 2\frac{||\mathbf{w}||_2}{\mathbf{w}^{\mathsf{T}}\mathbf{w}} = \frac{2}{||\mathbf{w}||_2}$. Note that $||\mathbf{w}||_2 = \sqrt{w_1^2 + \cdots + w_n^2}$. As maximising $\frac{2}{||\mathbf{w}||_2}$ is equivalent to min-



$\mathbf{w}^T\mathbf{x} + b = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}$

Figure 3.1: Separating Hyperplanes.



$\bar{\mathbf{x}} + t\mathbf{w}$
$t\mathbf{w}$
$\bar{\mathbf{x}}$
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = -1$

Figure 3.2: Distance between hyperplanes.

imising $\frac{\mathbf{w}^\mathsf{T}\mathbf{w}}{2}$, we have the following problem:

$$\begin{aligned} \min_{w,b} \quad & \tfrac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} \\ \text{subject to} \quad & y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1 \\ & i = 1, \ldots, l \end{aligned} \tag{3.1}$$

The constraint $y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1$ means

$$\begin{aligned} (\mathbf{w}^\mathsf{T}\mathbf{x}_i) + b \geq 1 \text{ if } y_i = 1, \\ (\mathbf{w}^\mathsf{T}\mathbf{x}_i) + b \leq -1 \text{ if } y_i = -1, \end{aligned} \tag{3.2}$$

That is, data in the class 1 must be on the right-hand side of $\mathbf{w}^\mathsf{T}\mathbf{x}+b = 0$ while data in the other class must be on the left-hand side. Note that the reason of maximising the distance between $\mathbf{w}^\mathsf{T}\mathbf{x} + b = \pm 1$ is based on Vapnik's Structural Risk Minimisation. The following example gives a simple illustration of maximal-margin separating hyperplanes:

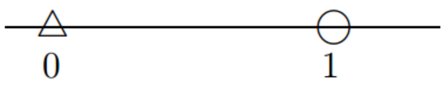


Figure 3.3: 1D Toy Example

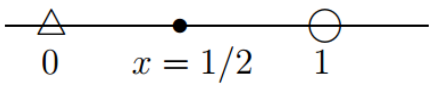


Figure 3.4: Solution Toy

Example
Given two training data in $\mathbb{R}^1$ as in the following Figure 3.3: What is the separating hyperplane?
We have two data points, namely $x_1 = 1$, $x_2 = 0$ with $\mathbf{y} = [+1, -1]^\mathsf{T}$. Furthermore, $w \in \mathbb{R}^1$, so Eq. 3.1 becomes

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}w^2 \\ \text{subject to} \quad & w \cdot 1 + b \geq 1 & (3.3) \\ & -1(w \cdot 0 + b) \geq 1 & (3.4) \end{aligned}$$

From Ineq. 3.4, $-b \geq -1$. Putting this into Ineq. 3.3, $w \geq 2$. In other words, for any (w, b) which satisfies 3.3 and 3.4, we have $w \geq 2$. As we are minimising $\frac{1}{2}w^2$, the smallest possibility is $w = 2$. Thus, $(w, b) = (2, -1)$ is the optimal solution. The separating hyperplane is $2x - 1 = 0$, in the middle of the two training data points (Figure 3.4).

In order to find the optimal $\mathbf{w}$ in the general case we need to solve optimisation problem 3.1. Before doing so, we need some basic knowledge regarding Lagrangian optimisation and Lagrangian duality.

3.1.1.1 Lagrangian Duality

The problem which we have to solve is a constrained optimisation problem. It is of the form

$$\begin{aligned} \min_{w} \quad & f(\mathbf{w}) \\ \text{subject to} \quad & g(\mathbf{w}) \leq 0. & (3.5) \end{aligned}$$

By convention, we write that $g(\mathbf{w}) \leq 0$, as a result this means that we multiply the constrains from (3.5) by minus one.

To solve this we use the method of Lagrange multipliers. We define the Lagrangian to be the original objective function added to a weighted combination of the constraints. The weights are called Lagrange multipliers. It will be helpful to focus on the simpler case with one inequality constraint and one Lagrange multiplier.

$$L(\mathbf{w}, a) = f(\mathbf{w}) + a g(\mathbf{w}) \tag{3.6}$$

Theorem 2
The original minimisation problem can be written as

$$\min_{\mathbf{w}} \max_{\mathbf{a} \geq 0} L(\mathbf{w}, a) \tag{3.7}$$

Proof: Looking at the inner term we get

$$\max_{a \geq 0} L(\mathbf{w}, a) = \begin{cases} f(\mathbf{w}), & g(\mathbf{w}) \leq 0 \\ \infty, & g(\mathbf{w}) > 0 \end{cases}$$

This is because when $g(\mathbf{w}) \leq 0$, we maximise (3.6) by setting $a = 0$. When $g(\mathbf{w}) > 0$, one can drive the value to infinity by setting a to a large number. Minimising the outer term, one sees that we obtain the minimum value of $f(\mathbf{w})$ such that the constraint $g(\mathbf{w}) \leq 0$ holds. Therefore, we can say that the two problems are equivalent. The primal solution to the problem is given by

$$\mathbf{p}^* = \min_{\mathbf{w}} \max_{a \geq 0} L(\mathbf{w}, a) \tag{3.8}$$

The dual solution to the problem is given by

$$\mathbf{d}^* = \max_{a \geq 0} \min_{\mathbf{w}} L(\mathbf{w}, a) \tag{3.9}$$

We claim that $\mathbf{d}^* \leq \mathbf{p}^*$. Let $\mathbf{w}^*$ be the $\mathbf{w}$ value that corresponds to the optimal primal solution $\mathbf{p}^*$. We can write for all $a \geq 0$

$$\max_{\tilde{a} \geq 0} L(\mathbf{w}^*, \tilde{a}) \geq L(\mathbf{w}^*, a) \geq \min_{\mathbf{w}} L(\mathbf{w}, a). \tag{3.10}$$

The Left-Hand-Side (LHS) of the above is obviously $\mathbf{p}^*$. This means we can interpret the Right-Hand-Side (RHS) as a lower bound on p^* for all $a \geq 0$. One obtains the best lower bound when maximising over a - this yields $\mathbf{d}^*$. Hence $\mathbf{d}^* \leq \mathbf{p}^*$ for any $f(\mathbf{w})$ and $g(\mathbf{w})$. However, if certain conditions are met, namely

- $f(\mathbf{w})$ is convex
- $g(\mathbf{w})$ is affine (e.g., $g(\mathbf{w}) = \mathbf{w}^T\mathbf{x} + b$)

then $\mathbf{d}^* = \mathbf{p}^*$.
For the SVM problem, both of these conditions hold. Finally, in order to solve the SVM optimisation problem using the dual, we need to further explore the optimality conditions.

3.1.1.2 Conditions for Optimality (Karush-Kuhn-Tucker Conditions)

Lagrangian duality theory also states a number of necessary and sufficient conditions that hold at the optimum solution.

1. $\mathbf{w}$ and a are feasible
2. $ag(\mathbf{w}) = 0$

Condition 1 means that $g(\mathbf{w}) \leq 0$ and $a \geq 0$. Condition 2 is called "complimentary slackness condition". It follows from the fact that the constraint $g(\mathbf{w})$ may or may not affect the final solution. If the minimum of $f(\mathbf{w})$ lies within the region $\{\mathbf{w} : g(\mathbf{w}) < 0\}$, then one can optimise $f(\mathbf{w})$ without the constraint (i.e., let $a = 0$). If the minimum of $f(\mathbf{w})$ lies outside this set, then the constraint is turned "on", and the final solution must satisfy $g(\mathbf{w}) = 0$. In this case, a behaves like a typical Lagrange multiplier for an equality constraint.

3.1.1.3 SVM dual problem

We are now ready to formulate the Lagrangian for optimisation problem (3.1). Since we have n data, we have also n constraints, one for each sample.
The Lagrangian is hence formulated as

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n} a_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1). \tag{3.11}$$

The solution of the dual problem is

$$\max_{a_i \geq 0} \min_{\mathbf{w}, b} L(\mathbf{w}, b, \mathbf{a}) \tag{3.12}$$

Since we are optimising a convex function with linear constraints, the dual solution will equal the primal solution. To optimise the dual (3.12), we need to minimise $L(\mathbf{w}, b, \mathbf{a})$ with respect to $\mathbf{w}$ and b for a fixed value of $\mathbf{a}$. We know that the optimal $\mathbf{w}$ and b must satisfy the condition that the partial derivatives of L with regards to $\mathbf{w}$ and b are 0.

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b, \mathbf{a}) = \mathbf{w} - \sum_{i=1}^{n} a_i y_i \mathbf{x}_i = \mathbf{0} \Rightarrow \tag{3.13}$$

$$\mathbf{w} = \sum_{i=1}^{n} a_i y_i \mathbf{x}_i \tag{3.14}$$

Similarly,

$$\frac{\partial L(\mathbf{w}, b, \mathbf{a})}{\partial b} = \sum_{i=1}^{n} a_i y_i = 0. \tag{3.15}$$

Therefore, for a fixed value of $\mathbf{a}$, we have a closed form solution for $\mathbf{w}$ that minimises $L(\mathbf{w}, b, \mathbf{a})$. We also have a condition on the sum of $a_i y_i$. We can plug them back into the dual expression.

$$L(\mathbf{a}) = \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j. \tag{3.16}$$

Finally, we are left with a function of $\mathbf{a}$ what we wish to maximise. Putting this together with the constraints $a_i \geq 0$ and the constraint $\sum_{i=1}^{n} a_i y_i = 0$, we obtain the following optimisation problem

$$\begin{array}{rl} \max\limits_{\mathbf{a}} & \mathbf{1}^T\mathbf{a} - \frac{1}{2}\mathbf{a}^T\mathbf{K}_y\mathbf{a} \\ \text{subject to} & a_i \geq 0, i = 1, \dots, n \\ & \mathbf{a}^T\mathbf{y} = 0 \end{array} \tag{3.17}$$

where $\mathbf{a} = [a_1, \dots, a_n]^T$, $\mathbf{1} = [1, \dots, 1]^T$, $\mathbf{y} = [y_1, \dots, y_n]^T$ and $\mathbf{K}_y = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$.
How do I solve this optimisation problem?

