Multivariate Statistics and Machine Learning

Korbinian Strimmer

15 December 2022

Contents

Welcome

These are the lecture notes for MATH38161, a course in **Multivariate Statistics and Machine Learning** for third year mathematics students at the Department of Mathematics of the University of Manchester.

The course text was written by Korbinian Strimmer from 2018–2022. This version is from 15 December 2022.

The notes will be updated from time to time. To view the current version visit the online MATH38161 lecture notes.

You may also download the MATH38161 lecture notes as PDF. For a paper copy it is recommended to print two pages per sheet.

License

Preface

About the author

Hello! My name is Korbinian Strimmer and I am a Professor in Statistics. I am a member of the Statistics group at the Department of Mathematics of the University of Manchester. You can find more information about me on my home page.

I have first taught this module in the winter term 2018 at the University of Manchester, and subsequently also in the years 2019–2022.

I hope you enjoy the course! If you have any questions, comments, or corrections then please email me at korbinian.strimmer@manchester.ac.uk.

About the module

Topics covered

The MATH38161 module is designed to run over the course of 11 weeks. It has six parts, each covering a particular aspect of multivariate statistics and machine learning:

1. Multivariate random variables and estimation in large and small sample settings (W1 and W2)
2. Transformations and dimension reduction (W3 and W4)
3. Unsupervised learning/clustering (W5 and W6)
4. Supervised learning/classification (W7 and W8)
5. Measuring and modelling multivariate dependencies (W9)
6. Nonlinear and nonparametric models (W10, W11)

This module focuses on:

- *Concepts and methods* (not on theory)
- *Implementation and application in R*
- *Practical data analysis and interpretation* (incl. report writing)
- *Modern tools in data science and statistics* (R markdown, R studio)

Additional support material

If you are a University of Manchester student and enrolled in this module you will find on Blackboard:

- a weekly learning plan for an 11 week study period,
- weekly worksheets with with examples (theory and application in R) and solutions in R Markdown, and
- exam papers of previous years.

Furthermore, there is also an MATH38161 online reading list hosted by the University of Manchester library.

Acknowledgements

Many thanks to Beatriz Costa Gomes for her help to compile the first draft of these course notes in the winter term 2018 while she was a graduate teaching assistant for this course. I also thank the many students who suggested corrections.

Chapter 1

Multivariate random variables

1.1 Essentials in multivariate statistics

1.1.1 Why multivariate statistics?

In science we use experiments to learn about underlying mechanisms of interest, both deterministic and stochastic, to compare different models and to verify or reject hypotheses about the world. Statistics provides tools to quantify this procedure and offers methods to link data (experiments) with probabilistic models (hypotheses).

In univariate statistics with we use relatively simple approaches based on a single random variable or single parameter. However, in practise we often have to consider multiple random variables and multiple parameters, so we need more complex models and also be able to deal with more complex data. Hence, the need for multivariate statistical approaches and models.

Specifically, multivariate statistics is concerned with methods and models for **random vectors** and **random matrices**, rather than just random univariate (scalar) variables. Therefore, in multivariate statistics we will frequently make use of matrix notation.

Closely related to multivariate statistics (traditionally a subfield of statistics) is machine learning (ML) which is traditionally a subfield of computer science. ML used to focus more on algorithms rather on probabilistic modelling but nowadays most machine learning methods are fully based on statistical multivariate approaches, so the two fields are converging.

Multivariate models provide a means to learn dependencies and interactions among the components of the random variables which in turn allow us to draw conclusion about underlying mechanisms of interest (e.g. in biological or medical problems).

Two main tasks:

- unsupervised learning (finding structure, clustering)
- supervised learning (training from labelled data, followed by prediction)

Challenges:

- complexity of model needs to be appropriate for problem and available data
- high dimensions make estimation and inference difficult
- computational issues

1.1.2 Univariate vs. multivariate random variables

Univariate random variable (dimension $d = 1$):

$$x \sim F$$

where x is a **scalar** and F is the distribution. $\mathrm{E}(x) = \mu$ denotes the mean and $\mathrm{Var}(x) = \sigma^2$ the variance of x.

Multivariate random **vector** of dimension d:

$$\boldsymbol{x} = (x_1, x_2, ..., x_d)^T \sim F$$

$\boldsymbol{x}$ is **vector** valued random variable.

The vector $\boldsymbol{x}$ is column vector (=matrix of size $d \times 1$). Its components $x_1, x_2, ..., x_d$ are univariate random variables. The dimension d is also often denoted by p or q.

1.1.3 Multivariate data

Vector notation:

Samples from a multivariate distribution are *vectors* (not scalars as for univariate normal):

$$\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n \overset{\text{iid}}{\sim} F$$

Matrix and component notation:

All the data points are commonly collected into a matrix $\boldsymbol{X}$.

In statistics the convention is to store each data vector in the rows of the data matrix $\boldsymbol{X}$:

$$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)^T = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & & & \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{pmatrix}$$

Therefore,

$$\boldsymbol{x}_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1d} \end{pmatrix}, \boldsymbol{x}_2 = \begin{pmatrix} x_{21} \\ \vdots \\ x_{2d} \end{pmatrix}, \ldots, \boldsymbol{x}_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{nd} \end{pmatrix}$$

Thus, in statistics the first index runs over $(1, ..., n)$ and denotes the samples while the second index runs over $(1, ..., d)$ and refers to the variables.

The **statistics convention on data matrices** is *not* universal! In fact, in most of the machine learning literature in engineering and computer science the data samples are stored in the columns so that the variables appear in the rows (thus in the engineering convention the data matrix is transposed compared to the statistics convention).

In order to avoid confusion and ambiguity it is recommended to prefer vector notation to describe data over matrix or component notation (see also the section below on estimating covariance matrices for examples).

1.1.4 Mean of a random vector

The mean / expectation of a random vector with dimensions d is also a vector with dimensions d:

$$\mathrm{E}(\boldsymbol{x}) = \boldsymbol{\mu} = \begin{pmatrix} \mathrm{E}(x_1) \\ \mathrm{E}(x_2) \\ \vdots \\ \mathrm{E}(x_d) \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_d \end{pmatrix}$$

1.1.5 Variance of a random vector

Recall the definition of mean and variance for a univariate random variable:

$$\mathrm{E}(x) = \mu$$

$$\mathrm{Var}(x) = \sigma^2 = \mathrm{E}((x-\mu)^2) = \mathrm{E}((x-\mu)(x-\mu)) = \mathrm{E}(x^2) - \mu^2$$

Definition of **variance of a random vector:**

$$\mathrm{Var}(\boldsymbol{x}) = \underbrace{\boldsymbol{\Sigma}}_{d \times d} = \mathrm{E}\left(\underbrace{(\boldsymbol{x} - \boldsymbol{\mu})}_{d \times 1} \underbrace{(\boldsymbol{x} - \boldsymbol{\mu})^T}_{1 \times d} \right) = \mathrm{E}(\boldsymbol{x}\boldsymbol{x}^T) - \boldsymbol{\mu}\boldsymbol{\mu}^T$$

The variance of a random vector is, therefore, **not** a vector but a **matrix**!

$$\mathbf{\Sigma} = (\sigma_{ij}) = \begin{pmatrix} \sigma_{11} & \dots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \dots & \sigma_{dd} \end{pmatrix}$$

This matrix is called the **Covariance Matrix**, with off-diagonal elements $\sigma_{ij} = \text{Cov}(x_i, x_j)$ and the diagonal $\sigma_{ii} = \text{Var}(X_i) = \sigma_i^2$.

1.1.6 Properties of the covariance matrix

1. $\mathbf{\Sigma}$ is real valued: $\sigma_{ij} \in \mathbb{R}$
2. $\mathbf{\Sigma}$ is symmetric: $\sigma_{ij} = \sigma_{ji}$
3. The diagonal of $\mathbf{\Sigma}$ contains $\sigma_{ii} = \text{Var}(x_i) = \sigma_i^2$, i.e. the variances of the components of $\boldsymbol{x}$.
4. Off-diagonal elements $\sigma_{ij} = \text{Cov}(x_i, x_j)$ represent linear dependencies among the x_i. $\Longrightarrow$ linear regression, correlation

How many separate entries does $\mathbf{\Sigma}$ have?

$$\mathbf{\Sigma} = (\sigma_{ij}) = \underbrace{\begin{pmatrix} \sigma_{11} & \dots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \dots & \sigma_{dd} \end{pmatrix}}_{d \times d}$$

with $\sigma_{ij} = \sigma_{ji}$.

Number of separate entries: $\frac{d(d+1)}{2}$.

This numbers grows with the square of the dimension d, i.e. is of order $O(d^2)$:

d	# entries
1	1
10	55
100	5050
1000	500500
10000	50005000

For large dimension d the covariance matrix has many components!

–> computationally expensive (both for storage and in handling) –> very challenging to estimate in high dimensions d.

Note: matrix inversion requires $O(d^3)$ operations using standard algorithms

such as Gauss Jordan elimination.[1] Hence, computing $\mathbf{\Sigma}^{-1}$ is computationally expensive for large d!

1.1.7 Eigenvalue decomposition of $\mathbf{\Sigma}$

Recall from linear matrix algebra that any real symmetric matrix has real eigenvalues and a complete set of orthogonal eigenvectors. These can be obtained by orthogonal eigendecomposition.

Applying eigenvalue decomposition to the covariance matrix yields

$$\mathbf{\Sigma} = \boldsymbol{U}\mathbf{\Lambda}\boldsymbol{U}^T$$

where $\boldsymbol{U}$ is an orthogonal matrix containing the eigenvectors of the covariance matrix and

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_d \end{pmatrix}$$

contains the corresponding eigenvalues λ_i.

Importantly, the eigenvalues of a covariance matrix are not only real-valued but are by construction further constrained to be non-negative. This can be seen by computing the quadratic form $z^T\mathbf{\Sigma}z$ where z is a non-random vector. For any non-zero z

$$\begin{aligned} z^T\mathbf{\Sigma}z &= z^T\mathrm{E}\left((x-\mu)(x-\mu)^T\right)z \\ &= \mathrm{E}\left(z^T(x-\mu)(x-\mu)^Tz\right) \\ &= \mathrm{E}\left(\left(z^T(x-\mu)\right)^2\right) \geq 0\,. \end{aligned}$$

Furthermore, with $y = \boldsymbol{U}^T z$ we get

$$\begin{aligned} z^T\mathbf{\Sigma}z &= z^T\boldsymbol{U}\mathbf{\Lambda}\boldsymbol{U}^Tz \\ &= y^T\mathbf{\Lambda}y = \sum_{i=1}^{d} y_i^2\lambda_i \end{aligned}$$

and hence all the $\lambda_i \geq 0$. Therefore the **covariance matrix $\mathbf{\Sigma}$ is always positive semi-definite**.

In fact, **unless there is collinearity** (i.e. a variable is a linear function the other variables) all eigenvalues will be positive and **$\mathbf{\Sigma}$ is positive definite**.

[1] Specialised matrix algorithms improve this to about $O(d^{2.373})$. Matrices with special symmetries (e.g. diagonal and block diagonal matrices) or properties (e.g. orthogonal matrix) can also be inverted much easier (see Appendix A).

1.1.8 Joint covariance matrix

Assume we have random vector $\boldsymbol{z}$ with mean $\mathrm{E}(\boldsymbol{z}) = \boldsymbol{\mu}_z$ and covariance matrix $\mathrm{Var}(\boldsymbol{z}) = \boldsymbol{\Sigma}_z$.

Often it makes sense to partion the components of $\boldsymbol{z}$ into two groups

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix}$$

This induces a corresponding partition in the expectation

$$\boldsymbol{\mu}_z = \begin{pmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{pmatrix}$$

where $\mathrm{E}(\boldsymbol{x}) = \boldsymbol{\mu}_x$ and $\mathrm{E}(\boldsymbol{y}) = \boldsymbol{\mu}_y$.

Furthermore, the covariance matrix can then be written as

$$\boldsymbol{\Sigma}_z = \begin{pmatrix} \boldsymbol{\Sigma}_x & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_y \end{pmatrix}$$

containing the within-group group covariance matrices $\boldsymbol{\Sigma}_x$ and $\boldsymbol{\Sigma}_y$ on the diagonal and the cross-covariance matrix $\boldsymbol{\Sigma}_{xy} = \boldsymbol{\Sigma}_{yx}^T$ as off-diagonal element. Note that the cross-covariance matrix is rectangular and not symmetric.

We above covariance matrix is often called **joint covariance matrix** for $\boldsymbol{x}$ and $\boldsymbol{y}$.

1.1.9 Quantities related to the covariance matrix

1.1.9.1 Correlation matrix $\boldsymbol{P}$

The correlation matrix $\boldsymbol{P}$ (= upper case greek "rho") is the standardised covariance matrix

$$\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}} = \mathrm{Cor}(x_i, x_j)$$

$$\rho_{ii} = 1 = \mathrm{Cor}(x_i, x_i)$$

$$\boldsymbol{P} = (\rho_{ij}) = \begin{pmatrix} 1 & \dots & \rho_{1d} \\ \vdots & \ddots & \vdots \\ \rho_{d1} & \dots & 1 \end{pmatrix}$$

where $\boldsymbol{P}$ ("upper case rho") is a symmetric matrix ($\rho_{ij} = \rho_{ji}$).

Note the **variance-correlation decomposition**

$$\boldsymbol{\Sigma} = \boldsymbol{V}^{\frac{1}{2}} \boldsymbol{P} \boldsymbol{V}^{\frac{1}{2}}$$

where $\boldsymbol{V}$ is a diagonal matrix containing the variances:

$$\boldsymbol{V} = \begin{pmatrix} \sigma_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_{dd} \end{pmatrix}$$

$$\boldsymbol{P} = \boldsymbol{V}^{-\frac{1}{2}} \boldsymbol{\Sigma} \boldsymbol{V}^{-\frac{1}{2}}$$

This is the definition of correlation written in matrix notation.