Example
In this example we will see how we can solve the optimisation problem using `quadprog` of Matlab. The `quadprog` function solves generic quadratic programming optimisation problems of the form:

$$\begin{array}{rc} \min\limits_{\mathbf{g}} & \mathbf{f}^T\mathbf{g} + \frac{1}{2}\mathbf{g}^T\mathbf{H}\mathbf{g} \\ \text{subject to} & \mathbf{A}\mathbf{g} \leq \mathbf{c}, \mathbf{A}_e\mathbf{g} = \mathbf{c}_e, \mathbf{g}_l \leq \mathbf{g} \leq \mathbf{g}_u \end{array} \tag{3.18}$$

This minimisation problem solves for the vector $\mathbf{g}$. The first step to solving our problem, is to encode it using the matrices $\mathbf{H}$, $\mathbf{A}$, $\mathbf{f}$, $\mathbf{c}$, $\mathbf{c}_e$, $\mathbf{g}_l$, $\mathbf{g}_u$ and $\mathbf{A}_e$. Assume we are given a set of data stored as columns in a data matrix $\mathbf{X} \in \mathbb{R}^{F \times n}$ and a vector $\mathbf{y}$ of labels $1, -1$. Then the SVM optimisation problem (3.17) can be reformulated to (3.18) by (a) changing maximisation to minimisation by reversing the sign of the cost function, (b) setting $\mathbf{g} = \mathbf{a}$, $\mathbf{H} = [y_i y_j \mathbf{x}_i^T \mathbf{x}_j]$, (c) $\mathbf{f} = -\mathbf{1}_n$, $\mathbf{A} = \mathbf{0}$ and $\mathbf{c} = \mathbf{0}$ (a dummy inequality constraint), $\mathbf{A}_e = [y_1, \dots, y_n]$ and $c_e = 0$, $\mathbf{g}_l = [0, \dots, 0]^T$, and finally $\mathbf{g}_u = [\infty \dots, \infty]^T$. Once we have created the matrices and vectors `quadprog` function can be used like so:

```
g = quadprog (H, f, A, c, A_e, c_e, g_l, g_u)
```

which will return the optimal values into vector g.
Assume we are given a set of data stored as columns in a data matrix $\mathbf{X} \in \Re^{F \times n}$ and a vector $\mathbf{y}$ of labels $1, -1$.

```
X1 = 4+randn(10,100);
X2 = randn(10,100);
X  = [X1 X2];
y1 = ones(1,100);
y2 = -ones(1,100);
y  = [y1 y2]';
f  = -ones(1,200);
A  = zeros(1,200);
H   = (y*y').*(X'*X);
c   = 0;
```

```
A_e = y';
c_e = 0;
g_l = zeros(200,1);
g_u = 100000*ones(200,1);
alpha = quadprog(H,f,A,c,A_e,c_e,g_l,g_u);
```

The claim is that the dual problem is more computationally convenient. We validate this claim by considering the KKT conditions, which must hold for the solution. In particular, the complementary slackness condition can be written as

$$\begin{aligned} a_i = 0 &\Rightarrow y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \\ a_i > 0 &\Rightarrow y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1. \end{aligned} \tag{3.19}$$

Furthermore, from the above conditions we can find b (from any support vector). A more numerical stable solution can be found by averaging over all support vectors as

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{\mathbf{x}_i \in \mathcal{S}} (y_i - \mathbf{w}^T\mathbf{x}_i) \tag{3.20}$$

where $\mathcal{S}$ is the set of support vectors and $N_{\mathcal{S}}$ its corresponding cardinality.
These conditions mathematically validate our original sparseness intuition. Points that lie beyond the margin will have $a_i = 0$, and so will not effect the final solution $\mathbf{w} = \sum_{i=1}^{n} a_i y_i \mathbf{x}_i$. The final set of points with non-zero a_i, or alternatively, the set of points with margin 1, are called the support vectors.

3.1.2 Mapping Data to Higher Dimensional Spaces

However, problems in practice may not be linearly separable (an example is provided in Figure 3.5). That is, there is no $(\mathbf{w}, b)$ which satisfies the constraints of 3.1. In this situation, we say 3.1 is "infeasible". We can introduce slack variables $\xi_i, i = 1, \dots, n$ in the constraints

$$\begin{aligned} \min_{w,b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \; i = 1, \dots, n \end{aligned} \tag{3.21}$$

That is, the constraints in 3.21 allow training data to not be on the correct side of the separating hyperplane $\mathbf{w}^{\mathsf{T}}\mathbf{x} + b = 0$. This happens when $\xi_i > 1$ and an example is provided in Figure 3.5.
We have $\xi_i \geq 0$ since if $\xi_i < 0$, we have $y_i(\mathbf{w}^{\mathsf{T}}x_i + b) \geq 1 - \xi_i \geq 1$ and the training data is already on the correct side. The new problem is always feasible since for any $(\mathbf{w}, b)$,

$$\xi_i \equiv \max(0, 1 - y_i(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i + b)), \; i = 1, \dots, l$$

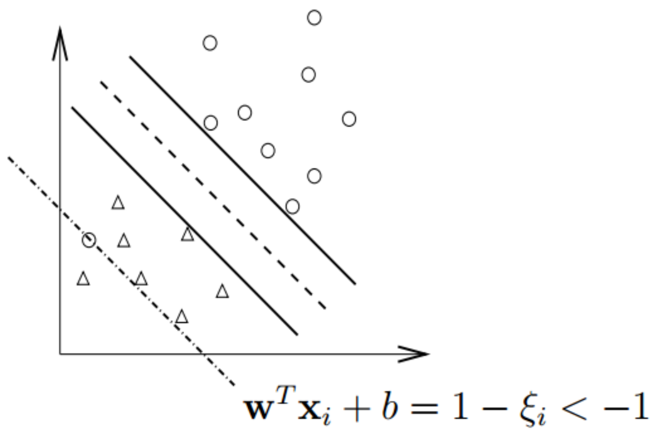


Figure 3.5: Allowing errors

we have a feasible solution $((\mathbf{w}, b, \boldsymbol{\xi}))$. Using this setting, we may worry that for linearly separable data, some ξ_i's could be larger than 1 and hence corresponding data could be wrongly classified. For the case that most data except some noisy ones are separable by a linear function, we would like $\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$ to correctly classify the majority of the points. Therefore, in the objective function we add a penalty term $C\sum_{i=1}^{l}\xi_i$, where $C > 0$ is the penalty parameter. To have the objective value as small as possible, most ξ_i's should be zero, so that the constraint goes back to its original form.
In order to formulate the dual of (3.21) we need to compute

$$L(\mathbf{w}, b, \xi_i, a_i, r_i) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n} a_i(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^{n} r_i\xi_i \tag{3.22}$$

with Lagrangian multipliers $a_i \geq 0$, $r_i \geq 0$. Computing the derivatives

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{n} a_i y_i \mathbf{x}_i = 0 \\ \frac{\partial L}{\partial b} &= \sum_{i=1}^{n} a_i y_i = 0 \\ \frac{\partial L}{\partial \xi_i} &= C - a_i - r_i = 0. \end{aligned} \tag{3.23}$$

Substituting (3.23) back to (3.22) we get the dual optimisation problem

$$\max_{\mathbf{a}} L(\mathbf{a}) = \mathbf{a}^T\mathbf{1} - \frac{1}{2}\mathbf{a}^T\mathbf{K}_y\mathbf{a} \tag{3.24}$$

$$\text{subject to } \mathbf{a}^T\mathbf{y} = 0,\ \ 0 \leq a_i \leq C \tag{3.25}$$

where $\mathbf{K}_y = [y_i y_j \mathbf{x}_i^T \mathbf{x}_j]$.
If data are distributed in a highly non-linear way, employing only a linear function causes many training instances to be on the wrong side of the hyperplane. As a

result, under-fitting occurs and the decision function does not perform well. To fit the training data better, we may think of using a non-linear curve. The problem is that it is very difficult to model non-linear curves. All we are familiar with are elliptic, hyperbolic, or parabolic curves, which are far from enough in practice. Instead of using more sophisticated curves, another approach is to map data into a higher dimensional space. In this higher dimensional space, it is more likely that data can be linearly separated. An example by mapping $\mathbf{x}$ from $\mathbb{R}^3$ to $\mathbb{R}^8$ is as follows

$$\phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, x_1^2, x_2^2 x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_2x_3, \sqrt{2}x_1x_3]$$

An extreme example is to map a data instance $\mathbf{x}$ to an infinite dimensional space

$$\phi(\mathbf{x}) = [1, \frac{x_1}{1!}, \frac{x_2^2}{2!}, \frac{x_3^3}{3!}, \dots]^\mathsf{T}$$

We then try to find a linear separating plane in a higher dimensional space so that 3.21 becomes

$$\begin{aligned}
\min_{w,b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{i=1}^{n}\xi_i \\
\text{subject to} \quad & y_i(\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \\
& \xi_i \geq 0 \; i = 1, \dots, n
\end{aligned} \tag{3.26}$$

3.1.3 The Dual Problem

The remaining problem is how to effectively solve 3.26. Especially after data are mapped into a higher dimensional space, the number of variables $(\mathbf{w}, b)$ becomes very large or even infinite. We handle this difficulty by solving the dual problem of 3.26

$$\begin{aligned}
\min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j y_i y_j \phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{x}_j) - \sum_{i=1}^{n} a_i \\
\text{subject to} \quad & 0 \leq a_i \leq C \; i = 1, \dots, n \\
& \sum_{i=1}^{n} y_i a_i = 0
\end{aligned} \tag{3.27}$$

This new problem of course has some relation with the original problem 3.26, and we hope that it can be solved more easily. We may write 3.27 in a matrix form for convenience:

$$\begin{aligned}
\min_{\boldsymbol{a}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\mathsf{T}\mathbf{K}_y\boldsymbol{\alpha} - \mathbf{1}^\mathsf{T}\boldsymbol{\alpha} \\
\text{subject to} \quad & 0 \leq a_i \leq C \; i = 1, \dots, l \\
& \mathbf{y}^\mathsf{T}\boldsymbol{\alpha} = 0
\end{aligned} \tag{3.28}$$

In 3.28, $\mathbf{1}$ is the vector of ones, C is the upper bound, $\mathbf{K}_y$ is an $n \times n$ positive semi-definite matrix, $\mathbf{K}_y \equiv [y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)]$, and $k(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{x}_j)$ is the kernel.