As with the covariance matrix, in many applications it makes sense to partition a joint correlation matrix

$$\boldsymbol{P_z} = \begin{pmatrix} \boldsymbol{P_x} & \boldsymbol{P_{xy}} \\ \boldsymbol{P_{yx}} & \boldsymbol{P_y} \end{pmatrix}$$

into within-group group correlation matrices $\boldsymbol{P_x}$ and $\boldsymbol{P_y}$ and a cross-correlation matrix $\boldsymbol{P_{xy}} = \boldsymbol{P_{yx}^T}$

1.1.9.2 Precision matrix or concentration matrix

$$\boldsymbol{\Omega} = (\omega_{ij}) = \boldsymbol{\Sigma}^{-1}$$

$\boldsymbol{\Omega}$ ("Omega") is the inverse of the covariance matrix.

The inverse of the covariance matrix can be obtained via the spectral decomposition, followed by inverting the eigenvalues λ_i:

$$\boldsymbol{\Sigma}^{-1} = \boldsymbol{U} \boldsymbol{\Lambda}^{-1} \boldsymbol{U}^T = \boldsymbol{U} \begin{pmatrix} \lambda_1^{-1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_d^{-1} \end{pmatrix} \boldsymbol{U}^T$$

Note that **all eigenvalues λ_i need to be positive so that $\boldsymbol{\Sigma}$ can be inverted.** (i.e., $\boldsymbol{\Sigma}$ needs to be positive definite).
If any $\lambda_i = 0$ then $\boldsymbol{\Sigma}$ is singular and not invertible.

Importance of $\boldsymbol{\Sigma}^{-1}$:

- Many expressions in multivariate statistics contain $\boldsymbol{\Sigma}^{-1}$ and not $\boldsymbol{\Sigma}$.
- $\boldsymbol{\Sigma}^{-1}$ has close connection with graphical models (e.g. conditional independence graph, partial correlations).
- $\boldsymbol{\Sigma}^{-1}$ is a natural parameter from an exponential family perspective.

1.1.9.3 Partial correlation matrix

This is a standardised version of the precision matrix, see later chapter on graphical models.

1.1.9.4 Total variation and generalised variance

To summarise the covariance matrix $\mathbf{\Sigma}$ in a single scalar value there are two commonly used measures:

- **total variation**: $\text{Tr}(\mathbf{\Sigma}) = \sum_{i=1}^{d} \lambda_i$
- **generalised variance**: $\det(\mathbf{\Sigma}) = \prod_{i=1}^{d} \lambda_i$

The generalised variance $\det(\mathbf{\Sigma})$ is also known as the volume of $\mathbf{\Sigma}$.

1.2 Multivariate normal distribution

The multivariate normal model is a generalisation of the univariate normal distribution from dimension 1 to dimension d.

1.2.1 Univariate normal distribution:

$$\text{Dimension } d = 1$$

$$x \sim N(\mu, \sigma^2)$$

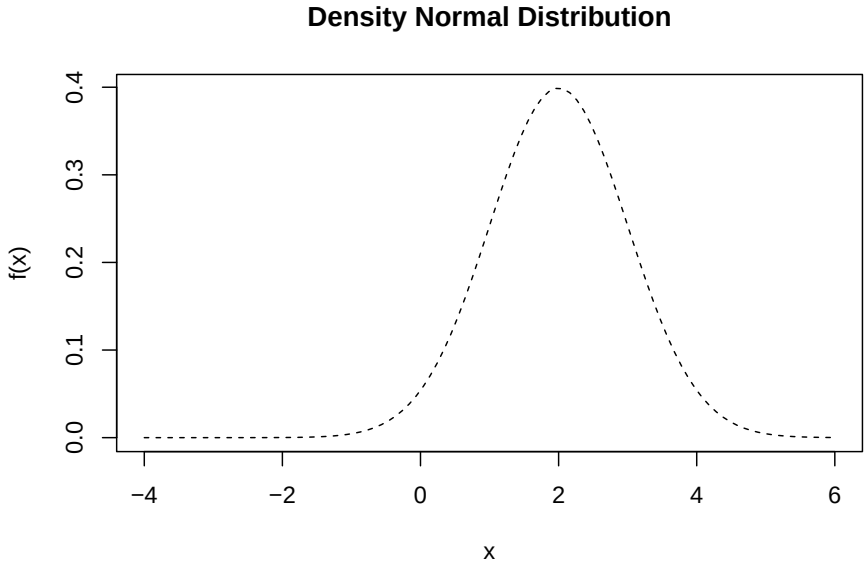
$$\text{E}(x) = \mu, \text{Var}(x) = \sigma^2$$

Density:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Plot of univariate normal density :

Unimodal with peak at μ, width determined by σ (in this plot: $\mu = 2, \sigma^2 = 1$)



Special case: **standard normal** with $\mu = 0$ and $\sigma^2 = 1$:

$$f(x|\mu = 0, \sigma^2 = 1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

Differential entropy:

$$H(F) = \frac{1}{2}(\log(2\pi\sigma^2) + 1)$$

Cross-entropy:

$$H(F_{\text{ref}}, F) = \frac{1}{2}\left(\frac{(\mu - \mu_{\text{ref}})^2}{\sigma^2} + \frac{\sigma_{\text{ref}}^2}{\sigma^2} + \log(2\pi\sigma^2)\right)$$

KL divergence:

$$D_{\text{KL}}(F_{\text{ref}}, F) = H(F_{\text{ref}}, F) - H(F_{\text{ref}}) = \frac{1}{2}\left(\frac{(\mu - \mu_{\text{ref}})^2}{\sigma^2} + \frac{\sigma_{\text{ref}}^2}{\sigma^2} - \log\left(\frac{\sigma_{\text{ref}}^2}{\sigma^2}\right) - 1\right)$$

Maximum entropy characterisation: the normal distribution is the unique distribution that has the highest (differential) entropy over all continuous distributions with support from minus infinity to plus infinity with a given mean and variance.

This is in fact one of the reasons why the normal distribution is so important (und useful) – if we only know that a random variable has a mean and variance,

and not much else, then using the normal distribution will be a reasonable and well justified working assumption!

1.2.2 Multivariate normal model

$$\text{Dimension } d$$

$$\boldsymbol{x} \sim N_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$\boldsymbol{x} \sim \text{MVN}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$\text{E}(\boldsymbol{x}) = \boldsymbol{\mu}, \text{Var}(\boldsymbol{x}) = \boldsymbol{\Sigma}$$

Density:

$$f(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \det(2\pi\boldsymbol{\Sigma})^{-\frac{1}{2}} \exp\left(\underbrace{-\frac{1}{2} \underbrace{(\boldsymbol{x}-\boldsymbol{\mu})^T}_{1\times d} \underbrace{\boldsymbol{\Sigma}^{-1}}_{d\times d} \underbrace{(\boldsymbol{x}-\boldsymbol{\mu})}_{d\times 1}}_{1\times 1=\text{scalar!}} \right)$$

- note that density contains precision matrix $\boldsymbol{\Sigma}^{-1}$
- inverting $\boldsymbol{\Sigma}$ implies inverting the eigenvalues λ_i of $\boldsymbol{\Sigma}$ (thus we need $\lambda_i > 0$)
- density also contains $\det(\boldsymbol{\Sigma}) = \prod_{i=1}^{d} \lambda_i \equiv$ product of eigenvalues of $\boldsymbol{\Sigma}$

Special case: **standard multivariate normal** with

$$\boldsymbol{\mu} = \boldsymbol{0}, \boldsymbol{\Sigma} = \boldsymbol{I} = \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix}$$

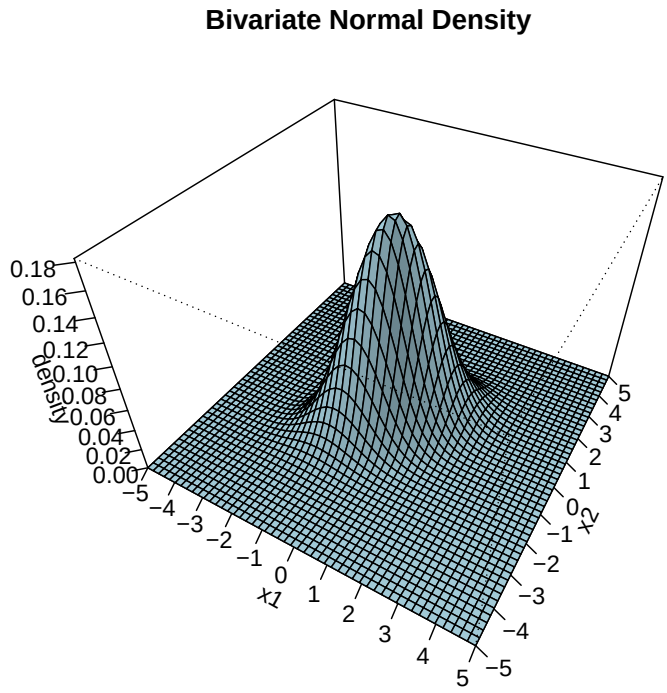
$$f(\boldsymbol{x}|\boldsymbol{\mu} = \boldsymbol{0}, \boldsymbol{\Sigma} = \boldsymbol{I}) = (2\pi)^{-d/2} \exp\left(-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{x}\right) = \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x_i^2}{2}\right)$$

which is equivalent to the product of d univariate standard normals!

Misc:

- for $d = 1$, multivariate normal reduces to normal.
- for $\boldsymbol{\Sigma}$ diagonal (i.e. $\boldsymbol{P} = \boldsymbol{I}$, no correlation), MVN is the product of univariate normals (see Worksheet 2).

Plot of MVN density:



- Location: $\boldsymbol{\mu}$
- Shape: $\boldsymbol{\Sigma}$
- Unimodal: **one** peak
- Support from $-\infty$ to $+\infty$ in each dimension

An interactive R Shiny web app of the bivariate normal density plot is available online at https://minerva.it.manchester.ac.uk/shiny/strimmer/bvn/ .

Differential entropy:

$$H = \frac{1}{2}(\log \det(2\pi\boldsymbol{\Sigma}) + d)$$

Cross-entropy:

$$H(F_{\text{ref}}, F) = \frac{1}{2}\left\{(\mu - \mu_{\text{ref}})^T \Sigma^{-1}(\mu - \mu_{\text{ref}}) + \text{Tr}\left(\Sigma^{-1}\Sigma_{\text{ref}}\right) + \log\det\left(2\pi\Sigma\right)\right\}$$

KL divergence:

$$\begin{aligned} D_{\text{KL}}(F_{\text{ref}}, F) &= H(F_{\text{ref}}, F) - H(F_{\text{ref}}) \\ &= \frac{1}{2}\left\{(\mu - \mu_{\text{ref}})^T \Sigma^{-1}(\mu - \mu_{\text{ref}}) + \text{Tr}\left(\Sigma^{-1}\Sigma_{\text{ref}}\right) - \log\det\left(\Sigma^{-1}\Sigma_{\text{ref}}\right) - d\right\} \end{aligned}$$

1.2.3 Shape of the multivariate normal density

Now we show that the contour lines of the multivariate normal density always take on the form of an ellipse, and that the radii of the ellipse is determined by the eigenvalues of Σ.

We start by observing that a circle with radius r around the origin can be described as the set of points (x_1, x_2) satisfying $x_1^2 + x_2^2 = r^2$, or equivalently, $\frac{x_1^2}{r^2} + \frac{x_2^2}{r^2} = 1$. This is generalised to the shape of an ellipse by allowing (in two dimensions) for two radii r_1 and r_2 with $\frac{x_1^2}{r_1^2} + \frac{x_2^2}{r_2^2} = 1$, or in vector notation $x^T \text{Diag}(r_1^2, r_2^2)^{-1} x = 1$. In d dimensions and allowing for rotation of the axes and a shift of the origin from 0 to μ the condition for an ellipse is

$$(x - \mu)^T Q \,\text{Diag}(r_1^2, \ldots, r_d^2)^{-1} Q^T (x - \mu) = 1$$

where Q is an orthogonal matrix whose column vectors indicate the direction of the axes.

A contour line of a probability density function is a set of connected points where the density assumes the same constant value. In the case of the multivariate normal distribution keeping the density at some fixed value implies that $(x - \mu)^T \Sigma^{-1}(x - \mu) = c$ where c is a constant. Using the eigenvalue decomposition of $\Sigma = U \Lambda U^T$ we can rewrite this condition as

$$(x - \mu)^T U \Lambda^{-1} U^T (x - \mu) = c\,.$$

This implies that

- the contour lines of the multivariate normal density are indeed ellipses,
- the squared radii are proportional to the eigenvalues of Σ and
- the direction of the axes correspond to the eigenvectors in U.

Equivalently, the positive square roots of the eigenvalues are proportional to the radii of the ellipse. Hence, for a singular covariance matrix with one or more $\lambda_i = 0$ the corresponding radii are zero.

An interactive R Shiny web app to play with the contour lines of the bivariate normal distribution is online at https://minerva.it.manchester.ac.uk/shiny/strimmer/bvn/ .