Therefore, the crucial point is whether the dual is easier to be solved than the primal. The number of variables in the dual is the size of the training set is n; a fixed number. In contrast, the number of variables in the primal problem varies depending on how data are mapped to a higher dimensional space. Therefore, moving from the primal to the dual means that we solve a finite-dimensional optimisation problem instead of a possibly infinite-dimensional one.

If $\phi(\mathbf{x})$ is an infinitely-long vector, there is no way to fully write it down and then calculate the inner product. Therefore, even though the dual possesses the advantage of having a finite number of variables, we could not even write the problem down before solving it. This is resolved by using special mapping functions ϕ so that $\phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{x}_j)$ is efficiently calculated (i.e., by using the kernel trick). Then, a decision function is written as

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^\mathsf{T}\phi(\mathbf{x}) + b) = \text{sign}\left(\sum_{i=1}^{l} y_i\alpha_i\phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{x}) + b\right) \tag{3.29}$$

In other words, for a test vector $\mathbf{x}$, if $\sum_{i=1}^{n} y_i\alpha_i\phi(\mathbf{x})^\mathsf{T}\phi(\mathbf{x}) + b > 0$, we classify it to be in the class 1. Otherwise, we classify it in the second class. We can see that only support vectors will affect the results in the prediction stage. In general, the number of support vectors is not large. Therefore, we can say SVM is used in order to derive important data (support vectors) from the training data.

3.2 Support Vector Regression

3.2.1 Linear Regression

Given training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ in Figure 3.6, where $\mathbf{x}_i$ is an input vector and y_i is the associated output value for $\mathbf{x}_i$, the traditional linear regression finds a linear function $\mathbf{w}^\mathsf{T}\mathbf{x} + b$ so that $(\mathbf{w}, b)$ is an optimal solution of

$$\min_{w,b} \quad \sum_{i=1}^{n}(y_i - (\mathbf{w}^\mathsf{T}\mathbf{x}_i + b))^2 \tag{3.30}$$

In other words, $\mathbf{w}^\mathsf{T}\mathbf{x} + b$ approximates training data by minimising the sum of square errors.

Note that F, the number of features, is in general less than n. Otherwise, a line passing through all points so that 3.30 is zero is the optimal function $\mathbf{w}^\mathsf{T}\mathbf{x} + b$. For such cases, over-fitting occurs.

Similar to classification, if the data is non-linearly distributed, a linear function is not good enough. Therefore, we also map data to a higher dimensional space by a function $\phi(\mathbf{x})$. Then $F \leq$ dimensionality of $\phi(\mathbf{x})$ and as a result over-fitting happens again. An example is in Figure 3.7.

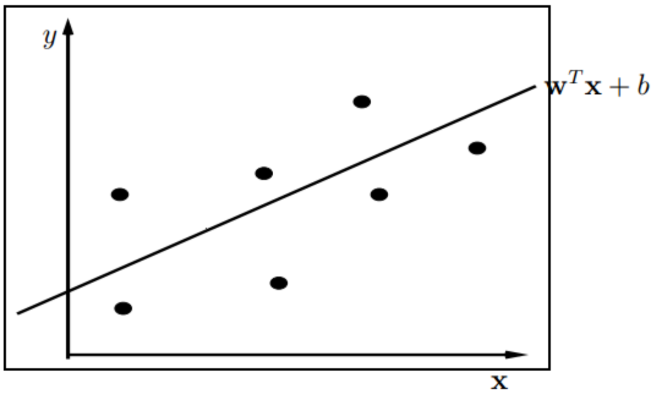


Figure 3.6: Linear Regression

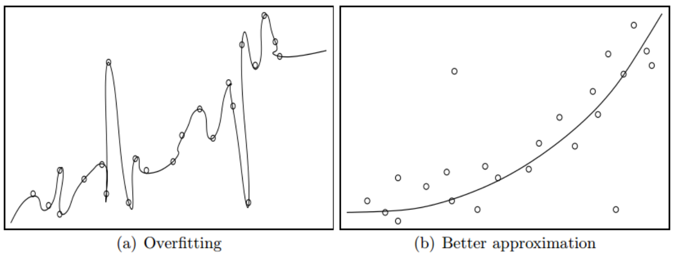


Figure 3.7: Non-linear Regression

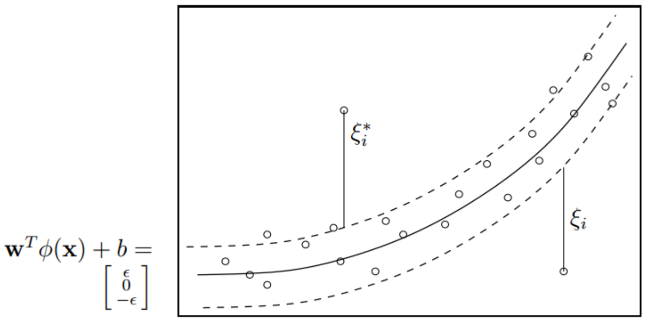

ξ_i^*
ξ_i
$\mathbf{w}^T\phi(\mathbf{x})+b=\begin{bmatrix}\epsilon\\0\\-\epsilon\end{bmatrix}$

Figure 3.8: Support Vector Regression

3.2.2 Support Vector Regression

To rectify the over-fitting problem after using ϕ, we consider the following reformulation of 3.30 (geometric interpretation is given in Figure 3.8):

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*} \quad & \sum_{i=1}^{n} \xi_i^2 + (\xi_i^*)^2 \\ \text{subject to} \quad & -\xi_i^* \le y_i - (\mathbf{w}^{\mathsf{T}}\phi(\mathbf{x}_i)+b) \le +\xi_i \\ & \xi_i, \xi_i^* \ge 0 \; i = 1,\dots,n \end{aligned} \tag{3.31}$$

It is easy to see that 3.30 (with $\mathbf{x}$ replaced by $\phi(\mathbf{x})$) and 3.31 are equivalent: If

$(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*)$ is optimal for 3.31, as $\xi_i^2 + (\xi_i^*)^2$ is minimised, we have

$$\xi_i = \max(y_i - (\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b), 0) \text{ and}$$
$$\xi_i^* = \max(-y_i + -(\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b), 0).$$

Therefore,

$$\xi_i^2 + (\xi_i^*)^2 = (y_i - (\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b))^2$$

Moreover, $\xi_i\xi_i^* = 0$ at an optimal solution.
Instead of using square errors, we can use linear ones:

$$\begin{aligned} \min_{\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*} \quad & \sum_{i=1}^{l} (\xi_i + \xi_i^*) \\ \text{subject to} \quad & -\xi_i^* \le y_i - (\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b) \le +\xi_i \\ & \xi_i, \xi_i^* \ge 0 \; i = 1, \dots, l \end{aligned}$$

Support vector regression (SVR) then employees two modifications to avoid over-fitting:

1. A threshold ϵ is given so that if the i-th datum satisfies

$$-\epsilon \le y_i - (\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b) \le \epsilon \tag{3.32}$$

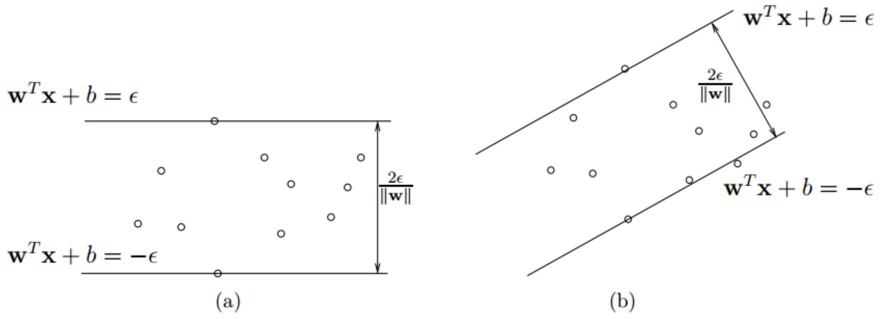
   it is considered a correct approximation. Then $\xi_i = \xi_i^* = 0$

2. To smooth the function $\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b$, an additional term $\mathbf{w}^\mathsf{T}\mathbf{w}$ is added to the objective function.

Thus, support vector regression solves the following optimisation problem:

$$\begin{aligned} \min_{w, b, \boldsymbol{\xi}, \boldsymbol{\xi}^*} \quad & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{i=1}^{l} (\xi_i + \xi_i^*) \\ \text{subject to} \quad & (\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b) - y_i \le \epsilon + \xi_i && (3.33) \\ & y_i - (\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b) \le \epsilon + \xi_i^* && (3.34) \\ & \xi_i, \xi_i^* \ge 0 \; i = 1, \dots, l \end{aligned}$$

Clearly, ξ_i is the upper training error (ξ_i^* is the lower) subject to the ϵ-insensitive tube $|y_i - (\mathbf{w}^\mathsf{T}\phi(\mathbf{x}_i) + b)| \le \epsilon$. This can be seen from Figure 3.8. If x_i is not in the tube, there is an error ξ_i or ξ_i^*, which we would like to minimise in the objective function. SVR avoids under-fitting and over-fitting the training data by minimising the training error $C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$ as well as the regularisation term $\frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w}$. Addition of the term $\mathbf{w}^\mathsf{T}\mathbf{w}$ can be explained by a similar way to that for classification problems. In Figure 3.9, under the condition that training data are in the ϵ-insensitive tube, we would like the approximate function to be as general as possible to represent the data distribution.



$\mathbf{w}^T\mathbf{x}+b=\epsilon$
$\mathbf{w}^T\mathbf{x}+b=-\epsilon$
$\frac{2\epsilon}{\|\mathbf{w}\|}$
(a)
$\mathbf{w}^T\mathbf{x}+b=\epsilon$
$\frac{2\epsilon}{\|\mathbf{w}\|}$
$\mathbf{w}^T\mathbf{x}+b=-\epsilon$
(b)

Figure 3.9: More general approximate function by maximising the distance between $\mathbf{w}^T\mathbf{x}+b=\pm\epsilon$

The parameters which control the regression quality are the cost of error C, the width of the tube ϵ, and the mapping function ϕ. Similar to support vector classification, as $\mathbf{w}$ may be a huge vector variable, we solve the dual problem

$$\min_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \quad \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^\mathsf{T}\mathbf{K}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)+\epsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*)+\sum_{i=1}^{n}y_i(\alpha_i-\alpha_i^*)$$

$$\text{subject to} \quad \sum_{i=1}^{n}(\alpha_i-\alpha_i^*)=0, 0\le a_i,a_i^*\le C,\ i=1,\ldots,n \tag{3.35}$$

$$\tag{3.36}$$

where $K_{ij}=k(\mathbf{x}_i,\mathbf{x}_j)\equiv\phi(\mathbf{x}_i)^\mathsf{T}\phi(\mathbf{x}_j)$. The derivation of the dual uses the same procedure for support vector classification. The primal-dual relation shows that

$$\mathbf{w}=\sum_{i=1}^{n}(-\alpha_i+\alpha_i^*)\phi(\mathbf{x}_i),$$

so the approximate function is:

$$\sum_{i=1}^{n}(-\alpha_i+\alpha_i^*)k(\mathbf{x}_i,\mathbf{x}_j)+b$$

Appendix A

A.1 Preliminaries on Vectors and Matrices

Below are some ”soft” definitions of vectors and matrices. We revise various representations of matrices and vectors, as well as useful definitions and identities.

A.1.1 Vectors and Vector Operators

A column vector $\mathbf{x} \in \mathbb{R}^n$ is defined as

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = [x_1 \dots x_n]^T = [x_i] \tag{A.1}$$

where $[x_1 \dots x_n]$ represents a row vector. In these notes we use column vectors. Use of row vectors will be explicitly noted.
A matrix $\mathbf{A} \in \mathbb{R}^{n \times l}$ is defined as the following collection

$$\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1l} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nl} \end{bmatrix} = [\mathbf{a}_1 \dots \mathbf{a}_l] = \begin{bmatrix} \tilde{\mathbf{a}}_1^T \\ \vdots \\ \tilde{\mathbf{a}}_n^T \end{bmatrix} = [a_{ij}] \tag{A.2}$$

where $\tilde{\mathbf{a}}_j$, $j = 1, \dots, n$ are row vectors.
The inner product between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ is a scalar c defined as

$$c = \mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i. \tag{A.3}$$

The squared ℓ_2 norm of a vector can be defined using the inner product as

$$||\mathbf{x}||_2^2 = \mathbf{x}^T\mathbf{x} = \sum_{i=1}^{n} x_i^2. \tag{A.4}$$

The cosine of the angle θ between two vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\cos(\theta(\mathbf{x}, \mathbf{y})) = \frac{\mathbf{x}^T\mathbf{y}}{||\mathbf{x}||_2||\mathbf{y}||_2}. \tag{A.5}$$

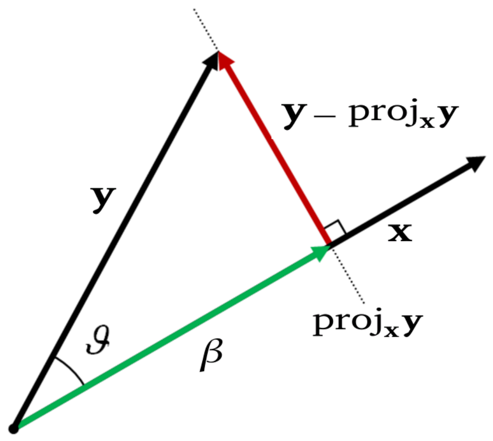

$\mathbf{y} - \mathrm{proj}_{\mathbf{x}}\mathbf{y}$
$\mathbf{y}$
$\mathbf{x}$
ϑ
β
$\mathrm{proj}_{\mathbf{x}}\mathbf{y}$

Figure A.1: Geometric interpretation of the projection of a vector $\mathbf{y}$ onto $\mathbf{x}$. The green vector is $\mathrm{proj}_{\mathbf{x}}\mathbf{y}$, while the red vector is $\mathbf{y} - \mathrm{proj}_{\mathbf{x}}\mathbf{y}$.

The cosine between two vectors can be used in order to define projections onto vectors. In particular, the projection of $\mathbf{y}$ onto $\mathbf{x}$, denoted as $\mathrm{proj}_{\mathbf{x}}\mathbf{y}$, is a vector that is co-linear to $\mathbf{x}$ and can be computed as

$$\mathrm{proj}_{\mathbf{x}}\mathbf{y} = \beta\mathbf{x} = \cos(\theta)||\mathbf{y}||_2\mathbf{x} = \frac{\mathbf{x}^T\mathbf{y}}{||\mathbf{x}||_2}\mathbf{x}. \tag{A.6}$$

The outer product of two vectors is the rank-one matrix defined as

$$\mathbf{x}\mathbf{y}^T = \begin{bmatrix} x_1y_1 & \dots & x_1y_n \\ \vdots & \ddots & \vdots \\ x_ny_1 & \dots & x_ny_n \end{bmatrix}. \tag{A.7}$$

A.1.2 Matrices and Matrix Operators

Let two matrices $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n\times l}$ and $\mathbf{B} = [b_{ij}] \in \mathbb{R}^{l\times m}$. These matrices can be represented as $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_l]$ and $\mathbf{B} = [\mathbf{b}_1 \dots \mathbf{b}_m]$, using column vectors, and as

$$\mathbf{A} = \begin{bmatrix} \tilde{\mathbf{a}}_1^T \\ \vdots \\ \tilde{\mathbf{a}}_n^T \end{bmatrix} = [a_{ij}]$$

and

$$\mathbf{B} = \begin{bmatrix} \tilde{\mathbf{b}}_1^T \\ \vdots \\ \tilde{\mathbf{b}}_l^T \end{bmatrix} = [b_{ij}],$$

using row vectors.

A.1.2.1 Matrix Norms

The most frequently used matrix norms are the Frobenius norm

$$||\mathbf{A}||_F = \sqrt{\sum_i \sum_j |a_{ij}^2|} \tag{A.8}$$

and the induced p-norms

$$||\mathbf{A}||_p = \sup_{\mathbf{x} \neq \mathbf{0}} \frac{||\mathbf{A}\mathbf{x}||_p}{||\mathbf{x}||_p} = \max_{||\mathbf{x}||_p = 1} ||\mathbf{A}\mathbf{x}||_p. \tag{A.9}$$

Important properties of the matrix norms include

- For all $\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{n \times q}$ it holds
$$||\mathbf{AB}||_p \leq ||\mathbf{A}||_p ||\mathbf{B}||_p. \tag{A.10}$$
- For all $\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{x} \in \mathbb{R}^n$ it holds
$$||\mathbf{A}\mathbf{x}||_p \leq ||\mathbf{A}||_p ||\mathbf{x}||_p. \tag{A.11}$$
- For all $\mathbf{A} \in \mathbb{R}^{m \times n}$, we can compute the induced norm for $p = 1$ as
$$||\mathbf{A}||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|. \tag{A.12}$$
- For all $\mathbf{A} \in \mathbb{R}^{m \times n}$, we can compute the induced norm for $p = \infty$ as
$$||\mathbf{A}||_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|. \tag{A.13}$$
- For all $\mathbf{A} \in \mathbb{R}^{m \times n}$, we can compute the induced norm for $p = 2$ as
$$||\mathbf{A}||_2 = \sigma_1, \tag{A.14}$$
i.e. the largest singular value of $\mathbf{A}$.

A.1.2.2 Matrix Multiplications

The matrix multiplication between $\mathbf{A} \in \mathbb{R}^{n \times l}$ and $\mathbf{B} \in \mathbb{R}^{l \times m}$ (the number of rows of $\mathbf{A}$ must be equal to the number of columns of $\mathbf{B}$) can be defined using the following forms

$$\begin{aligned} \mathbf{AB} &= \left[\sum_{k=1}^{l} a_{ik} b_{kj} \right] = \begin{bmatrix} \tilde{\mathbf{a}}_1^T \\ \vdots \\ \tilde{\mathbf{a}}_n^T \end{bmatrix} [\mathbf{b}_1 \dots \mathbf{b}_m] = [\tilde{\mathbf{a}}_i^T \mathbf{b}_j] \\ &= [\mathbf{A}\mathbf{b}_1 \dots \mathbf{A}\mathbf{b}_m] = \begin{bmatrix} \tilde{\mathbf{a}}_1^T \mathbf{B} \\ \vdots \\ \tilde{\mathbf{a}}_n^T \mathbf{B} \end{bmatrix} \\ &= \sum_{k=1}^{l} \mathbf{a}_k \tilde{\mathbf{b}}_k^T. \end{aligned} \tag{A.15}$$

Furthermore, the i, j element of $\mathbf{AB}$ can be expressed as

$$[\mathbf{AB}]_{ij} = [\sum_{k=1}^{l} \mathbf{a}_k \tilde{\mathbf{b}}_k^T]_{ij} = \sum_{k=1}^{l} [\mathbf{a}_k \tilde{\mathbf{b}}_k^T]_{ij} = \sum_{k=1}^{l} a_{ik} b_{kj}. \tag{A.16}$$

Using the above a special case is the matrix-vector multiplication

$$\mathbf{Ab} = \begin{bmatrix} \tilde{\mathbf{a}}_1^T \\ \vdots \\ \tilde{\mathbf{a}}_n^T \end{bmatrix} \mathbf{b} = \begin{bmatrix} \tilde{\mathbf{a}}_1^T \mathbf{b} \\ \vdots \\ \tilde{\mathbf{a}}_n^T \mathbf{b} \end{bmatrix} = [\tilde{\mathbf{a}}_j^T \mathbf{b}]. \tag{A.17}$$

Assume that we are given a basis $\{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$ and an arbitrary vector $\mathbf{x}$ which can be written as a linear combination of the basis as

$$\mathbf{x} = \sum_{i=1}^{n} k_i \mathbf{u}_i = \mathbf{Uk} \tag{A.18}$$

where $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_n]$ and $\mathbf{k} = [k_1, \ldots, k_n]$.
The identity matrix is defined as identity element of matrix multiplication (i.e., $\mathbf{AI} = \mathbf{IA} = \mathbf{A}$). An example of identity matrix for 3×3 matrices is

$$\mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ 3} \times \text{3 Identity Matrix.} \tag{A.19}$$