1.2.4 Three types of covariances

Following the above we can parameterise a covariance matrix in terms of its

i) volume,
ii) shape, and
iii) orientation

by writing

$$\boldsymbol{\Sigma} = \kappa\, \boldsymbol{U}\boldsymbol{A}\boldsymbol{U}^T = \boldsymbol{U}\ (\kappa \boldsymbol{A})\ \boldsymbol{U}^T$$

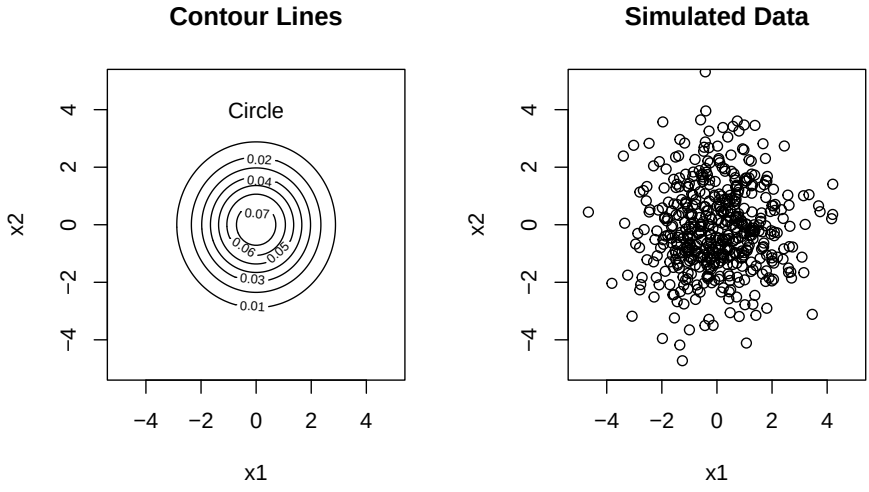
with $\boldsymbol{A} = \text{Diag}(a_1, \ldots, a_d)$ and $\det(\boldsymbol{A}) = \prod_{i=1}^{d} a_i = 1$. Note that in this parameterisation the eigenvalues of $\boldsymbol{\Sigma}$ are $\lambda_i = \kappa a_i$.

i) The **volume** is $\det(\boldsymbol{\Sigma}) = \kappa^d$, determined by a single parameter κ. This parameter can be interpreted as the length of the side of a d-dimensional hypercube.
ii) The **shape** is determined by the diagonal matrix $\boldsymbol{A}$ with $d-1$ free parameters. Note that there are only $d-1$ and not d free parameters because of the constraint $\det(\boldsymbol{A}) = 1$.
iii) The **orientation** is given by the orthogonal matrix $\boldsymbol{U}$, with $d(d-1)/2$ free parameters.

This leads to classification of covariances into three varieties:

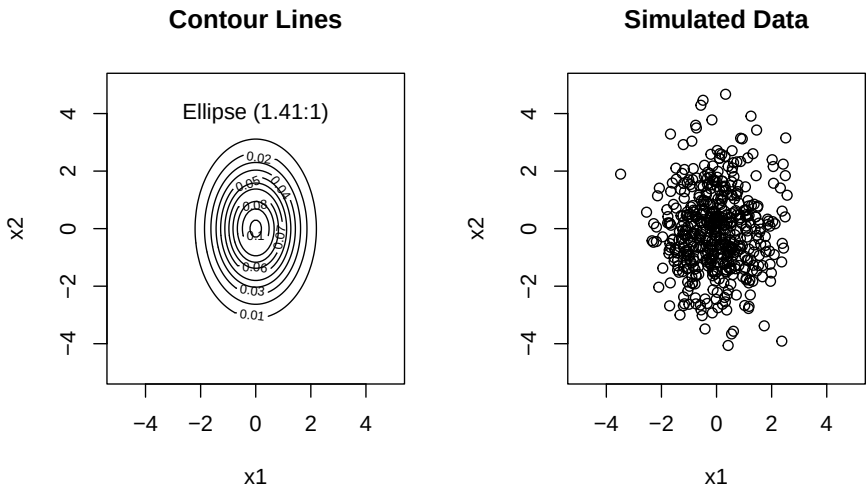
Type 1: spherical covariance $\boldsymbol{\Sigma} = \kappa \boldsymbol{I}$, with spherical contour lines, 1 free parameter ($\boldsymbol{A} = \boldsymbol{I}, \boldsymbol{U} = \boldsymbol{I}$).

Example: $\boldsymbol{\Sigma} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$ with $\sqrt{\lambda_1/\lambda_2} = 1$:



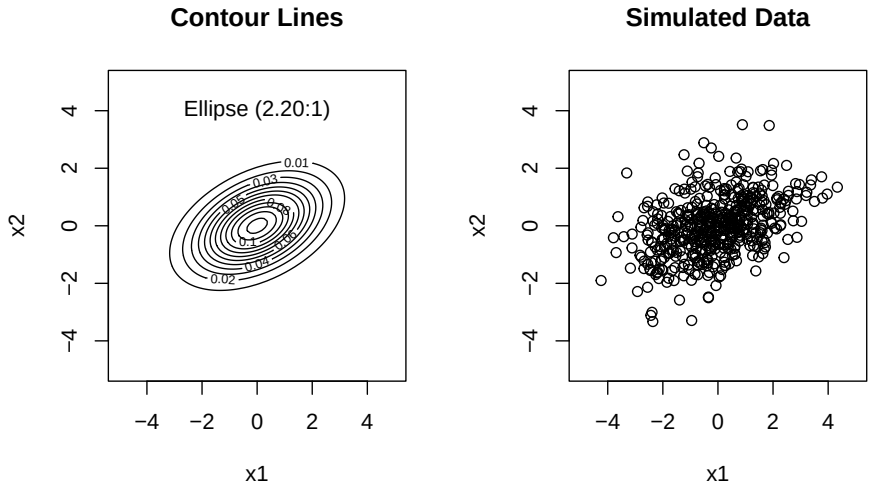
Type 2: **diagonal covariance $\boldsymbol{\Sigma} = \kappa \boldsymbol{A}$**, with elliptical contour lines and the axes of the ellipse oriented parallel to the coordinates, d free parameters ($\boldsymbol{U} = \boldsymbol{I}$).

Example: $\boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ with $\sqrt{\lambda_1/\lambda_2} \approx 1.41$:



Type 3: **general unrestricted covariance $\boldsymbol{\Sigma}$**, with elliptical contour lines, with the axes of the ellipse oriented according to the column vectors in $\boldsymbol{U}$, $d(d+1)/2$ free parameters.

Example: $\boldsymbol{\Sigma} = \begin{pmatrix} 2 & 0.6 \\ 0.6 & 1 \end{pmatrix}$ with $\sqrt{\lambda_1/\lambda_2} \approx 2.20$:



1.2.5 Concentration of probability mass for small and large dimension

The density of the multivariate normal distribution has a bell shape with a single mode. Intuitively, we may assume that most of the probability mass is always concentrated around this mode, as it is in the univariate case ($d = 1$). While this is still true for small dimensions (small d) we now show that this intuition is incorrect for high dimensions (large d).

For simplicity we consider the standard multivariate normal distribution with dimension d

$$\boldsymbol{x} \sim N_d(\boldsymbol{0}, \boldsymbol{I}_d)$$

with a spherical covariance $\boldsymbol{I}_d$ and sample $\boldsymbol{x}$. The squared Euclidean length of $\boldsymbol{x}$ is $r^2 = ||\boldsymbol{x}||^2 = \boldsymbol{x}^T\boldsymbol{x} = \sum_{i=1}^d x_i^2$. The corresponding density of the d-dimensional standard multivariate normal distribution is

$$g_d(\boldsymbol{x}) = (2\pi)^{-d/2} e^{-\boldsymbol{x}^T\boldsymbol{x}/2}$$

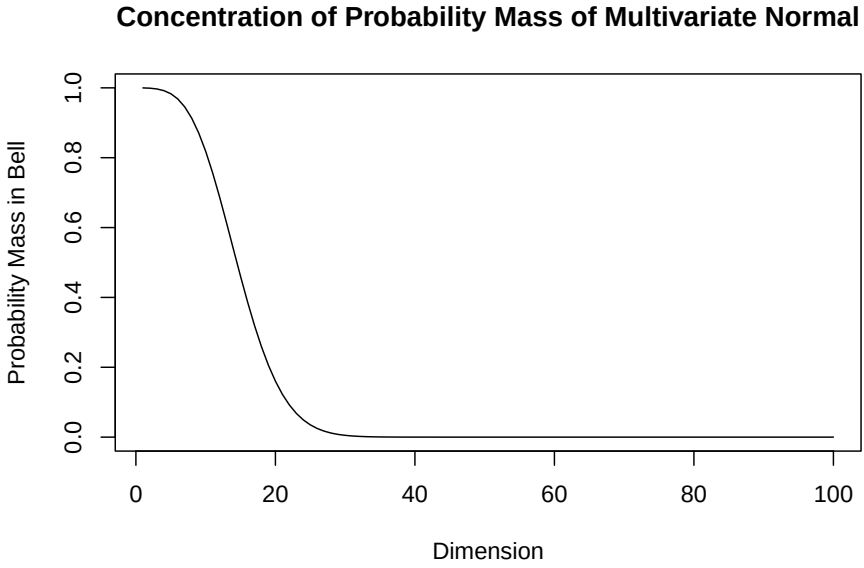
A natural way to define the main part of the "bell" of the standard multivariate normal as the set of all $\boldsymbol{x}$ for which the density is larger than a specified fraction η (say 0.001) of the maximum value of the density $g_d(0)$ at the peak at zero. To formalise

$$B = \left\{ \boldsymbol{x} : \frac{g_d(\boldsymbol{x})}{g_d(0)} > \eta \right\}$$

which can be equivalently written as the set

$$B = \{ \boldsymbol{x} : \boldsymbol{x}^T\boldsymbol{x} = r^2 < -2\log(\eta) = r^2_{\max} \}$$

Each individual component in the sample $\boldsymbol{x}$ is independently distributed as $x_i \sim N(0,1)$, hence $r^2 \sim \chi^2_d$ is chi-squared distributed with degree of freedom d. The probability $\Pr(\boldsymbol{x} \in B)$ can thus be obtained as the value of the cumulative density function of a chi-squared distribution with d degrees of freedom at $r^2_{\max}$. Computing this probability for fixed η as a function of the dimension d we obtain the following curve:



The above plot is for $\eta = 0.001$. You can see that for dimensions up to around $d = 10$ the probability mass is indeed concentrated the bell in the center but from $d = 30$ onwards it has moved completely to the tail of the distribution.

1.3 Estimation in large and small sample settings

In practical application of multivariate normal model we need to learn its parameters from observed data points:

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \overset{\text{iid}}{\sim} N_d\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$$

We first consider the case when there are many measurements available (n large), and then subsequently the case when the number of data points n is small compared to the dimenions and the number of parameters.

In a previous course in year 2 (see MATH20802 Statistical Methods) the method of maximum likelihood as well as the essentials of Bayesian statistics were

introduced. Here we apply these approaches to the problem of estimating the parameters of the multivariate normal distribution.

1.3.1 Strategies for large sample estimation

1.3.1.1 Empirical estimators (outline)

For large n we have thanks to the law of large numbers:

$$\underbrace{F}_{\text{true}} \approx \underbrace{\widehat{F}_n}_{\text{empirical}}$$

We now would like to estimate A which is a *functional* $A = m(F)$ of the distribution F — recall that a functional is a function that takes another function as argument. For example all standard distributional summaries such as the mean, the median etc. are derived from F and hence are functionals of F.

The *empirical estimate* is obtained by replacing the unknown true distribution F with the observed empirical distribution: $\hat{A} = m(\widehat{F}_n)$.

For example, the expectation of a random variable is approximated/estimated as the average over the observations:

$$\mathrm{E}_F(\boldsymbol{x}) \approx \mathrm{E}_{\widehat{F}_n}(\boldsymbol{x}) = \frac{1}{n}\sum_{k=1}^{n} \boldsymbol{x}_k$$

$$\mathrm{E}_F(g(\boldsymbol{x})) \approx \mathrm{E}_{\widehat{F}_n}(g(\boldsymbol{x})) = \frac{1}{n}\sum_{k=1}^{n} g(\boldsymbol{x}_k)$$

Simple recipe to obtain an empirical estimator: simply replace the expectation operator by the sample average.

What does this work: the empirical distribution $\widehat{F}_n$ is the nonparametric maximum likelihood estimate of F (see below for likelihood estimation).

Note: the approximation of F by $\widehat{F}_n$ is also the basis other approaches such as Efron's bootstrap method (1979).[2]

1.3.1.2 Maximum likelihood estimation (outline)

R.A. Fisher (1922):[3] model-based estimators using the density or probability mass function

[2] Efron, B. 1979. Bootstrap methods: Another look at the jackknife. The Annals of Statistics **7**:1–26. https://doi.org/10.1214/aos/1176344552

[3] Fisher, R. A. 1922. On the mathematical foundations of theoretical statistics. Philosophical Transactions of the Royal Society A **222**:309–368. https://doi.org/10.1098/rsta.1922.0009

log-likelihood function:

$$\log L(\boldsymbol{\theta}) = \sum_{k=1}^{n} \underbrace{\log f}_{\text{log-density}} \Big(\underbrace{x_i}_{\text{data}} \Big| \underbrace{\boldsymbol{\theta}}_{\text{parameters}} \Big)$$

likelihood = probability to observe data given the model parameters

Maximum likelihood estimate:

$$\hat{\boldsymbol{\theta}}^{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \log L(\boldsymbol{\theta})$$

Maximum likelihood (ML) finds the parameters that make the observed data most likely (it does *not* find the most probable model!)

Recall from MATH20802 Statistical Methods that maximum likelihood is closely linked to minimising the relative entropy (KL divergence) $D_{\text{KL}}(F, F_{\boldsymbol{\theta}})$ between the unknown true model F to the specified model $F_{\boldsymbol{\theta}}$. Specifically, for large sample size n the model $F_{\hat{\boldsymbol{\theta}}}$ fit by maximum likelihood is indeed the model that is closest to F.

Correspondingly, the great appeal of **maximum likelihood estimates** (MLEs) is that they **are optimal for large n**, i.e. so that **for large sample size no estimator can be constructed that outperforms the MLE** (note the emphasis on "for large n"!). A further advantage of the method of maximum likelihood is that it does not only provide a point estimate but also the asymptotic error (via the Fisher information which is related to the curvature of the log-likelihood function).

1.3.2 Large sample estimates of mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

1.3.2.1 Empirical estimates:

Recall the definitions:

$$\boldsymbol{\mu} = \mathrm{E}(\boldsymbol{x})$$

and

$$\boldsymbol{\Sigma} = \mathrm{E}\left((\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T\right)$$

For the empirical estimate we replace the expectations by the corresponding sample averages.

These resulting estimators can be written in three different ways:

Vector notation:

$$\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{k=1}^{n} \boldsymbol{x}_k$$

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\sum_{k=1}^{n}(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})^T = \frac{1}{n}\sum_{k=1}^{n}\boldsymbol{x}_k\boldsymbol{x}_k^T - \hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T$$

Data matrix notation:

The empirical mean and covariance can also be written in terms of the data matrix $\boldsymbol{X}$ (using statistics convention):

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\boldsymbol{X}^T\mathbf{1}_n$$

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\boldsymbol{X}^T\boldsymbol{X} - \hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T$$

Component notation:

The corresponding component notation with $\boldsymbol{X} = (x_{ki})$ is:

$$\hat{\mu}_i = \frac{1}{n}\sum_{k=1}^{n}x_{ki}$$

$$\hat{\sigma}_{ij} = \frac{1}{n}\sum_{k=1}^{n}(x_{ki} - \hat{\mu}_i)(x_{kj} - \hat{\mu}_j)$$

$$\hat{\boldsymbol{\mu}} = \begin{pmatrix}\hat{\mu}_1\\ \vdots \\ \hat{\mu}_d\end{pmatrix}, \widehat{\boldsymbol{\Sigma}} = (\hat{\sigma}_{ij})$$

Variance estimate:

$$\hat{\sigma}_{ii} = \frac{1}{n}\sum_{k=1}^{n}(x_{ki} - \hat{\mu}_i)^2$$

Note the factor $\frac{1}{n}$ (not $\frac{1}{n-1}$)

Engineering and machine learning convention:

Using the engineering and machine learning convention for the data matrix $\boldsymbol{X}$ the estimators are written as

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\boldsymbol{X}\mathbf{1}_n$$

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n}\boldsymbol{X}\boldsymbol{X}^T - \hat{\boldsymbol{\mu}}\hat{\boldsymbol{\mu}}^T$$

In the corresponding component notation the two indices for the columns and rowns are interchanged.

To avoid confusion when using matrix or component notation you need to always state which convention is used! In these notes we strictly follow the statistics convention.

1.3.2.2 Maximum likelihood estimates

We now derive the MLE of the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of the multivariate normal distribution. The corresponding log-likelihood function is

$$\begin{aligned}\log L(\boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \sum_{k=1}^{n} \log f(\boldsymbol{x}_k|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= -\frac{nd}{2}\log(2\pi) - \frac{n}{2}\log\det(\boldsymbol{\Sigma}) - \frac{1}{2}\sum_{k=1}^{n}(\boldsymbol{x}_k - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_k - \boldsymbol{\mu})\,.\end{aligned}$$

Written in terms of the precision matrix $\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1}$ this becomes

$$\log L(\boldsymbol{\mu}, \boldsymbol{\Omega}) = -\frac{nd}{2}\log(2\pi) + \frac{n}{2}\log\det(\boldsymbol{\Omega}) - \frac{1}{2}\sum_{k=1}^{n}(\boldsymbol{x}_k - \boldsymbol{\mu})^T\boldsymbol{\Omega}(\boldsymbol{x}_k - \boldsymbol{\mu})\,.$$

First, to find the MLE for $\boldsymbol{\mu}$ we compute

$$\nabla_{\mu}\log L(\boldsymbol{\mu}, \boldsymbol{\Omega}) = \frac{\partial \log L(\boldsymbol{\mu}, \boldsymbol{\Omega})}{\partial \boldsymbol{\mu}} = \sum_{k=1}^{n}\boldsymbol{\Omega}(\boldsymbol{x}_k - \boldsymbol{\mu})$$

noting that $\boldsymbol{\Omega}$ is symmetric (see the Appendix for rules in vector calculus). Setting this equal to zero we get $\sum_{k=1}^{n}\boldsymbol{x}_k = n\hat{\boldsymbol{\mu}}_{ML}$ and thus

$$\hat{\boldsymbol{\mu}}_{ML} = \frac{1}{n}\sum_{k=1}^{n}\boldsymbol{x}_k\,.$$

Next, to obtain the MLE for $\boldsymbol{\Omega}$ we compute

$$\frac{\partial \log L(\boldsymbol{\mu}, \boldsymbol{\Omega})}{\partial \boldsymbol{\Omega}} = \frac{n}{2}\boldsymbol{\Omega}^{-1} - \frac{1}{2}\sum_{k=1}^{n}(\boldsymbol{x}_k - \boldsymbol{\mu})(\boldsymbol{x}_k - \boldsymbol{\mu})^T$$

(see the Appendix for rules in matrix calculus). Setting this equal to zero and substituting the MLE for $\boldsymbol{\mu}$ we get

$$\widehat{\boldsymbol{\Omega}}_{ML}^{-1} = \frac{1}{n}\sum_{k=1}^{n}(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}})^T = \widehat{\boldsymbol{\Sigma}}_{ML}\,.$$

Therefore, the MLEs are identical to the empirical estimates.

Note the factor $\frac{1}{n}$ in the MLE of the covariance matrix.

1.3.2.3 Distribution of the empirical / maximum likelihood estimates

With $\boldsymbol{x}_1, ..., \boldsymbol{x}_n \sim N_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ one can find the exact distributions of the estimators.

1. Distribution of the estimate of the mean:

$$\hat{\boldsymbol{\mu}}_{ML} \sim N_d\left(\boldsymbol{\mu}, \frac{\boldsymbol{\Sigma}}{n}\right)$$

Since $\mathrm{E}(\hat{\boldsymbol{\mu}}_{ML}) = \boldsymbol{\mu} \Longrightarrow \hat{\boldsymbol{\mu}}_{ML}$ is unbiased.

2. Distribution of the covariance estimate:

$$\widehat{\boldsymbol{\Sigma}}_{ML} \sim \text{Wishart}\left(\frac{\boldsymbol{\Sigma}}{n}, n-1\right)$$

Since $\mathrm{E}(\widehat{\boldsymbol{\Sigma}}_{ML}) = \frac{n-1}{n}\boldsymbol{\Sigma} \Longrightarrow \widehat{\boldsymbol{\Sigma}}_{ML}$ is biased, with $\text{Bias}(\widehat{\boldsymbol{\Sigma}}_{ML}) = \boldsymbol{\Sigma} - \mathrm{E}(\widehat{\boldsymbol{\Sigma}}_{ML}) = -\frac{\boldsymbol{\Sigma}}{n}$.

Easy to make unbiased: $\widehat{\boldsymbol{\Sigma}}_{UB} = \frac{n}{n-1}\widehat{\boldsymbol{\Sigma}}_{ML} = \frac{1}{n-1}\sum_{k=1}^{n}\left(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}}\right)\left(\boldsymbol{x}_k - \hat{\boldsymbol{\mu}}\right)^T$ is unbiased.

But unbiasedness of an estimator is **not** a very relevant criterion in multivariate statistics as we will see in the next section.

1.3.3 Problems with maximum likelihood in small sample settings and high dimensions

Modern data is high dimensional!

Data sets with $n < d$, i.e. high dimension d and small sample size n are now common in many fields, e.g., medicine, biology but also finance and business analytics.

$$n = 100 \,(\text{e.g, patients/samples})$$

$$d = 20000 \,(\text{e.g., genes/SNPs/proteins/variables})$$

Reasons:

- the number of measured variables is increasing quickly with technological advances (e.g. genomics)
- but the number of samples cannot be similary increased (for cost and ethical reasons)

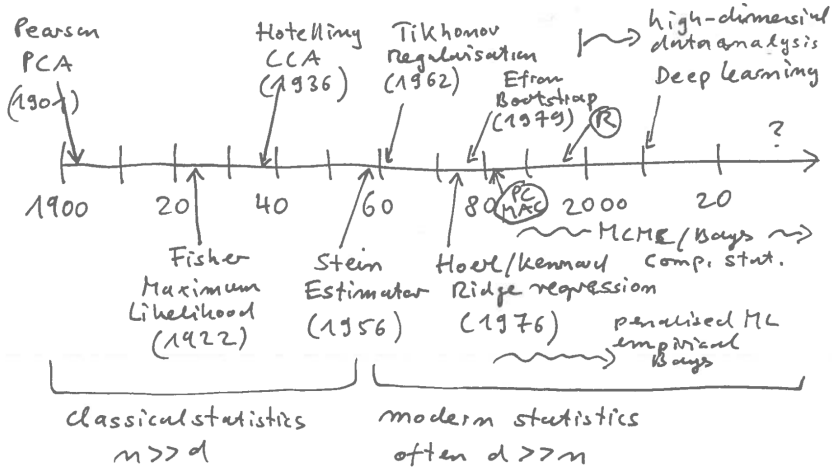
General problems of MLEs:

1. ML estimators are optimal only if **sample size is large** compared to the number of parameters. However, this optimality is not any more valid if sample size is moderate or smaller than the number of parameters.
2. If there is not enough data the **ML estimate overfits**. This means ML fits the current data perfectly but the resulting model does not generalise well (i.e. model will perform poorly in prediction)
3. If there is a choice between different models with different complexity **ML will always select the model with the largest number of parameters**.

-> for high-dimensional data with small sample size maximum likelihood estimation does not work!!!

History of Statistics:

Much of modern statistics (from 1960 onwards) is devoted to the development of inference and estimation techniques that work with complex, high-dimensional data.



- Maximum likelihood is a method from classical statistics (time up to about 1960).
- From 1960 modern (computational) statistics emerges, starting with **"Stein Paradox" (1956):** Charles Stein showed that in a **multivariate setting** ML estimators are **dominated by** (= are always worse than) shrinkage estimators!
- For example, there is a shrinkage estimator for the mean that is better (in terms of MSE) than the average (which is the MLE)!

Modern statistics has developed many different (but related) methods for use in high-dimensional, small sample settings:

- regularised estimators
- shrinkage estimators
- penalised maximum likelihood estimators

- Bayesian estimators
- Empirical Bayes estimators
- KL / entropy-based estimators

Most of this is out of scope for our class, but will be covered in advanced statistical courses.

Next, we describe a **simple regularised estimator for the estimation of the covariance** that we will use later (i.e. in classification).

1.3.4 Estimation of covariance matrix in small sample settings

Problems with ML estimate of $\boldsymbol{\Sigma}$

1. Σ has $O(d^2)$ number of parameters! $\implies \hat{\boldsymbol{\Sigma}}^{\text{MLE}}$ requires *a lot* of data! $n \gg d$ or d^2
2. if $n < d$ then $\hat{\boldsymbol{\Sigma}}$ is positive **semi**-definite (even if Σ is p.d.f.!)
   $\implies \hat{\boldsymbol{\Sigma}}$ will have **vanishing eigenvalues** (some $\lambda_i = 0$) and thus **cannot be inverted** and is singular!

Note that in many expression in multivariate statistics we actually need to use the inverse of the covariances matrix, e.g. in the density of the multivariate normal distribution, so it is essential that we get an invertible covariance matrix estimate.

Making the ML estimate of $\boldsymbol{\Sigma}$ invertible

There is a simple numerical trick credited to A. N. Tikhonov to make $\hat{\boldsymbol{\Sigma}}$ invertible, by adding a small number (say $\varepsilon = 10^{-6}$ to the diagonal elements of $\hat{\boldsymbol{\Sigma}}$:

$$\boldsymbol{S}_{\text{Tik}} = \hat{\boldsymbol{\Sigma}} + \varepsilon \boldsymbol{I}$$

The resulting $\boldsymbol{S}_{\text{Tik}}$ is **positive definite** because the sum of a symmetric positive definite matrix ($\varepsilon \boldsymbol{I}$) and a symmetric positive semi-definite matrix ($\hat{\boldsymbol{\Sigma}}$) is always positive definite (see Appendix A).

However, while this simple regularisation results in an invertible matrix the estimator itself has not improved over the MLE, and the matrix $\boldsymbol{S}_{\text{Tik}}$ will also be poorly conditioned (i.e. large condition number).

Simple Bayes-type regularised estimate of $\boldsymbol{\Sigma}$

Regularised estimator $\boldsymbol{S}^*$ = convex combination of $\boldsymbol{S} = \hat{\boldsymbol{\Sigma}}^{\text{MLE}}$ and $\boldsymbol{I}_d$ (identity matrix) to get

Regularisation:

$$\underbrace{\boldsymbol{S}^*}_{\text{regularised estimate}} = (1-\lambda) \underbrace{\boldsymbol{S}}_{\text{ML estimate}} + \underbrace{\lambda}_{\text{shrinkage intensity}} \underbrace{\boldsymbol{I}_d}_{\text{target}}$$

Idea: choose $\lambda \in [0, 1]$ such that $\boldsymbol{S}^*$ is better (e.g. in terms of MSE) than both $\boldsymbol{S}$ and $\boldsymbol{I}_d$. Note that λ does not need to be small like ε.

This form of estimator is corresponds to computing the mean of the Bayesian posterior by directly shrinking the MLE towards a prior mean (target):

$$\underbrace{\boldsymbol{S}^*}_{\text{posterior mean}} = \underbrace{\lambda \boldsymbol{I}_d}_{\text{prior information}} + (1-\lambda) \underbrace{\boldsymbol{S}}_{\text{data summarised by maximum likelihood}}$$

- Prior information helps to infer $\boldsymbol{\Sigma}$ even in small samples.
- also called shrinkage estimator since the off-diagonal entries are shrunk towards zero.
- this type of linear shrinkage/regularisation is natural for exponential family models (Diaconis and Ylvisaker, 1979).
- Instead of a diagonal target other options are possible, e.g. block-diagonal or banded covariances.
- If λ is not prespecified but learned from data (see below) then the resulting estimate is an empirical Bayes estimator.
- The resulting estimate will typically be biased as mixing in the target will increase the bias.