Using matrix multiplications we can define matrix integer power as

$$\mathbf{A}^k = \mathbf{AA} \cdots \mathbf{A}, k \text{ times.} \tag{A.20}$$

Fractional power of a matrix $\mathbf{A}$ can be defined a matrix $\mathbf{B} = \mathbf{A}^{\frac{1}{k}}$ such that

$$\mathbf{B}^k = \mathbf{A}. \tag{A.21}$$

A.1.2.3 Matrix Transposition

Matrix transposition operation is defined as $\mathbf{A}^T = \begin{bmatrix} a_{11} & \ldots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1l} & \ldots & a_{nl} \end{bmatrix} = [\tilde{\mathbf{a}}_1 \ldots \tilde{\mathbf{a}}_n] = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_l^T \end{bmatrix}$ i.e. rows become columns and columns rows. Important property is that

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T. \tag{A.22}$$

Important expansion of the matrix multiplication $\mathbf{AA}^T$ is the following

$$\mathbf{AA}^T = \sum_{k=1}^{l} \mathbf{a}_k \mathbf{a}_k^T. \tag{A.23}$$

Using the above the quadratic form $\mathbf{x}^T\mathbf{A}\mathbf{A}^T\mathbf{x}$ can be expanded as

$$\mathbf{x}^T\mathbf{A}\mathbf{A}^T\mathbf{x} = \sum_{k=1}^{l} \mathbf{x}^T\mathbf{a}_k\mathbf{a}_k^T\mathbf{x} = \sum_{k=1}^{l}\sum_{j=1}^{n}\sum_{i=1}^{n} x_j x_i a_{jk} a_{ik} = \sum_{k=1}^{l} (\mathbf{a}_k^T\mathbf{x})^2. \tag{A.24}$$

The matrix product $\mathbf{ABC}^T$

$$[\mathbf{ABC}^T]_{il} = \sum_j a_{ij}[\mathbf{BC}^T]_{jl} = \sum_j a_{ij}\sum_k b_{jk}c_{lk} = \sum_j\sum_k a_{ij}b_{jk}c_{lk}. \tag{A.25}$$

Example (Derivatives of Quadratic Forms)
Let function $f(\mathbf{X}) = \mathbf{x}^T\mathbf{B}\mathbf{x}$

$$f = \sum_i\sum_j x_i x_j b_{ij}. \tag{A.26}$$

First we need to split function f as

$$\begin{aligned} f &= \sum_i\sum_j x_i x_j b_{ij} \\ &= \sum_{i\neq k}\sum_{j\neq k} x_i x_j b_{ij} + \sum_i x_i x_k b_{ik} + \sum_{j\neq k} x_k x_j b_{kj} \\ &= \sum_{i\neq k}\sum_{j\neq k} x_i x_j b_{ij} + \sum_{i\neq k} x_i x_k b_{ik} + \sum_{j\neq k} x_k x_j b_{kj} + x_k^2 b_{kk} \end{aligned} \tag{A.27}$$

Then, we can compute

$$\begin{aligned} \frac{\partial f}{\partial x_k} &= \sum_{i\neq k} x_i b_{ik} + \sum_{j\neq k} x_j b_{kj} + 2x_k b_{kk} \\ &= \sum_i x_i b_{ik} + \sum_j x_j b_{kj} \\ &= [\mathbf{Bx}]_k + [\mathbf{B}^T\mathbf{x}]_k. \end{aligned} \tag{A.28}$$

Hence, $\nabla_{\mathbf{x}} f = \mathbf{Bx} + \mathbf{B}^T\mathbf{x}$. If $\mathbf{B}$ is symmetric then $\nabla_{\mathbf{x}} f = 2\mathbf{Bx}$.

A.1.2.4 Trace Operator

Matrix trace operation on square matrices is defined as $\text{tr}(\mathbf{A}) : \mathbf{A} \in \mathbb{R}^{n\times n} \to \mathbb{R}$

$$\text{tr}(\mathbf{A}) = \sum_{i=1}^{n} a_{ii}. \tag{A.29}$$

Some important properties of the trace operator

- $\text{tr}(\mathbf{A}+\mathbf{B}) = \text{tr}(\mathbf{A}) + \text{tr}(\mathbf{B})$.

- $\text{tr}(c\mathbf{A}) = c\text{tr}(\mathbf{A})$ for all scalars $c \in \mathbb{R}$
- $\text{tr}(\mathbf{A}) = \text{tr}(\mathbf{A}^T)$
- $\text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA})$, hence $\text{tr}(\mathbf{X}^T\mathbf{Y}) = \text{tr}(\mathbf{XY}^T)$
- $\text{tr}(\mathbf{ABCD}) = \text{tr}(\mathbf{BCDA}) = \text{tr}(\mathbf{DABC}) = \text{tr}(\mathbf{CDAB})$
- If $\mathbf{A}$ is an $n \times n$ matrix and $\lambda_1, \dots, \lambda_n$ are its corresponding eigenvalues then $\text{tr}(\mathbf{A}) = \sum_i \lambda_i$.
- $||\mathbf{A}||_F^2 = \text{tr}(\mathbf{A}^T\mathbf{A})$.

Example (Trace derivatives A: Linear Case)
Let function $f(\mathbf{X}) = \text{tr}(\mathbf{AXB})$

$$f = \sum_i [\mathbf{AXB}]_{ii} = \sum_i \sum_j a_{ij}[\mathbf{XB}]_{ji} = \sum_i \sum_j \sum_k a_{ij}x_{jk}b_{ki}. \tag{A.30}$$

Now we can compute $\nabla_{\mathbf{X}} f = [\frac{\partial f}{\partial x_{jk}}]$

$$\frac{\partial f}{\partial x_{jk}} = \sum_i a_{ij}b_{ki} = [\mathbf{BA}]_{kj} = [(\mathbf{BA})^T]_{jk}. \tag{A.31}$$

Hence, $\nabla_{\mathbf{X}}\text{tr}(\mathbf{AXB}) = \mathbf{A}^T\mathbf{B}^T$.

Example (Trace derivatives A: Quadratic Case)
Let function $f(\mathbf{W}) = \text{tr}(\mathbf{W}^T\mathbf{BW})$. Using the results of the example of derivatives with quadratic forms we get

$$\begin{aligned} f &= \sum_i \mathbf{w}_i^T\mathbf{B}\mathbf{w}_i \\ &= \sum_i \sum_j \sum_r w_{ji}w_{ri}b_{jr} \\ &= \sum_i \left(\sum_{j\neq k} \sum_{r \neq j} w_{ji}w_{ri}b_{jr} + \sum_{r\neq k} w_{ki}w_{ri}b_{kr} + \sum_{j \neq k} w_{ji}w_{ki}b_{jk} + w_{ki}^2 b_{kk} \right). \end{aligned} \tag{A.32}$$

Now we can compute $\nabla_{\mathbf{W}} f = [\frac{\partial f}{\partial w_{ki}}]$

$$\begin{aligned} \frac{\partial f}{\partial w_{ki}} &= \sum_{r\neq k} w_{ri}b_{kr} + \sum_{j\neq k} w_{ji}b_{jk} + 2w_{ki}b_{kk} \\ &= \sum_r w_{ri}b_{kr} + \sum_j w_{ji}b_{jk} \\ &= [\mathbf{BW}]_{ki} + [\mathbf{B}^T\mathbf{W}]_{ki}. \end{aligned} \tag{A.33}$$

Hence, $\nabla_{\mathbf{W}}\text{tr}(\mathbf{W}^T\mathbf{B}\mathbf{W}) = \mathbf{B}\mathbf{W} + \mathbf{B}^T\mathbf{W}$. If $\mathbf{B}$ is symmetric then $\nabla_{\mathbf{W}}\text{tr}(\mathbf{W}^T\mathbf{B}\mathbf{W}) = 2\mathbf{B}\mathbf{W}$.

A.1.2.5 Matrix Determinant

Matrix determinant is defined as (Laplace formula)

$$\text{det}(\mathbf{A}) = \sum_{j=1}^{n}(-1)^{j+k}a_{jk}|\mathbf{A}_{jk}| \tag{A.34}$$

where $\mathbf{A}_{jk}$ is defined as the determinant of the $(n-1)\times(n-1)$ matrix that is produced from $\mathbf{A}$ by removing the j-th row and k-th column.

Example (Determinant)

Assume matrix $\mathbf{A} = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 2 & 0 & -1 \end{bmatrix}$

$$\begin{aligned}\text{det}(\mathbf{A}) &= (-1)^{1+2}\cdot 2\cdot \begin{bmatrix} -1 & 3 \\ 2 & -1 \end{bmatrix} + (-1)^{2+2}\cdot 1\cdot \begin{bmatrix} -2 & -3 \\ 2 & -1 \end{bmatrix} + (-1)^{3+2}\cdot 0\cdot \begin{bmatrix} -2 & -3 \\ -1 & 3 \end{bmatrix} \\ &= (-2)\cdot((-1)\cdot(-1) - 2\cdot 3) + 1\cdot((-2)\cdot(-1) - 2\cdot(-3)) \\ &= (-2)\cdot(-5) + 8 = 18\end{aligned} \tag{A.35}$$

Some important properties of the determinant of matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$

- $\text{det}(\mathbf{I}) = 1$.
- $\text{det}(\mathbf{A}) = \text{det}(\mathbf{A}^T)$
- $\text{det}(\mathbf{A}\mathbf{B}) = \text{det}(\mathbf{A})\text{det}(\mathbf{B})$
- $\text{det}(c\mathbf{A}) = c^n\text{det}(\mathbf{A})$
- If $\mathbf{A}$ is a triangular matrix the $\text{det}(\mathbf{A}) = \prod_{i=1}^{n} a_{ii}$.
- If $\mathbf{A}$ is an $n \times n$ matrix and $\lambda_1, \dots, \lambda_n$ are its corresponding eigenvalues then $\text{det}(\mathbf{A}) = \prod_i \lambda_i$.
- $\nabla_{\mathbf{A}}\text{det}(\mathbf{A}) = \text{det}(\mathbf{A})(\mathbf{A}^{-1})^T$
- Determinant of block matrices. Let the block matrix and $\mathbf{A}$ be invertible, then $\text{det}\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} = \text{det}(\mathbf{A})\text{det}(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})$. Due to the above we have $\text{det}\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} = \text{det}(\mathbf{A})\text{det}(\mathbf{D})$.