How to find optimal shrinkage / regularisation parameter λ?

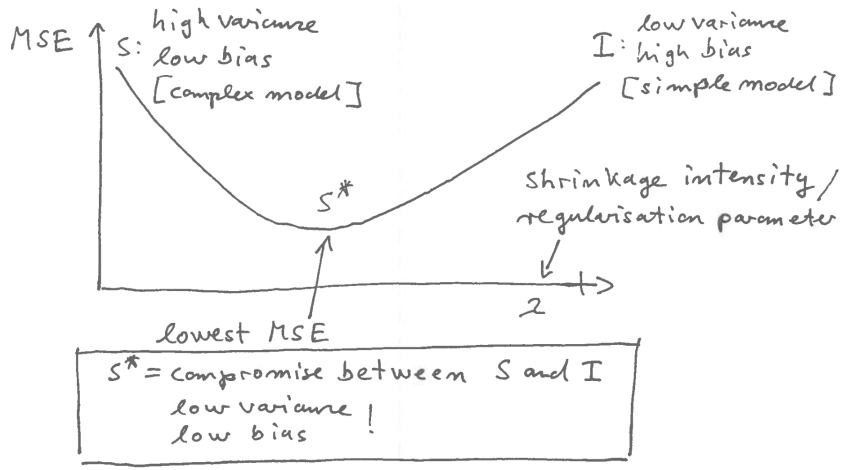
One way to do this is to minimise MSE (Mean Squared Error). This is also called L2 regularisation or Ridge regularisation.

Bias-variance trade-off: MSE is composed of squared bias and variance.

$$\text{MSE}(\theta) = \text{E}((\hat{\theta} - \theta)^2) = \text{Bias}(\hat{\theta})^2 + \text{Var}(\hat{\theta})$$

with $\text{Bias}(\hat{\theta}) = \text{E}(\hat{\theta}) - \theta$

$\boldsymbol{S}$: ML estimate, many parameters, low bias, high variance
$\boldsymbol{I}_d$: "target", no parameters, high bias, low variance
$\Longrightarrow$ **reduce high variance of $\boldsymbol{S}$ by *introducing* a bit of bias through $\boldsymbol{I}_d$!**
$\Longrightarrow$ overall, MSE is decreased



Challenge: since we don't know the true $\boldsymbol{\Sigma}$ we cannot actually compute the MSE directly but have to estimate it! How is this done in practise?

- by cross-validation (=resampling procedure)
- by using some analytic approximation (e.g. Stein's formula)

In Worksheet 3 the empirical estimator of covariance is compared with the regularised covariance estimator implemented in the R package "corpcor". This uses a regularisation similar as above (but for the correlation rather than the covariance matrix) and it employs an analytic data-adaptive estimate of the shrinkage intensity λ. This estimator is a variant of an empirical Bayes / James-Stein estimator (see MATH20802 Statistical Methods).

Summary

- In multivariate statistics, it is useful (and often necessary) to utilise prior information!
- Regularisation introduces bias and reduces variance, minimising overall MSE
- Unbiased estimation (a highly valued property in classical statistics!) is not a good idea in multivariate settings and often leads to poor estimators.

1.4 Categorical and multinomial distribution

In the previous section, we have seen how the multivariate normal distribution generalises the univariate normal distribution. In this section, we consider the multivariate generalisations of the Bernoulli and the binomial distribution, which are given by the categorical and the multinomial distributions, respectively.

1.4.1 Categorical distribution

The **categorical distribution** is a generalisation of the Bernoulli distribution and is correspondingly also known as **Multinoulli** distribution.

Assume we have K classes labelled "class 1", "class 2", ..., "class K". A *discrete* random variable with a state space consisting of these K classes has a categorical distribution $\text{Cat}(\boldsymbol{\pi})$. The parameter vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)^T$ specifies the probabilities of each of the K classes with Pr("class k") $= \pi_k$. The parameters satisfy $\pi_k \in [0, 1]$ and $\sum_{k=1}^K \pi_k = 1$, hence there are $K-1$ independent parameters in a categorical distribution (and not K).

Sampling from a categorical distributions $\text{Cat}(\boldsymbol{\pi})$ yields one of K classes. There are several ways to numerically represent "class k", for example simply by the corresponding number k. However, instead of this "integer encoding" it is often more convenient to use the so-called "one hot encoding" where the class is represented by an indicator vector $\boldsymbol{x} = (x_1, \ldots, x_K)^T = (0, 0, \ldots, 1, \ldots, 0)^T$ containing zeros everywhere except for the element $x_k = 1$ at position k. Thus all $x_k \in \{0, 1\}$ and $\sum_{k=1}^K x_k = 1$.

The expectation of $\boldsymbol{x} \sim \text{Cat}(\boldsymbol{\pi})$ is $\text{E}(\boldsymbol{x}) = \boldsymbol{\pi}$, with $\text{E}(x_k) = \pi_k$. The covariance matrix is $\text{Var}(\boldsymbol{x}) = \text{Diag}(\boldsymbol{\pi}) - \boldsymbol{\pi}\boldsymbol{\pi}^T$. In component notation $\text{Var}(x_i) = \pi_i(1 - \pi_i)$ and $\text{Cov}(x_i, x_j) = -\pi_i \pi_j$. This follows directly from the definition of the variance $\text{Var}(\boldsymbol{x}) = \text{E}(\boldsymbol{x}\boldsymbol{x}^T) - \text{E}(\boldsymbol{x})\text{E}(\boldsymbol{x})^T$ and noting that $x_i^2 = x_i$ and $x_i x_j = 0$ if $i \neq j$. Note that the variance matrix $\text{Var}(\boldsymbol{x})$ is singular by construction, as the K random variables $x_1, \ldots, x_K$ are dependent through the constraint $\sum_{k=1}^K x_k = 1$.

The corresponding probability mass function (pmf) can be written conveniently in terms of x_k as

$$f(\boldsymbol{x}) = \prod_{k=1}^K \pi_k^{x_k} = \begin{cases} \pi_k & \text{if } x_k = 1 \end{cases}$$

and the log pmf as

$$\log f(\boldsymbol{x}) = \sum_{k=1}^K x_k \log \pi_k = \begin{cases} \log \pi_k & \text{if } x_k = 1 \end{cases}$$

In order to be more explicit that the categorical distribution has $K - 1$ and not K parameters we rewrite the log-density with $\pi_K = 1 - \sum_{k=1}^{K-1} \pi_k$ and $x_K = 1 - \sum_{k=1}^{K-1} x_k$ as

$$\begin{aligned} \log f(\boldsymbol{x}) &= \sum_{k=1}^{K-1} x_k \log \pi_k + x_K \log \pi_K \\ &= \sum_{k=1}^{K-1} x_k \log \pi_k + \left(1 - \sum_{k=1}^{K-1} x_k\right) \log \left(1 - \sum_{k=1}^{K-1} \pi_k\right) \end{aligned}$$

Note that there is no particular reason to choose π_K as derived, in its place any other of the π_k may be selected.

For $K = 2$ the categorical distribution reduces to the Bernoulli Ber(p) distribution, with $\pi_1 = p$ and $\pi_2 = 1 - p$.

1.4.2 Multinomial distribution

The multinomial distribution arises from repeated categorical sampling, just like the binomial distribution arises from repeated Bernoulli sampling.

1.4.2.1 Univariate case

Binomial distribution:

Repeat Bernoulli Ber(π) experiment n times:

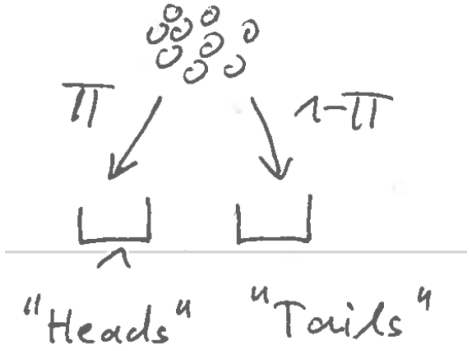
$$x \sim \text{Bin}(n, \pi)$$
$$x \in \{0, \ldots, n\}$$
$$\text{E}(x) = n\,\pi$$
$$\text{Var}(x) = n\,\pi(1 - \pi)$$

Standardised to unit interval:

$$\frac{x}{n} \in \left\{0, \frac{1}{n}, \ldots, 1\right\}$$
$$\text{E}\left(\frac{x}{n}\right) = \pi$$
$$\text{Var}\left(\frac{x}{n}\right) = \frac{\pi(1 - \pi)}{n}$$

Urn model:

distribute n balls into two bins



1.4.2.2 Multivariate case

Multinomial distribution:

Draw n times from categorical distribution Cat($\boldsymbol{\pi}$):

$$\boldsymbol{x} \sim \text{Mult}(n, \boldsymbol{\pi})$$

$$x_i \in \{0, 1, ..., n\}; \sum_{i=1}^{K} x_i = n$$

$$\text{E}(\boldsymbol{x}) = n\,\boldsymbol{\pi}$$

$$\text{Var}(x_i) = n\,\pi_i(1 - \pi_i)$$

$$\text{Cov}(x_i, x_j) = -n\,\pi_i\pi_j$$

Standardised to unit interval:

$$\frac{x_i}{n} \in \left\{0, \frac{1}{n}, \frac{2}{n}, ..., 1\right\}$$

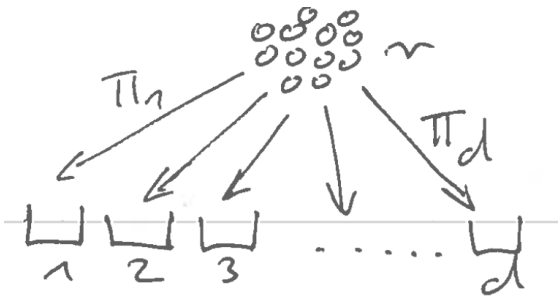
$$\text{E}\left(\frac{\boldsymbol{x}}{n}\right) = \boldsymbol{\pi}$$

$$\text{Var}\left(\frac{x_i}{n}\right) = \frac{\pi_i(1 - \pi_i)}{n}$$

$$\text{Cov}\left(\frac{x_i}{n}, \frac{x_j}{n}\right) = -\frac{\pi_i\pi_j}{n}$$

Urn model:

distribute n balls into K bins:



1.4.3 Entropy and maximum likelihood analysis for the categorical distribution

In the following we compute the KL divergence, the MLE and other related quantities for the categorical distribution.

This generalises the same calculations for the Bernoulli distribution discussed in the earlier module MATH20802 Statistical Methods.

Example 1.1. KL divergence between two categorical distributions with K classes:

With $P = \text{Cat}(\boldsymbol{p})$ and $Q = \text{Cat}(\boldsymbol{q})$ and corresponding probabilities $p_1, \ldots, p_K$ and $q_1, \ldots, q_K$ satisfying $\sum_{i=1}^K p_i = 1$ and $\sum_{i=1}^K q_i = 1$ we get:

$$D_{\text{KL}}(P, Q) = \sum_{i=1}^{K} p_i \log\left(\frac{p_i}{q_i}\right)$$

To be explicit that there are only $K - 1$ parameters in a categorical distribution we can also write

$$D_{\text{KL}}(P, Q) = \sum_{i=1}^{K-1} p_i \log\left(\frac{p_i}{q_i}\right) + p_K \log\left(\frac{p_K}{q_K}\right)$$

with $p_K = \left(1 - \sum_{i=1}^{K-1} p_i\right)$ and $q_K = \left(1 - \sum_{i=1}^{K-1} q_i\right)$.

Example 1.2. Expected Fisher information of the categorical distribution:

We first compute the Hessian matrix $\nabla\nabla^T \log f(\boldsymbol{x})$ of the log-probability mass function, where the differentiation is with regard to $\pi_1, \ldots, \pi_{K-1}$.

The diagonal entries of the Hessian matrix (with $i = 1, \ldots, K - 1$) are

$$\frac{\partial^2}{\partial \pi_i^2} \log f(\boldsymbol{x}) = -\frac{x_i}{\pi_i^2} - \frac{x_K}{\pi_K^2}$$

and its off-diagonal entries are (with $j = 1, \ldots, K - 1$)

$$\frac{\partial^2}{\partial \pi_i \partial \pi_j} \log f(\boldsymbol{x}) = -\frac{x_K}{\pi_K^2}$$

Recalling that $\text{E}(x_i) = \pi_i$ we can compute the expected Fisher information matrix

for a categorical distribution as

$$\begin{aligned}\boldsymbol{I}^{\text{Fisher}}(\pi_1,\ldots,\pi_{K-1}) &= -\text{E}\left(\nabla\nabla^T \log f(\boldsymbol{x})\right) \\ &= \begin{pmatrix} \frac{1}{\pi_1}+\frac{1}{\pi_K} & \cdots & \frac{1}{\pi_K} \\ \vdots & \ddots & \vdots \\ \frac{1}{\pi_K} & \cdots & \frac{1}{\pi_{K-1}}+\frac{1}{\pi_K} \end{pmatrix} \\ &= \text{Diag}\left(\frac{1}{\pi_1},\ldots,\frac{1}{\pi_{K-1}}\right)+\frac{1}{\pi_K}\mathbf{1}\end{aligned}$$

For $K = 2$ and $\pi_1 = p$ this reduces to the expected Fisher information of a Bernoulli variable

$$\begin{aligned}I^{\text{Fisher}}(p) &= \left(\frac{1}{p}+\frac{1}{1-p}\right) \\ &= \frac{1}{p(1-p)}\end{aligned}$$

Example 1.3. Quadratic approximation of KL divergence of the categorical distribution:

The expected Fisher information arises from a local quadratic approximation of the KL divergence:

$$D_{\text{KL}}(F_{\boldsymbol{\theta}}, F_{\boldsymbol{\theta}+\boldsymbol{\varepsilon}}) \approx \frac{1}{2}\boldsymbol{\varepsilon}^T \boldsymbol{I}^{\text{Fisher}}(\boldsymbol{\theta})\boldsymbol{\varepsilon}$$

and

$$D_{\text{KL}}(F_{\boldsymbol{\theta}+\boldsymbol{\varepsilon}}, F_{\boldsymbol{\theta}}) \approx \frac{1}{2}\boldsymbol{\varepsilon}^T \boldsymbol{I}^{\text{Fisher}}(\boldsymbol{\theta})\boldsymbol{\varepsilon}$$

We now consider the KL divergence $D_{\text{KL}}(P, Q)$ between the categorical distribution $P = \text{Cat}(\boldsymbol{p})$ with probabilities $\boldsymbol{p} = (p_1,\ldots,p_K)^T$ with the categorical distribution $Q = \text{Cat}(\boldsymbol{q})$ with probabilities $\boldsymbol{q} = (q_1,\ldots,q_K)^T$.

First, we keep P fixed and assume that Q is a perturbed version of P with $\boldsymbol{q} = \boldsymbol{p} + \boldsymbol{\varepsilon}$. Note that the perturbations $\boldsymbol{\varepsilon} = (\varepsilon_1,\ldots,\varepsilon_K)^T$ satisfy $\sum_{k=1}^K \varepsilon_k = 0$

because $\sum_{k=1}^{K} p_i = 1$ and $\sum_{k=1}^{K} q_i = 1$. Thus $\varepsilon_K = -\sum_{k=1}^{K-1} \varepsilon_k$. Then

$$\begin{aligned}
D_{\mathrm{KL}}(P, Q = P + \varepsilon) &= D_{\mathrm{KL}}(\mathrm{Cat}(\boldsymbol{p}), \mathrm{Cat}(\boldsymbol{p} + \boldsymbol{\varepsilon})) \\
&\approx \frac{1}{2}(\varepsilon_1, \ldots, \varepsilon_{K-1})\, \boldsymbol{I}^{\mathrm{Fisher}}(p_1, \ldots, p_{K-1}) \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_{K-1} \end{pmatrix} \\
&= \frac{1}{2}\left(\sum_{k=1}^{K-1} \frac{\varepsilon_k^2}{p_k} + \frac{\left(\sum_{k=1}^{K-1} \varepsilon_k\right)^2}{p_K} \right) \\
&= \frac{1}{2} \sum_{k=1}^{K} \frac{\varepsilon_k^2}{p_k} \\
&= \frac{1}{2} \sum_{k=1}^{K} \frac{(p_k - q_k)^2}{p_k} \\
&= \frac{1}{2} D_{\mathrm{Neyman}}(P, Q)
\end{aligned}$$

Similarly, if we keep Q fixed and consider P as a disturbed version of Q we get

$$\begin{aligned}
D_{\mathrm{KL}}(P = Q + \varepsilon, Q) &= D_{\mathrm{KL}}(\mathrm{Cat}(\boldsymbol{q} + \boldsymbol{\varepsilon}), \mathrm{Cat}(\boldsymbol{q})) \\
&\approx \frac{1}{2} \sum_{k=1}^{K} \frac{(p_k - q_k)^2}{q_k} \\
&= \frac{1}{2} D_{\mathrm{Pearson}}(P, Q)
\end{aligned}$$

Note that in both approximations we divide by the probabilities of the distribution that is kept fixed.

Note the appearance of the *Pearson χ^2 divergence* and the *Neyman χ^2 divergence* in the above. Both are, like the KL divergence, part of the family of f-divergences. The Neyman χ^2 divergence is also known as the reverse Pearson divergence as $D_{\mathrm{Neyman}}(P, Q) = D_{\mathrm{Pearson}}(Q, P)$.

Example 1.4. Maximum likelihood estimation of the parameters of the categorical distribution:

Maximum likelihood estimation seems trivial at first sight but it is in fact a bit more complicated since there are only $K - 1$ free parameters, and not K. So we either need to optimise with regard to a specific set of $K - 1$ parameters (which is what we do below) or use a constrained optimisation procedure to enforce that $\sum_{k=1}^{K} \pi_k = 1$ (for example by using a Lagrange multiplier).

- The data: We observe n samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. The data matrix of dimension $n \times K$ is $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T = (x_{ik})$. It contains each $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{iK})^T$. The

corresponding summary (minimal sufficient) statistics are $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i = (\bar{x}_1, \ldots, \bar{x}_K)^T$ with $\bar{x}_k = \frac{1}{n}\sum_{i=1}^{n} x_{ik}$. We can also write $\bar{x}_K = 1 - \sum_{k=1}^{K-1} \bar{x}_k$. The number of samples for class k is $n_k = n\bar{x}_k$ with $\sum_{k=1}^{K} n_k = n$.

- The log-likelihood is

$$\begin{aligned} l_n(\pi_1, \ldots, \pi_{K-1}) &= \sum_{i=1}^{n} \log f(\boldsymbol{x}_i) \\ &= \sum_{i=1}^{n} \left(\sum_{k=1}^{K-1} x_{ik} \log \pi_k + \left(1 - \sum_{k=1}^{K-1} x_{ik}\right) \log \left(1 - \sum_{k=1}^{K-1} \pi_k\right) \right) \\ &= n \left(\sum_{k=1}^{K-1} \bar{x}_k \log \pi_k + \left(1 - \sum_{k=1}^{K-1} \bar{x}_k\right) \log \left(1 - \sum_{k=1}^{K-1} \pi_k\right) \right) \\ &= n \left(\sum_{k=1}^{K-1} \bar{x}_k \log \pi_k + \bar{x}_K \log \pi_K \right) \end{aligned}$$

- Score function (gradient)

$$\begin{aligned} \boldsymbol{S}_n(\pi_1, \ldots, \pi_{K-1}) &= \nabla l_n(\pi_1, \ldots, \pi_{K-1}) \\ &= \begin{pmatrix} \frac{\partial}{\partial \pi_1} l_n(\pi_1, \ldots, \pi_{K-1}) \\ \vdots \\ \frac{\partial}{\partial \pi_{K-1}} l_n(\pi_1, \ldots, \pi_{K-1}) \end{pmatrix} \\ &= n \begin{pmatrix} \frac{\bar{x}_1}{\pi_1} - \frac{\bar{x}_K}{\pi_K} \\ \vdots \\ \frac{\bar{x}_{K-1}}{\pi_{K-1}} - \frac{\bar{x}_K}{\pi_K} \end{pmatrix} \end{aligned}$$

  Note in particular the need for the second term that arises because π_K depends on all the $\pi_1, \ldots, \pi_{K-1}$.

- Maximum likelihood estimate: Setting $\boldsymbol{S}_n(\hat{\pi}_1^{ML}, \ldots, \hat{\pi}_{K-1}^{ML}) = 0$ yields $K-1$ equations

$$\bar{x}_i \left(1 - \sum_{k=1}^{K-1} \hat{\pi}_k^{ML}\right) = \hat{\pi}_i^{ML} \left(1 - \sum_{k=1}^{K-1} \bar{x}_k\right)$$

  for $i = 1, \ldots, K-1$ and with solution

$$\hat{\pi}_i^{ML} = \bar{x}_i$$

  It also follows that

$$\hat{\pi}_K^{ML} = 1 - \sum_{k=1}^{K-1} \hat{\pi}_k^{ML} = 1 - \sum_{k=1}^{K-1} \bar{x}_k = \bar{x}_K$$

The maximum likelihood estimator is therefore the frequency of of the occurance of a class among the n samples.

Example 1.5. Observed Fisher information of the categorical distribution:

We first need to compute the negative Hessian matrix of the log likelihood function $-\nabla\nabla^T l_n(\pi_1, \ldots, \pi_{K-1})$ and then evaluate it at the MLEs $\hat{\pi}_1^{ML}, \ldots, \hat{\pi}_{K-1}^{ML}$.

The diagonal entries of the Hessian matrix (with $i = 1, \ldots, K-1$) are

$$\frac{\partial^2}{\partial \pi_i^2} l_n(\pi_1, \ldots, \pi_{K-1}) = -n\left(\frac{\bar{x}_i}{\pi_i^2} + \frac{\bar{x}_K}{\pi_K^2}\right)$$

and its off-diagonal entries are (with $j = 1, \ldots, K-1$)

$$\frac{\partial^2}{\partial \pi_i \partial \pi_j} l_n(\pi_1, \ldots, \pi_{K-1}) = -\frac{n\bar{x}_K}{\pi_K^2}$$

Thus, the observed Fisher information matrix at the MLE for a categorical distribution is

$$\boldsymbol{J}_n(\hat{\pi}_1^{ML}, \ldots, \hat{\pi}_{K-1}^{ML}) = n\begin{pmatrix} \frac{1}{\hat{\pi}_1^{ML}} + \frac{1}{\hat{\pi}_K^{ML}} & \cdots & \frac{1}{\hat{\pi}_K^{ML}} \\ \vdots & \ddots & \vdots \\ \frac{1}{\hat{\pi}_K^{ML}} & \cdots & \frac{1}{\hat{\pi}_{K-1}^{ML}} + \frac{1}{\hat{\pi}_K^{ML}} \end{pmatrix}$$

For $K = 2$ this reduces to the observed Fisher information of a Bernoulli variable

$$\begin{aligned} J_n(\hat{p}_{ML}) &= n\left(\frac{1}{\hat{p}_{ML}} + \frac{1}{1-\hat{p}_{ML}}\right) \\ &= \frac{n}{\hat{p}_{ML}(1-\hat{p}_{ML})} \end{aligned}$$

The inverse of the observed Fisher information is:

$$\boldsymbol{J}_n(\hat{\pi}_1^{ML}, \ldots, \hat{\pi}_{K-1}^{ML})^{-1} = \frac{1}{n}\begin{pmatrix} \hat{\pi}_1^{ML}(1-\hat{\pi}_1^{ML}) & \cdots & -\hat{\pi}_1^{ML}\hat{\pi}_{K-1}^{ML} \\ \vdots & \ddots & \vdots \\ -\hat{\pi}_{K-1}^{ML}\hat{\pi}_1^{ML} & \cdots & \hat{\pi}_{K-1}^{ML}(1-\hat{\pi}_{K-1}^{ML}) \end{pmatrix}$$

To show that this is indeed the inverse we use the Woodbury matrix identity (see Appendix)

$$(\boldsymbol{A} + \boldsymbol{UBV})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{B}^{-1} + \boldsymbol{VA}^{-1}\boldsymbol{U})^{-1}\boldsymbol{VA}^{-1}$$

with

- $B = 1$,

- $\boldsymbol{u} = (\pi_1, \ldots, \pi_{K-1})^T$,
- $\boldsymbol{v} = -\boldsymbol{u}^T$,
- $\boldsymbol{A} = \text{Diag}(\boldsymbol{u})$ and its inverse $\boldsymbol{A}^{-1} = \text{Diag}(\pi_1^{-1}, \ldots, \pi_{K-1}^{-1})$.

Then $\boldsymbol{A}^{-1}\boldsymbol{u} = \mathbf{1}_{K-1}$ and $1 - \boldsymbol{u}^T\boldsymbol{A}^{-1}\boldsymbol{u} = \pi_K$. With this

$$\boldsymbol{J}_n(\hat{\pi}_1^{ML}, \ldots, \hat{\pi}_{K-1}^{ML})^{-1} = \frac{1}{n}\left(\boldsymbol{A} - \boldsymbol{u}\boldsymbol{u}^T\right)$$

and

$$\boldsymbol{J}_n(\hat{\pi}_1^{ML}, \ldots, \hat{\pi}_{K-1}^{ML}) = n\left(\boldsymbol{A}^{-1} + \frac{1}{\pi_K}\mathbf{1}_{K-1\times K-1}\right)$$

For $K = 2$ the inverse observed Fisher information of the categorical distribution reduces to that of the Bernoulli distribution

$$J_n(\hat{p}_{ML})^{-1} = \frac{\hat{p}_{ML}(1 - \hat{p}_{ML})}{n}$$

The inverse observed Fisher information is useful, e.g., as the asymptotic variance of the maximum likelihood estimate.

Example 1.6. Wald statistic for the categorical distribution:

The squared Wald statistic is

$$\begin{aligned}
t(\boldsymbol{p}_0)^2 &= (\hat{\pi}_1^{ML} - p_1^0, \ldots, \hat{\pi}_{K-1}^{ML} - p_{K-1}^0)\boldsymbol{J}_n(\hat{\pi}_1^{ML}, \ldots, \hat{\pi}_{K-1}^{ML})\begin{pmatrix} \hat{\pi}_1^{ML} - p_1^0 \\ \vdots \\ \hat{\pi}_{K-1}^{ML} - p_{K-1}^0 \end{pmatrix} \\
&= n\left(\sum_{k=1}^{K-1}\frac{(\hat{\pi}_k^{ML} - p_k^0)^2}{\hat{\pi}_k^{ML}} + \frac{\left(\sum_{k=1}^{K-1}(\hat{\pi}_k^{ML} - p_k^0)\right)^2}{\hat{\pi}_K^{ML}}\right) \\
&= n\left(\sum_{k=1}^{K}\frac{(\hat{\pi}_k^{ML} - p_k^0)^2}{\hat{\pi}_k^{ML}}\right) \\
&= nD_{\text{Neyman}}(\text{Cat}(\hat{\boldsymbol{\pi}}_{ML}), \text{Cat}(\boldsymbol{p}_0))
\end{aligned}$$

With $n_1, \ldots, n_K$ the observed counts with $n = \sum_{k=1}^K n_k$ and $\hat{\pi}_k^{ML} = \frac{n_k}{n} = \bar{x}_k$, and $n_1^{\text{expect}}, \ldots, n_K^{\text{expect}}$ the expected counts $n_k^{\text{expect}} = np_k^0$ under $\boldsymbol{p}_0$ we can write the squared Wald statistic as follows:

$$t(\boldsymbol{p}_0)^2 = \sum_{k=1}^{K}\frac{(n_k - n_k^{\text{expect}})^2}{n_k} = \chi^2_{\text{Neyman}}$$

This is known as the Neyman chi-squared statistic (note the *observed* counts in its denominator) and it is asymptotically distributed as χ^2_{K-1} because there are $K - 1$ free parameters in $\boldsymbol{p}_0$.