A.1.2.6 Matrix Inverse

The invertible matrix theorem. Let $\mathbf{A}$ be a square $n \times n$ matrix over $\mathbb{R}$. The following statements are all equivalent

- $\mathbf{A}$ is invertible (or non-singular or non-degenerate)
- $\det(\mathbf{A}) \neq 0$
- $\mathbf{A}$ has full rank $\text{rank}(\mathbf{A}) = n$
- The system $\mathbf{Ax} = \mathbf{0}$ has only one trivial solution $\mathbf{x} = \mathbf{0}$.
- The null space of $\mathbf{A}$ is the empty space.
- The system $\mathbf{Ax} = \mathbf{b}$ has a unique solution for each $\mathbf{b}$.
- The mapping $\mathbf{x} \mapsto \mathbf{Ax}$ is one to one and onto (bijection).
- The columns of $\mathbf{A}$ are linearly independent.
- The columns of $\mathbf{A}$ form a basis of $\mathbb{R}^n$.
- $\mathbf{A}^T$ is invertible (hence the rows of $\mathbf{A}$ are linearly independent).
- There is a unique $n \times n$ matrix $\mathbf{A}^{-1}$ such that $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$.
- $\mathbf{A}$ does not have any zero eigenvalues.

Block matrix inversion

$$\mathbf{A} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B})^{-1} \\ -(\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B})^{-1}\mathbf{CA}^{-1} & (\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B})^{-1} \end{bmatrix} \tag{A.36}$$

where $\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B}$ is the so-called Schur complement of $\mathbf{A}$.
In linear algebra, two $n \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$ are called similar if

$$\mathbf{B} = \mathbf{P}^{-1}\mathbf{AP}. \tag{A.37}$$

Similar matrices have the same rank, determinant, trace and eigenvalues.

A.1.2.7 Matrix Pseudo-Inverse

The pseudo-inverse of an $m \times n$ matrix $\mathbf{A}$ is a matrix that generalizes to arbitrary matrices the notion of inverse of a square, invertible matrix.

- If $\mathbf{A}$ is full column rank, $\text{rank}(\mathbf{A}) = n \leq m$, that is $\mathbf{A}^T\mathbf{A}$ is not singular, then $\mathbf{A}^\dagger$ is a left inverse of $\mathbf{A}$ (i.e., $\mathbf{A}^\dagger\mathbf{A} = \mathbf{I}$). We have the closed-form expression
$$\mathbf{A}^\dagger = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T. \tag{A.38}$$
- If $\mathbf{A}$ is full row rank, $\text{rank}(\mathbf{A}) = m \leq n$, that is $\mathbf{AA}^T$ is not singular, then $\mathbf{A}^\dagger$ is a right inverse of $\mathbf{A}$ (i.e., $\mathbf{AA}^\dagger = \mathbf{I}$). We have the closed-form expression
$$\mathbf{A}^\dagger = \mathbf{A}^T(\mathbf{AA}^T)^{-1}. \tag{A.39}$$
- If $\mathbf{A}$ is square and invertible matrix then $\mathbf{A}^\dagger = \mathbf{A}^{-1}$.

A.1.2.8 Range, Null Space and Rank of a matrix

There are two important sub-spaces associated with a matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$. The range of $\mathbf{A}$ is defined by

$$\text{ran}(\mathbf{A}) = \{\mathbf{y} \in \mathbb{R}^n : \mathbf{y} = \mathbf{A}\mathbf{x}, \mathbb{R}^m\} \tag{A.40}$$

and the null space of $\mathbf{A}$ is defined by

$$\text{null}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^m : \mathbf{A}\mathbf{x} = \mathbf{0}\}. \tag{A.41}$$

If $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n]]$ is a column partitioning, then[1]

$$\text{ran}(\mathbf{A}) = \text{span}\{\mathbf{a}_1, \ldots, \mathbf{a}_m\}. \tag{A.42}$$

The column rank of $\mathbf{A}$ is the dimension of the column space of $\mathbf{A}$,while the row rank of $\mathbf{A}$ is the dimension of the row space [2] of $\mathbf{A}$. A fundamental result in linear algebra is that the column rank and the row rank are always equal.
Hence, column or row rank (e.g., the number of linear independent columns, $\text{rank}(\mathbf{A}) = \dim(\text{ran}(\mathbf{A}))$) is simply called rank of matrix $\mathbf{A}$.
A square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is said to have full rank if $\text{rank}(\mathbf{A}) = n$. A matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$ has full rank if $\text{rank}(\mathbf{A}) = \min\{n, m\}$. If a matrix does not have full rank, then it is called rank deficient.

- For all matrices $\mathbf{A} \in \mathbb{R}^{n\times m}$ we have that $\text{rank}(\mathbf{A}) \leq \min\{n, m\}$
- Rank is the dimension of the largest square sub-matrix of a matrix that has a non-zero determinant.
- $\text{rank}(\mathbf{A}) = 0$ iff $\mathbf{A} = \mathbf{0}$.
- If $\mathbf{B} \in \mathbb{R}^{m\times k}$, then

$$\text{rank}(\mathbf{A}\mathbf{B}) \leq \min(\text{rank}(\mathbf{A}), \text{rank}(\mathbf{B})). \tag{A.43}$$

  If $\text{rank}(\mathbf{B})$ is of rank m, then $\text{rank}(\mathbf{A}\mathbf{B}) = \text{rank}(\mathbf{A})$.
- If $\mathbf{C} \in \mathbb{R}^{l\times n}$ of rank n, then

$$\text{rank}(\mathbf{C}\mathbf{A}) = \text{rank}(\mathbf{A}). \tag{A.44}$$

- The rank of $\mathbf{A}$ is equal to r if and only if there exists an invertible $n \times n$ matrix $\mathbf{X}$ and an invertible $m \times m$ matrix $\mathbf{Y}$ such that

$$\mathbf{X}\mathbf{A}\mathbf{Y} = \begin{bmatrix} \mathbf{I}_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{A.45}$$

  where $\mathbf{I}_r$ denotes the $r \times r$ identity matrix.
-

$$\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}^T) = \text{rank}(\mathbf{A}\mathbf{A}^T) = \text{rank}(\mathbf{A}^T\mathbf{A}). \tag{A.46}$$

- Rank-nullity theorem. The rank and the nullity[3] of a matrix equals the number of columns of the matrix.

[1]Column space, also referred to as the range of a matrix, is the span (set of all possible linear combinations) of its column vectors
[2]Row space is the span (set of all possible linear combinations) of its row vectors
[3]Nullity is the dimension of the null space of the matrix.

A.1.2.9 Eigenvalues and Eigenvectors

The determination of the eigenvalues and eigenvectors of a system is extremely important in machine learning. Each eigenvalue is paired with a corresponding so-called eigenvector.
Let $\mathbf{A}$ be a square $n \times n$ matrix. Vector $\mathbf{x}$ is a right eigenvector of matrix $\mathbf{A}$ with a corresponding eigenvalue λ if

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}. \tag{A.47}$$

The above equation can be stated equivalently as,

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}. \tag{A.48}$$

The above equation has a non-zero solution if and only if the determinant $|\mathbf{A} - \lambda\mathbf{I}|$ is zero. Therefore, the eigenvalues of $\mathbf{A}$ are values of λ that satisfy the equation

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0. \tag{A.49}$$

Hence, a way to find eigenvalues analytically is by finding the roots of the above polynomial (which is called the characteristic polynomial of $\mathbf{A}$).
The generalised eigenvectors of matrices $\mathbf{A}$ and $\mathbf{B}$ are vectors that satisfy

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x} \tag{A.50}$$

and λ is the corresponding generalised eigenvalue. If $\mathbf{B}$ is invertible, then the original problem can be written in the form

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{A.51}$$

which is a standard eigenvalue problem. However, in most situations it is preferable not to perform the inversion, but rather to solve the generalised eigenvalue problem as stated originally.
The possible values of λ must obey the following equation

$$\det(\mathbf{A} - \lambda\mathbf{B}) = 0. \tag{A.52}$$

For an arbitrary matrix $\mathbf{A}$ its eigenvalues and eigenvectors could be complex. In machine learning, generally we will work with symmetric matrices. If $\mathbf{A}$ is a symmetric $n \times n$ matrix, then all its eigenvalues and eigenvectors are real. Furthermore, its eigenvectors form an orthonormal basis of $\mathbb{R}^n$. Hence, it admits the following eigendecomposition

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T,\ \mathbf{U}^T\mathbf{U} = \mathbf{I}_n,\ \mathbf{U}\mathbf{U}^T = \mathbf{I}_n. \tag{A.53}$$

If $\mathbf{A}$ and $\mathbf{B}$ are symmetric and $\mathbf{B}$ is a positive-definite matrix, the generalised eigenvalues λ are real and eigenvectors $\mathbf{v}_i$ and $\mathbf{v}_j$ with distinct eigenvalues are $\mathbf{B}$-orthogonal

$$\mathbf{v}_i^T\mathbf{B}\mathbf{v}_j = 0. \tag{A.54}$$

A.1.2.10 Positive and Negative Definite Matrices

A symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called positive definite if for all $\mathbf{x} \in \mathbb{R}^n$

$$\mathbf{x}^T \mathbf{A} \mathbf{x} > 0. \tag{A.55}$$

Similarly it is called negative positive if $\mathbf{x}^T \mathbf{A} \mathbf{x} < 0$ for all $\mathbf{x} \in \mathbb{R}^n$.
Furthermore, a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is positive semi-definite if the above inequality is not strict (i.e., $\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$). All matrices of the form $\mathbf{B}\mathbf{B}^T$ are positive semi-definite (the proof is in equation (A.24)).

Theorem 3
A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is positive definite iff all eigenvalues are positive.

Proof: Assume that all eigenvalues λ_i are positive. Then according to the eigendecomposition of symmetric matrices in (A.53) we have $\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$. The columns of $\mathbf{U}$ constitute a base of $\mathbb{R}^n$. Hence, $\mathbf{x} = \mathbf{U}\mathbf{c}$ for all $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{c} \neq \mathbf{0}$. Then,

$$\mathbf{x}^T \mathbf{A} \mathbf{x} = \mathbf{c}^T \mathbf{U}^T \mathbf{U} \boldsymbol{\Lambda} \mathbf{U} \mathbf{U}^T \mathbf{c} = \mathbf{c}^T \boldsymbol{\Lambda} \mathbf{c} = \sum_{i=1}^{n} c_i^2 \lambda_i > 0. \tag{A.56}$$

Hence, $\mathbf{A}$ is positive definite.
Assume now that $\mathbf{A}$ is positive definite. Then, again

$$\mathbf{x}^T \mathbf{A} \mathbf{x} = \mathbf{c}^T \boldsymbol{\Lambda} \mathbf{c} = \sum_{i=1}^{n} c_i^2 \lambda_i > 0 \tag{A.57}$$

which holds for all $\mathbf{c} \in \mathbb{R}^n$. Now, by choosing as $\mathbf{c}$ the columns of the identity $\mathbf{I}_n$ matrix, the above inequality turns into $\lambda_i > 0$. Hence, all eigenvalues are positive.