Example 1.7. Wilks log-likelihood ratio statistic for the categorical distribution:

The Wilks log-likelihood ratio is

$$W(\boldsymbol{p}_0) = 2(l_n(\hat{\pi}_1^{ML}, \dots, \hat{\pi}_{K-1}^{ML}) - l_n(p_1^0, \dots, p_{K-1}^0))$$

with $\boldsymbol{p}_0 = c(p_1^0, \dots, p_K^0)^T$. As the probabilities sum up to 1 there are only $K-1$ free parameters.

The log-likelihood at the MLE is

$$l_n(\hat{\pi}_1^{ML}, \dots, \hat{\pi}_{K-1}^{ML}) = n \sum_{k=1}^{K} \bar{x}_k \log \hat{\pi}_k^{ML} = n \sum_{k=1}^{K} \bar{x}_k \log \bar{x}_k$$

with $\hat{\pi}_k^{ML} = \frac{n_k}{n} = \bar{x}_k$. Note that here and in the following the sums run from 1 to K where the K-th component is always computed from the components 1 to $K-1$, as in the previous section. The log-likelihood at $\boldsymbol{p}_0$ is

$$l_n(p_1^0, \dots, p_{K-1}^0) = n \sum_{k=1}^{K} \bar{x}_k \log p_k^0$$

so that the Wilks statistic becomes

$$W(\boldsymbol{p}_0) = 2n \sum_{k=1}^{K} \bar{x}_k \log \left(\frac{\bar{x}_k}{p_k^0} \right)$$

It is asymptotically chi-squared distributed with $K-1$ degrees of freedom.

Note that for this model the Wilks statistic is equal to the KL Divergence

$$W(\boldsymbol{p}_0) = 2n D_{\mathrm{KL}}(\mathrm{Cat}(\hat{\boldsymbol{\pi}}_{ML}), \mathrm{Cat}(\boldsymbol{p}_0))$$

The Wilks log-likelihood ratio statistic for the categorical distribution is also known as the G test statistic where $\hat{\boldsymbol{\pi}}_{ML}$ corresponds to the observed frequencies (as observed in data) and $\boldsymbol{p}_0$ are the expected frequencies (i.e. hypothesised to be the true frequencies).

Using observed counts n_k and expected counts $n_k^{\text{expect}} = np_k^0$ we can write the Wilks statistic respectively the G-statistic as follows:

$$W(\boldsymbol{p}_0) = 2 \sum_{k=1}^{K} n_k \log \left(\frac{n_k}{n_k^{\text{expect}}} \right)$$

Example 1.8. Quadratic approximation of the Wilks log-likelihood ratio statistic for the categorical distribution:

Developing the Wilks statistic $W(\boldsymbol{p}_0)$ around the MLE $\hat{\boldsymbol{\pi}}_{ML}$ yields the squared Wald statistic which for the categorical distribution is the Neyman chi-squared statistic:

$$\begin{aligned} W(\boldsymbol{p}_0) &= 2nD_{\mathrm{KL}}(\mathrm{Cat}(\hat{\boldsymbol{\pi}}_{ML}), \mathrm{Cat}(\boldsymbol{p}_0)) \\ &\approx nD_{\mathrm{Neyman}}(\mathrm{Cat}(\hat{\boldsymbol{\pi}}_{ML}), \mathrm{Cat}(\boldsymbol{p}_0)) \\ &= \sum_{k=1}^{K} \frac{(n_k - n_k^{\mathrm{expect}})^2}{n_k} \\ &= \chi^2_{\mathrm{Neyman}} \end{aligned}$$

If instead we approximate the KL divergence assuming $\boldsymbol{p}_0$ as fixed we arrive at

$$\begin{aligned} 2nD_{\mathrm{KL}}(\mathrm{Cat}(\hat{\boldsymbol{\pi}}_{ML}), \mathrm{Cat}(\boldsymbol{p}_0)) &\approx nD_{\mathrm{Pearson}}(\mathrm{Cat}(\hat{\boldsymbol{\pi}}_{ML}), \mathrm{Cat}(\boldsymbol{p}_0)) \\ &= \sum_{k=1}^{K} \frac{(n_k - n_k^{\mathrm{expect}})^2}{n_k^{\mathrm{expect}}} \\ &= \chi^2_{\mathrm{Pearson}} \end{aligned}$$

which is the well-known Pearson chi-squared statistic (note the *expected* counts in its denominator).

1.5 Further multivariate distributions

In the following we describe further multivariate distributions. Specifically, we discuss properties of the

- the Dirichlet distribution (the generalisation of the Beta distribution),
- the Wishart distribution (the generalisation of the Gamma distribution, also known as scaled χ^2 distribution), and of
- the inverse Wishart distribution (the generalisation of the inverse Gamma distribution).

1.5.1 Dirichlet distribution

1.5.1.1 Univariate case

Beta distribution

$$\begin{aligned} x &\sim \mathrm{Beta}(\alpha, \beta) \\ x &\in [0, 1] \\ \alpha &> 0; \beta > 0 \\ m &= \alpha + \beta \end{aligned}$$

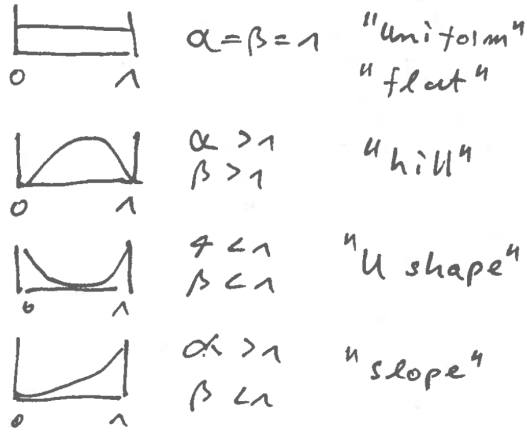
$$\mu = \frac{\alpha}{m} \in [0, 1]$$

$$\mathrm{E}(x) = \mu$$

$$\mathrm{Var}(x) = \frac{\mu(1-\mu)}{m+1}$$

compare with unit standardised binomial!

Different shapes



Useful as distribution for a proportion π

Bayesian Model:

Beta prior: $\pi \sim \mathrm{Beta}(\alpha, \beta)$

Binomial likelihood: $x|\pi \sim \mathrm{Bin}(n, \pi)$

1.5.1.2 Multivariate case

Dirichlet distribution

$$\boldsymbol{x} \sim \mathrm{Dir}(\boldsymbol{\alpha})$$

$$x_i \in [0, 1]; \sum_{i=1}^{d} x_i = 1$$

$$\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_d)^T > 0$$

$$m = \sum_{i=1}^{d} \alpha_i$$

$$\mu_i = \frac{\alpha_i}{m} \in [0, 1]$$

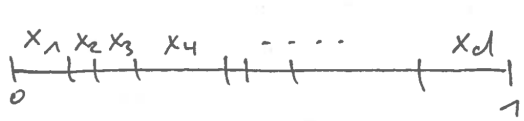
$$\mathrm{E}(x_i) = \mu_i$$

$$\mathrm{Var}(x_i) = \frac{\mu_i(1 - \mu_i)}{m + 1}$$

$$\mathrm{Cov}(x_i, x_j) = -\frac{\mu_i \mu_j}{m + 1}$$

compare with unit standardised multinomial!

Stick breaking" model



Useful as distribution for a proportion $\boldsymbol{\pi}$

Bayesian Model:

Dirichlet prior: $\boldsymbol{\pi} \sim \mathrm{Dir}(\boldsymbol{\alpha})$

Multinomial likelihood: $\boldsymbol{x}|\boldsymbol{\pi} \sim \mathrm{Mult}(n, \boldsymbol{\pi})$

1.5.2 Wishart distribution

The Wishart distribution is a multivariate generalisation of the gamma distribution, also known as univariate Wishart distribution and scaled chi-squared distribution. The exponential and chi-squared distribution are special cases.

1.5.2.1 Univariate case

Gamma distribution:

$$z_1, z_2, \ldots, z_m \overset{\text{iid}}{\sim} N(0, \sigma_z^2)$$

$$x = \sum_{i=1}^{m} z_i^2$$

$$\mu_x = m\sigma_z^2$$

Then x is distributed as:

$$x \sim \mathrm{W}_1\left(\frac{\mu_x}{m}, m\right) = \mathrm{Gam}\left(\alpha = \frac{1}{2}m, \beta = 2\frac{\mu_x}{m}\right) = \frac{\mu_x}{m}\chi^2_m$$

where α is the shape and β the scale parameter of the gamma distribution.

The mean and variance of x are:

$$\mathrm{E}(x) = \mu_x$$

$$\mathrm{Var}(x) = \frac{2\mu_x^2}{m}$$

Useful as the distribution of sample variance:

$$y_1, \ldots, y_n \sim N(\mu_y, \sigma^2)$$

Known mean μ_y:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \mu_y)^2 \sim \mathrm{W}_1\left(\frac{\sigma^2}{n}, n\right)$$

Unknown mean μ_y (estimated by $\bar{y}$):

$$\widehat{\sigma^2}_{ML} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 \sim \mathrm{W}_1\left(\frac{\sigma^2}{n}, n-1\right)$$

$$\widehat{\sigma^2}_{UB} = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2 \sim \mathrm{W}_1\left(\frac{\sigma^2}{n-1}, n-1\right)$$

1.5.2.2 Multivariate case

Wishart distribution:

$$\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_m \overset{\text{iid}}{\sim} N_d(0, \boldsymbol{\Sigma}_z)$$

$$\underbrace{\boldsymbol{X}}_{d\times d} = \sum_{i=1}^{m} \underbrace{\boldsymbol{z}_i \boldsymbol{z}_i^T}_{d\times d}$$

$$\boldsymbol{M} = (\mu_{ij}) = m\boldsymbol{\Sigma}_{\boldsymbol{z}}$$

Then $\boldsymbol{X}$ (a random matrix!) is distributed as:

$$\boldsymbol{X} \sim \mathrm{W}_d\left(\frac{\boldsymbol{M}}{m}, m\right)$$

with mean and variances:

$$\mathrm{E}(\boldsymbol{X}) = \boldsymbol{M}$$

$$\mathrm{Var}(x_{ij}) = \frac{\mu_{ij}^2 + \mu_{ii}\mu_{jj}}{m}$$

Useful as distribution of sample covariance:

$$\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n \sim N_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y}_i - \boldsymbol{\mu})(\boldsymbol{y}_i - \boldsymbol{\mu})^T \sim \mathrm{W}_d\left(\boldsymbol{\Sigma}/n, n\right)$$

$$\widehat{\boldsymbol{\Sigma}}_{ML} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y}_i - \bar{\boldsymbol{y}})(\boldsymbol{y}_i - \bar{\boldsymbol{y}})^T \sim \mathrm{W}_d\left(\boldsymbol{\Sigma}/n, n-1\right)$$

$$\widehat{\boldsymbol{\Sigma}}_{UB} = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{y}_i - \bar{\boldsymbol{y}})(\boldsymbol{y}_i - \bar{\boldsymbol{y}})^T \sim \mathrm{W}_d\left(\boldsymbol{\Sigma}/(n-1), n-1\right)$$

1.5.3 Inverse Wishart distribution

The inverse Wishart distribution is a multivariate generalisation of the inverse gamma distribution (also known as inverse scaled chi-squared distribution).

1.5.3.1 Univariate case

Inverse gamma distribution (with α shape and β scale parameter):

$$x \sim \mathrm{W}_1^{-1}(k\mu, k+2) = \text{Inv-Gam}\left(\alpha = \frac{k+2}{2}, \beta = \frac{k\mu}{2}\right) = k\mu\,\text{Inv-}\chi^2_{k+2}$$

Then x has mean and variance

$$\mathrm{E}(x) = \mu$$

$$\mathrm{Var}(x) = \frac{2\mu^2}{k-2}$$

Relationship to gamma distribution:

$$\frac{1}{x} \sim W_1\left(\frac{1}{k\mu}, k+2\right) = \mathrm{Gam}\left(\frac{k+2}{2}, \frac{2}{k\mu}\right) = \frac{1}{k\mu}\,\chi^2_{k+2}$$

1.5.3.2 Multivariate case

Inverse Wishart distribution:

$$\underbrace{\boldsymbol{X}}_{d\times d} \sim \mathrm{W}_d^{-1}\left(k\ \underbrace{\boldsymbol{M}}_{d\times d},\ k+d+1\right)$$

$$\mathrm{E}(\boldsymbol{X}) = \boldsymbol{M}$$

$$\mathrm{Var}(x_{ij}) = \frac{2}{k-2}\frac{(k+2)\mu_{ij}^2 + k\,\mu_{ii}\mu_{jj}}{2k+2}$$

Relationship to Wishart:

$$\boldsymbol{X}^{-1} \sim \mathrm{W}_d\left(\frac{\boldsymbol{M}^{-1}}{k},\ k+d+1\right)$$

The inverse Wishart distribution is useful as prior and posterior distribution of the variance where k is the sample size parameter and $\boldsymbol{M}$ resp. μ is mean of the distribution for $\boldsymbol{\Sigma}$ and σ^2 (in the univariate case).

1.5.4 Further distributions

https://en.wikipedia.org/wiki/List_of_probability_distributions

Wikipedia is a good source for information about distributions.