A.1.2.11 Triangular Matrices

Triangular matrices are very important matrices in linear algebra, because they allow for efficient computations. A triangular matrix is a special kind of square matrix. A square matrix is called lower triangular if all the entries above the main diagonal are zero. Similarly, a square matrix is called upper triangular if all the entries below the main diagonal are zero. A triangular matrix is one that is either lower triangular or upper triangular. A matrix that is both upper and lower triangular is called a diagonal matrix.
Properties of triangular matrices

- The sum of two upper (lower) triangular matrices is upper (lower) triangular.
- The product of two upper (lower) triangular matrices is upper (lower) triangular.
- The inverse of an invertible upper (lower) triangular matrix is upper (lower) triangular.

- The product of an upper (lower) triangular matrix by a constant is an upper (lower) triangular matrix.
- A matrix $\mathbf{A}$ which is simultaneously triangular and normal (i.e., $\mathbf{A}\mathbf{A}^T = \mathbf{A}^T\mathbf{A}$) is also diagonal.
- The transpose of an upper triangular matrix is a lower triangular matrix and vice versa.
- The determinant of a triangular matrix equals the product of the diagonal entries.
- The diagonal entries of a triangular matrix give the multiset of eigenvalues.

Proof of the above are exercises of the first tutorial.

A.1.2.12 QR decomposition

Theorem 4
If $\mathbf{A} = \mathbf{QR}$ is the QR decomposition of matrix $\mathbf{A}$ then

$$|det(\mathbf{A})| = |\prod_i r_{ii}|. \tag{A.58}$$

Proof: First we prove that $|\text{det}(\mathbf{Q})| = 1$.

$$\mathbf{Q}\mathbf{Q}^T = \mathbf{I} \Rightarrow \text{det}(\mathbf{Q})\text{det}(\mathbf{Q}^T) = 1 \Rightarrow (\text{det}(\mathbf{Q}))^2 = 1 \Rightarrow |\text{det}(\mathbf{Q})| = 1. \tag{A.59}$$

Hence, $|\text{det}(\mathbf{A})| = |\prod_i r_{ii}|$.

A.2 Inner Products

Inner products allow the rigorous introduction of intuitive geometrical quantities, such as the length of a vector and the angle or distance between two vectors. We may already be familiar with the **scalar product/dot product** in $\mathbb{R}^n$

$$\boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i \,. \tag{A.60}$$

However, inner products are more general concepts with specific properties, which we will now introduce.

Definition 13
Let V be a vector space and $\beta : V \times V \to \mathbb{R}$ a bilinear mapping (i.e., linear in both arguments).

- *β is called* **symmetric** *if $\beta(\boldsymbol{x}, \boldsymbol{y}) = \beta(\boldsymbol{y}, \boldsymbol{x})$ for all $\boldsymbol{x}, \boldsymbol{y} \in V$.*
- *β is called* **positive definite** *if for all $\boldsymbol{x} \neq \mathbf{0}$: $\beta(\boldsymbol{x}, \boldsymbol{x}) > 0$. $\beta(\mathbf{0}, \mathbf{0}) = 0$.*

- *A positive definite, symmetric bilinear mapping $\beta : V \times V \to \mathbb{R}$ is called* **inner product** *on V. We typically write $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ instead of $\beta(\boldsymbol{x}, \boldsymbol{y})$.*
- *The pair $(V, \langle \cdot, \cdot \rangle)$ is called* **inner product space, normed vector space** *or (real)* **vector space with inner product***. If we use the dot product defined in* (A.60)*, we call $(V, \langle \cdot, \cdot \rangle)$ a* **Euclidean vector space***.*

Example (Inner Product)
Consider $V = \mathbb{R}^2$. If we define $\beta(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle := x_1 y_1 - (x_1 y_2 + x_2 y_1) + 2 x_2 y_2$ then β is an inner product but different from the dot product.

In an inner product space, the inner product allows us to introduce concepts, such as lengths, distances and orthogonality.

A.2.1 Lengths, Distances, Orthogonality

Definition 14 (Norm)
Consider an inner product space $(V, \langle \cdot, \cdot \rangle)$. Then $\|\boldsymbol{x}\| := \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$ is the **length** *or* **norm** *of $\boldsymbol{x} \in V$. The mapping*

$$\|\cdot\| : V \to \mathbb{R} \tag{A.61}$$
$$\boldsymbol{x} \mapsto \|\boldsymbol{x}\| \tag{A.62}$$

is called **norm***.*

Example (Length of Vectors)
In geometry, we are often interested in lengths of vectors. We can now use the inner product to compute them. For instance, in a Euclidean vector space, where we use the dot product, if $\boldsymbol{x} = [1, 2]^\top$ then its norm/length is $\|\boldsymbol{x}\| = \sqrt{1^2 + 2^2} = \sqrt{5}$

Remark 27
The norm $\|\cdot\|$ possesses the following properties:

1. $\|\boldsymbol{x}\| \geq 0$ *for all $\boldsymbol{x} \in V$ and $\|\boldsymbol{x}\| = 0 \Leftrightarrow \boldsymbol{x} = \boldsymbol{0}$*
2. $\|\lambda \boldsymbol{x}\| = |\lambda| \cdot \|\boldsymbol{x}\|$ *for all $\boldsymbol{x} \in V$ and $\lambda \in \mathbb{R}$*
3. **Minkowski inequality:** $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ *for all $\boldsymbol{x}, \boldsymbol{y} \in V$*

Definition 15 (Distance and Metric)
Consider an inner product space $(V, \langle \cdot, \cdot \rangle)$. Then $d(\boldsymbol{x}, \boldsymbol{y}) := \|\boldsymbol{x} - \boldsymbol{y}\|$ is called **distance** *of $\boldsymbol{x}, \boldsymbol{y} \in V$. The mapping*

$$d : V \times V \to \mathbb{R} \tag{A.63}$$
$$(\boldsymbol{x}, \boldsymbol{y}) \mapsto d(\boldsymbol{x}, \boldsymbol{y}) \tag{A.64}$$

is called **metric***.*

A metric d satisfies:

1. d is positive definite, i.e., $d(\boldsymbol{x}, \boldsymbol{y}) \geq 0$ for all $\boldsymbol{x}, \boldsymbol{y} \in V$ and $d(\boldsymbol{x}, \boldsymbol{y}) = 0 \Leftrightarrow \boldsymbol{x} = \boldsymbol{y}$
2. d is symmetric, i.e., $d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x})$ for all $\boldsymbol{x}, \boldsymbol{y} \in V$.
3. **Triangular inequality:** $d(\boldsymbol{x}, \boldsymbol{z}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z})$.

Definition 16 (Orthogonality)
Vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are **orthogonal** *if $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0$, and we write $\boldsymbol{x} \perp \boldsymbol{y}$*

Theorem 5
Let $(V, \langle \cdot, \cdot \rangle)$ be an inner product space and $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in V$. Then:

1. **Cauchy-Schwarz inequality:** $|\langle \boldsymbol{x}, \boldsymbol{y} \rangle| \leq \|\boldsymbol{x}\| \, \|\boldsymbol{y}\|$
2. **Minkowski inequality:** $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$
3. **Triangular inequality:** $d(\boldsymbol{x}, \boldsymbol{z}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z})$
4. **Parallelogram law:** $\|\boldsymbol{x} + \boldsymbol{y}\| + \|\boldsymbol{x} - \boldsymbol{y}\| = 2\|\boldsymbol{x}\|^2 + 2\|\boldsymbol{y}\|^2$
5. $4\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \|\boldsymbol{x} + \boldsymbol{y}\|^2 - \|\boldsymbol{x} - \boldsymbol{y}\|^2$
6. $\boldsymbol{x} \perp \boldsymbol{y} \Leftrightarrow \|\boldsymbol{x} + \boldsymbol{y}\|^2 = \|\boldsymbol{x}\|^2 + \|\boldsymbol{y}\|^2$

The Cauchy-Schwarz inequality allows us to define angles ω in inner product spaces between two vectors $\boldsymbol{x}, \boldsymbol{y}$. Assume that $\boldsymbol{x} \neq \boldsymbol{0}, \boldsymbol{y} \neq \boldsymbol{0}$. Then

$$-1 \leq \frac{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}{\|\boldsymbol{x}\| \, \|\boldsymbol{y}\|} \leq 1 \,. \tag{A.65}$$

Therefore, there exists a unique $\omega \in [0, \pi)$ with

$$\cos \omega = \frac{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}{\|\boldsymbol{x}\| \, \|\boldsymbol{y}\|} \,. \tag{A.66}$$

The number ω is the **angle** between $\boldsymbol{x}$ and $\boldsymbol{y}$.

A.2.2 Applications of Inner Products

Inner products allow us to compute angles between vectors or distances. A major purpose of inner products is to determine whether vectors are orthogonal to each other; in this case $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0$. This plays an important role for orthogonal projections, which can be used for (linear) dimensionality reduction. The inner product also allows us to determine specific bases of vector (sub)spaces, where each vector is orthogonal to all others (orthogonal bases), e.g., using the Gram-Schmidt method. These bases are important optimization and numerical algorithms for solving linear

equation systems. For instance, Krylov subspace methods[4], such as Conjugate Gradients or GMRES, minimize residual errors that are orthogonal to each other (Stoer and Burlirsch, 2002).
In machine learning, inner products are important in the context of kernel methods (Schölkopf and Smola, 2002). Kernel methods exploit the fact that many linear algorithms can be expressed purely by inner product computations.[5] Then, the "kernel trick" allows us to compute these inner products implicitly in a (potentially infinite-dimensional) feature space, without even knowing this feature space explicitly. This allowed the "non-linearization" of many algorithms used in machine learning, such as kernel-PCA (Schölkopf et al., 1998) for dimensionality reduction. Gaussian processes (Rasmussen and Williams, 2006) also fall into the category of kernel methods and are the current state-of-the-art in probabilistic regression (fitting curves to data points).

A.3 Useful Matrix Identities

To avoid explicit inversion of a possibly singular matrix, we often employ the following three identities:

$$(\boldsymbol{A}^{-1}+\boldsymbol{B}^{-1})^{-1} = \boldsymbol{A}(\boldsymbol{A}+\boldsymbol{B})^{-1}\boldsymbol{B} = \boldsymbol{B}(\boldsymbol{A}+\boldsymbol{B})^{-1}\boldsymbol{A} \tag{A.67}$$

$$(\boldsymbol{Z}+\boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^\top)^{-1} = \boldsymbol{Z}^{-1} - \boldsymbol{Z}^{-1}\boldsymbol{U}(\boldsymbol{W}^{-1}+\boldsymbol{V}^\top\boldsymbol{Z}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}^\top\boldsymbol{Z}^{-1} \tag{A.68}$$

$$(\boldsymbol{A}+\boldsymbol{B}\boldsymbol{C})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{I}+\boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B})^{-1}\boldsymbol{C}\boldsymbol{A}^{-1}\,. \tag{A.69}$$

The **Searle identity** in (A.67) is useful if the individual inverses of $\boldsymbol{A}$ and $\boldsymbol{B}$ do not exist or if they are ill conditioned. The **Woodbury identity** in (A.68) can be used to reduce the computational burden: If $\boldsymbol{Z} \in \mathbb{R}^{p\times p}$ is diagonal, the inverse $\boldsymbol{Z}^{-1}$ can be computed in $\mathcal{O}(p)$. Consider the case where $\boldsymbol{U} \in \mathbb{R}^{p\times q}$, $\boldsymbol{W} \in \mathbb{R}^{q\times q}$, and $\boldsymbol{V}^\top \in \mathbb{R}^{q\times p}$ with $p \gg q$. The inverse $(\boldsymbol{Z}+\boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^\top)^{-1} \in \mathbb{R}^{p\times p}$ would require $\mathcal{O}(p^3)$ computations (naively implemented). Using (A.68), the computational burden reduces to $\mathcal{O}(p)$ for the inverse of the diagonal matrix $\boldsymbol{Z}$ plus $\mathcal{O}(q^3)$ for the inverse of $\boldsymbol{W}$ and the inverse of $\boldsymbol{W}^{-1}+\boldsymbol{V}^\top\boldsymbol{Z}^{-1}\boldsymbol{U} \in \mathbb{R}^{q\times q}$. Therefore, the inversion of a $p\times p$ matrix can be reduced to the inversion of $q\times q$ matrices, the inversion of a diagonal $p\times p$ matrix, and some matrix multiplications, all of which require less than $\mathcal{O}(p^3)$ computations. The **Kailath inverse** in (A.69) is a special case of the Woodbury identity in (A.68) with $\boldsymbol{W} = \boldsymbol{I}$. The Kailath inverse makes the inversion of $\boldsymbol{A}+\boldsymbol{B}\boldsymbol{C}$ numerically a bit more stable if $\boldsymbol{A}+\boldsymbol{B}\boldsymbol{C}$ is ill-conditioned and $\boldsymbol{A}^{-1}$ exists.

[4]The basis for the Krylov subspace is derived from the Cayley-Hamilton theorem, which allows us to compute the inverse of a matrix in terms of a linear combination of its powers.

[5]Matrix-vector multiplication $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ falls into this category since b_i is dot product of the ith row of $\boldsymbol{A}$ with $\boldsymbol{x}$.

Bibliography

Akaike, H. (1974). A New Look at the Statistical Model Identification. *IEEE Transactions on Automatic Control*, 19(6):716–723. pages 62

Belhumeur, P. N., Hespanha, J. P., and Kriegman, D. J. (1997). Eigenfaces vs. Fisherfaces: Recognition using Class Specific Linear Projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720. pages 2

Bertsekas, D. P. (1999). *Nonlinear Programming*. Athena Scientific. pages 56

Bickson, D., Dolev, D., Shental, O., Siegel, P. H., and Wolf, J. K. (2007). Linear Detection via Belief Propagation. In *Proceedings of the Annual Allerton Conference on Communication, Control, and Computing*. pages 23

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag. pages 1, 17, 18, 49

Bottou, L. (1998). Online Algorithms and Stochastic Approximations. In *Online Learning and Neural Networks*, pages 1–34. Cambridge University Press. pages 56

Bryson, A. E. (1961). A Gradient Method for Optimizing Multi-stage Allocation Processes. In *Proceedings of the Harvard University Symposium on Digital Computers and Their Applications*. pages 36

Burges, C. J. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167. pages 2

Chickering, D. M. and Heckerman, D. (1996). Efficient Approximations for the Marginal Likelihood of Incomplete Data Given a Bayesian Network. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. pages 63

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M. A., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012). Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems*, pages 1–11. pages 56

Deisenroth, M. P. and Mohamed, S. (2012). Expectation Propagation in Gaussian Process Dynamical Systems. In *Advances in Neural Information Processing Systems*, pages 2618–2626. pages 23

Deisenroth, M. P. and Ohlsson, H. (2011). A General Perspective on Gaussian Filtering and Smoothing: Explaining Current and Deriving New Algorithms. In *Proceedings of the American Control Conference*. pages 13

Dreyfus, S. (1962). The Numerical Solution of Variational Problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45. pages 36

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159. pages 56

Gal, Y., van der Wilk, M., and Rasmussen, C. E. (2014). Distributed Variational Inference in Sparse Gaussian Process Regression and Latent Variable Models. In *Advances in Neural Information Processing Systems*. pages 56

Goh, G. (2017). Why Momentum Really Works. *Distill*. pages 54

Golub, G. H. and Van Loan, C. F. (2012). *Matrix Computations*, volume 4. JHU Press. pages 1, 2, 73

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian Processes for Big Data. In Nicholson, A. and Smyth, P., editors, *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. AUAI Press. pages 56

Herbrich, R., Minka, T., and Graepel, T. (2007). TrueSkill(TM): A Bayesian Skill Rating System. In *Advances in Neural Information Processing Systems*, pages 569–576. MIT Press. pages 22, 23

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic Variational Inference. *Journal of Machine Learning Research*, 14(1):1303–1347. pages 56

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press. pages 5

Jefferys, W. H. and Berger, J. O. (1992). Ockham's Razor and Bayesian Analysis. *American Scientist*, 80:64–72. pages 58

Jimenez Rezende, D., Mohamed, S., and Wierstra, D. (2014). Stochastic Backpropagation and Variational Inference in Deep Latent Gaussian Models. In *Proceedings of the International Conference on Machine Learning*. pages 8, 16

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME—Journal of Basic Engineering*, 82(Series D):35–45. pages 13

Kelley, H. J. (1960). Gradient Theory of Optimal Flight Paths. *Ars Journal*, 30(10):947–954. pages 36

Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, pages 1–13. pages 56

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *Proceedings of the International Conference on Learning Representations*. pages 8, 16

Kittler, J. and Föglein, J. (1984). Contextual Classification of Multispectral Pixel Data. *IMage and Vision Computing*, 2(1):13–29. pages 22, 23

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models*. MIT Press. pages 22

Lin, C.-J. (2006). A guide to support vector machines. *Department of Computer Science & Information Engineering, National Taiwan University, Taiwan*. pages 2

MacKay, D. J. C. (1992). Bayesian Interpolation. *Neural Computation*, 4:415–447. pages 58

MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK. pages 58, 59

McEliece, R. J., MacKay, D. J. C., and Cheng, J.-F. (1998). Turbo Decoding as an Instance of Pearl's "Belief Propagation" Algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152. pages 23

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-Level Control through Deep Reinforcement Learning. *Nature*, 518:529–533. pages 56

Murphy, K. P. (2012). *Machine Learning: A Proabilistic Perspective*. MIT Press, Cambridge, MA, USA. pages 11, 13, 17, 58, 60, 61

O'Hagan, A. (1991). Bayes-Hermite Quadrature. *Journal of Statistical Planning and Inference*, 29:245–260. pages 61

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann. pages 23

Petersen, K. B. and Pedersen, M. S. (2012). The Matrix Cookbook. Version 20121115. pages 33

Rasmussen, C. E. and Ghahramani, Z. (2001). Occam's Razor. In *Advances in Neural Information Processing Systems 13*, pages 294–300. The MIT Press. pages 62

Rasmussen, C. E. and Ghahramani, Z. (2003). Bayesian Monte Carlo. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 489–496. The MIT Press, Cambridge, MA, USA. pages 61

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA. pages 13, 125

Roweis, S. and Ghahramani, Z. (1999). A Unifying Review of Linear Gaussian Models. *Neural Computation*, 11(2):305–345. pages 13

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536. pages 36, 54

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels—Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, MA, USA. pages 125

Schölkopf, B., Smola, A. J., and Müller, K.-R. (1998). Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10(5):1299–1319. pages 125

Schwarz, G. E. (1978). Estimating the Dimension of a Model. *Annals of Statistics*, 6(2):461–464. pages 63

Shental, O., Bickson, D., P. H. Siegel and, J. K. W., and Dolev, D. (2008). Gaussian Belief Propagatio Solver for Systems of Linear Equations. In *IEEE International Symposium on Information Theory*. pages 23

Shor, N. Z. (1985). *Minimization Methods for Non-differentiable Functions*. Springer. pages 56

Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2006). TextonBoost: Joint Appearance, Shape and Context Modeling for Mulit-Class Object Recognition and Segmentation. In *Proceedings of the European Conference on Computer Vision*. pages 23

Spiegelhalter, D. and Smith, A. F. M. (1980). Bayes Factors and Choice Criteria for Linear Models. *Journal of the Royal Statistical Society B*, 42(2):213–220. pages 58

Stoer, J. and Burlirsch, R. (2002). *Introduction to Numerical Analysis*. Springer. pages 61, 125

Sucar, L. E. and Gillies, D. F. (1994). Probabilistic Reasoning in High-Level Vision. *Image and Vision Computing*, 12(1):42–60. pages 23

Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., and Rother, C. (2008). A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-based Priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):1068–1080. pages 23

Tibshirani, R. (1996). Regression Selection and Shrinkage via the Lasso. *Journal of the Royal Statistical Society B*, 58(1):267–288. pages 50

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society: Series B*, 61(3):611–622. pages 13

Toussaint, M. (2012). Some Notes on Gradient Descent. pages 54

Turk, M. and Pentland, A. (1991). Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86. pages 2