Chapter 2

Transformations and dimension reduction

Motivation: In the following we study transformations of random vectors and their distributions. These transformation are very important since they either transform simple distributions into more complex distributions or allow to simplify complex models. In machine learning invertible mappings of transformations for probability distributions are known as "normalising flows".

2.1 Linear Transformations

2.1.1 Location-scale transformation

Also known as affine transformation.

$$\boldsymbol{y} = \underbrace{\boldsymbol{a}}_{\text{location parameter}} + \underbrace{\boldsymbol{B}}_{\text{scale parameter}} \boldsymbol{x}$$

$\boldsymbol{y}$: $m \times 1$ random vector

$\boldsymbol{a}$: $m \times 1$ vector, location parameter

$\boldsymbol{B}$: $m \times d$ matrix, scale parameter , $m \geq 1$

$\boldsymbol{x}$: $d \times 1$ random vector

Mean and variance of the original vector $\boldsymbol{x}$:

$$\mathrm{E}(\boldsymbol{x}) = \boldsymbol{\mu}_{\boldsymbol{x}}$$
$$\mathrm{Var}(\boldsymbol{x}) = \boldsymbol{\Sigma}_{\boldsymbol{x}}$$

Mean and variance of the transformed random vector $\boldsymbol{y}$:

$$\mathrm{E}(\boldsymbol{y}) = \boldsymbol{a} + \boldsymbol{B}\boldsymbol{\mu}_{\boldsymbol{x}}$$
$$\mathrm{Var}(\boldsymbol{y}) = \boldsymbol{B}\boldsymbol{\Sigma}_{\boldsymbol{x}}\boldsymbol{B}^T$$

Cross-covariance $\boldsymbol{\Phi} = \Sigma_{xy} = \mathrm{Cov}(\boldsymbol{x}, \boldsymbol{y})$ between $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\boldsymbol{\Phi} = \mathrm{Cov}(\boldsymbol{x}, \boldsymbol{B}\boldsymbol{x}) = \boldsymbol{\Sigma}_{\boldsymbol{x}}\boldsymbol{B}^T$$

Note that $\boldsymbol{\Phi}$ is a matrix of dimensions $d \times m$ as the dimension of $\boldsymbol{x}$ is d and the dimension of $\boldsymbol{y}$ is m.

Cross-correlation $\boldsymbol{\Psi} = \boldsymbol{P}_{xy} = \mathrm{Cor}(\boldsymbol{x}, \boldsymbol{y})$ between $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\boldsymbol{\Psi} = \boldsymbol{V}_{\boldsymbol{x}}^{-1/2}\boldsymbol{\Phi}\boldsymbol{V}_{\boldsymbol{y}}^{-1/2}$$

where $\boldsymbol{V}_{\boldsymbol{x}} = \mathrm{Diag}(\boldsymbol{\Sigma}_{\boldsymbol{x}})$ and $\boldsymbol{V}_{\boldsymbol{y}} = \mathrm{Diag}(\boldsymbol{B}\boldsymbol{\Sigma}_{\boldsymbol{x}}\boldsymbol{B}^T)$ are diagonal matrices containing the variances for the components of $\boldsymbol{x}$ and $\boldsymbol{y}$. The dimensions of the matrix $\boldsymbol{\Psi}$ are also $d \times m$.

Special cases/examples:

Example 2.1. Univariate case ($d = 1, m = 1$): $y = a + bx$

- $\mathrm{E}(y) = a + b\mu$
- $\mathrm{Var}(y) = b^2\sigma^2$
- $\mathrm{Cov}(y, x) = b\sigma^2$
- $\mathrm{Cor}(y, x) = \frac{b\sigma^2}{\sqrt{b^2\sigma^2}\sqrt{\sigma^2}} = 1$

Example 2.2. Sum of two random univariate variables: $y = x_1 + x_2$, i.e. $a = 0$ and $\boldsymbol{B} = (1, 1)$

- $\mathrm{E}(y) = \mathrm{E}(x_1 + x_2) = \mu_1 + \mu_2$
- $\mathrm{Var}(y) = \mathrm{Var}(x_1 + x_2) = (1,1)\begin{pmatrix}\sigma_1^2 & \sigma_{12}\\ \sigma_{12} & \sigma_2^2\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix} = \sigma_1^2 + \sigma_2^2 + 2\sigma_{12} = \mathrm{Var}(x_1) + \mathrm{Var}(x_2) + 2\,\mathrm{Cov}(x_1, x_2)$

Example 2.3. $y_1 = a_1 + b_1x_1$ and $y_2 = a_2 + b_2x_2$, i.e. $\boldsymbol{a} = \begin{pmatrix}a_1\\a_2\end{pmatrix}$ and $\boldsymbol{B} = \begin{pmatrix}b_1 & 0\\0 & b_2\end{pmatrix}$

- $\mathrm{E}(\boldsymbol{y}) = \begin{pmatrix}a_1\\a_2\end{pmatrix} + \begin{pmatrix}b_1 & 0\\0 & b_2\end{pmatrix}\begin{pmatrix}\mu_1\\\mu_2\end{pmatrix} = \begin{pmatrix}a_1 + b_1\mu_1\\a_2 + b_2\mu_2\end{pmatrix}$
- $\mathrm{Var}(\boldsymbol{y}) = \begin{pmatrix}b_1 & 0\\0 & b_2\end{pmatrix}\begin{pmatrix}\sigma_1^2 & \sigma_{12}\\\sigma_{12} & \sigma_2^2\end{pmatrix}\begin{pmatrix}b_1 & 0\\0 & b_2\end{pmatrix} = \begin{pmatrix}b_1^2\sigma_1^2 & b_1b_2\sigma_{12}\\b_1b_2\sigma_{12} & b_2^2\sigma_2^2\end{pmatrix}$
  note that $\mathrm{Cov}(y_1, y_2) = b_1b_2\mathrm{Cov}(x_1, x_2)$

2.1.2 Squared multiple correlation

Definition of squared multiple correlation

Squared multiple correlation $\text{MCor}(y, \boldsymbol{x})^2$ is a scalar measure summarising the linear association between a scalar response variable y and a set of predictors $\boldsymbol{x} = (x_1, \ldots, x_d)^T$. It is defined as

$$\begin{aligned}\text{MCor}(y, \boldsymbol{x})^2 &= \boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_x^{-1}\boldsymbol{\Sigma}_{xy}/\sigma_y^2 \\ &= \boldsymbol{P}_{yx}\boldsymbol{P}_x^{-1}\boldsymbol{P}_{xy}\end{aligned}$$

If y can be perfectly linearly predicted by $\boldsymbol{x}$ then $\text{MCor}(y, \boldsymbol{x})^2 = 1$.

The empirical estimate of $\text{MCor}(y, \boldsymbol{x})^2$ is the R^2 coefficient that you will find in any software for linear regression.

See the corresponding section in MATH20802 Statistical Methods.

Squared multiple correlation for affine transformation

Since we linearly transform $\boldsymbol{x}$ into $\boldsymbol{y}$ with no additional error involved we expect that for each component y_i in $\boldsymbol{y}$ we have $\text{MCor}(y_i, \boldsymbol{x})^2 = 1$. This can be shown directly by computing

$$\begin{aligned}\left(\text{MCor}(y_1, \boldsymbol{x})^2, \ldots, \text{MCor}(y_m, \boldsymbol{x})^2\right)^T &= \text{Diag}\left(\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_x^{-1}\boldsymbol{\Sigma}_{xy}\right)/\text{Diag}\left(\boldsymbol{\Sigma}_y\right) \\ &= \text{Diag}\left(\boldsymbol{B}\boldsymbol{\Sigma}_x\boldsymbol{\Sigma}_x^{-1}\boldsymbol{\Sigma}_x\boldsymbol{B}^T\right)/\text{Diag}\left(\boldsymbol{B}\boldsymbol{\Sigma}_x\boldsymbol{B}^T\right) \\ &= \text{Diag}\left(\boldsymbol{B}\boldsymbol{\Sigma}_x\boldsymbol{B}^T\right)/\text{Diag}\left(\boldsymbol{B}\boldsymbol{\Sigma}_x\boldsymbol{B}^T\right) \\ &= (1, \ldots, 1)^T\end{aligned}$$

2.1.3 Invertible location-scale transformation

If $m = d$ (square $\boldsymbol{B}$) and $\det(\boldsymbol{B}) \neq 0$ then the affine transformation is **invertible**.

Forward transformation:

$$\boldsymbol{y} = \boldsymbol{a} + \boldsymbol{B}\boldsymbol{x}$$

Back transformation:

$$\boldsymbol{x} = \boldsymbol{B}^{-1}(\boldsymbol{y} - \boldsymbol{a})$$

Invertible transformations thus provide a one-to-one map between $\boldsymbol{x}$ and $\boldsymbol{y}$.

Example 2.4. Mahalanobis transform

We assume $\text{E}(\boldsymbol{x}) = \boldsymbol{\mu}_x$ and a positive definite covariance matrix $\text{Var}(\boldsymbol{x}) = \boldsymbol{\Sigma}_x$ with $\det(\boldsymbol{\Sigma}_x) > 0$.

The Mahalanobis transformation is given by

$$\boldsymbol{y} = \boldsymbol{\Sigma}_x^{-1/2}(\boldsymbol{x} - \boldsymbol{\mu}_x)$$

This corresponds to an affine transformation with $\boldsymbol{a} = -\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1/2}\boldsymbol{\mu}_{\boldsymbol{x}}$ and $\boldsymbol{B} = \boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1/2}$.

The inverse principal matrix square root $\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1/2}$ can be computed by eigendecomposition (see Appendix).

The mean and the variance of $\boldsymbol{y}$ becomes

$$\mathrm{E}(\boldsymbol{y}) = \boldsymbol{0}$$

and

$$\mathrm{Var}(\boldsymbol{y}) = \boldsymbol{I}_d$$

.

The Mahalanobis transforms performs three functions:

1. Centering ($-\boldsymbol{\mu}$)
2. Standardisation $\mathrm{Var}(y_i) = 1$
3. Decorrelation $\mathrm{Cor}(y_i, y_j) = 0$ for $i \neq j$

In the **univariate case ($d = 1$)** the coefficients reduce to $a = -\frac{\mu_x}{\sigma_x}$ and $B = \frac{1}{\sigma_x}$ and the Mahalanobis transform becomes

$$y = \frac{x - \mu_x}{\sigma_x}$$

i.e. it applies centering + standardisation.

The **Mahalanobis transformation** appears implicitly in many places in multivariate statistics, e.g. in the multivariate normal density. It is a particular example of a whitening transformation (of which there are infinitely many, see later in the course).

Example 2.5. Inverse Mahalanobis transformation

The inverse of the Mahalanobis transform is given by

$$\boldsymbol{y} = \boldsymbol{\mu}_{\boldsymbol{y}} + \boldsymbol{\Sigma}_{\boldsymbol{y}}^{1/2}\boldsymbol{x}$$

As the Mahalanobis transform is a whitening transform the inverse Mahalonobis transform is sometimes called the Mahalanobis colouring transformation. The coefficients in the affine transformation are $\boldsymbol{a} = \boldsymbol{\mu}_{\boldsymbol{y}}$ and $\boldsymbol{B} = \boldsymbol{\Sigma}_{\boldsymbol{y}}^{1/2}$.

Starting with $\mathrm{E}(\boldsymbol{x}) = \boldsymbol{0}$ and $\mathrm{Var}(\boldsymbol{x}) = \boldsymbol{I}_d$ the mean and variance of the transformed variable are

$$\mathrm{E}(\boldsymbol{y}) = \boldsymbol{\mu}_{\boldsymbol{y}}$$

and

$$\mathrm{Var}(\boldsymbol{y}) = \boldsymbol{\Sigma}_{\boldsymbol{y}}$$

2.1.4 Transformation of a density under an invertible location-scale transformation:

Assume $\boldsymbol{x} \sim F_{\boldsymbol{x}}$ with density $f_{\boldsymbol{x}}(\boldsymbol{x})$.

After linear transformation $\boldsymbol{y} = \boldsymbol{a} + \boldsymbol{B}\boldsymbol{x}$ we get $\boldsymbol{y} \sim F_{\boldsymbol{y}}$ with density

$$f_{\boldsymbol{y}}(\boldsymbol{y}) = |\det(\boldsymbol{B})|^{-1} f_{\boldsymbol{x}}\left(\boldsymbol{B}^{-1}(\boldsymbol{y} - \boldsymbol{a})\right)$$

Example 2.6. Transformation of standard normal with inverse Mahalanobis transform

Assume $\boldsymbol{x}$ is multivariate standard normal $\boldsymbol{x} \sim N_d(\boldsymbol{0}, \boldsymbol{I}_d)$ with density

$$f_{\boldsymbol{x}}(\boldsymbol{x}) = (2\pi)^{-d/2} \exp\left(-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{x}\right)$$

Then the density after applying the inverse Mahalanobis transform $\boldsymbol{y} = \boldsymbol{\mu}_{\boldsymbol{y}} + \boldsymbol{\Sigma}_{\boldsymbol{y}}^{1/2}\boldsymbol{x}$ is

$$\begin{aligned} f_{\boldsymbol{y}}(\boldsymbol{y}) &= |\det(\boldsymbol{\Sigma}_{\boldsymbol{y}}^{1/2})|^{-1}(2\pi)^{-d/2} \exp\left(-\frac{1}{2}(\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}})^T \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1/2} \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1/2}(\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}})\right) \\ &= (2\pi)^{-d/2} \det(\boldsymbol{\Sigma}_{\boldsymbol{y}})^{-1/2} \exp\left(-\frac{1}{2}(\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}})^T \boldsymbol{\Sigma}_{\boldsymbol{y}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_{\boldsymbol{y}})\right) \end{aligned}$$

$\Longrightarrow \boldsymbol{y}$ has multivariate normal density $N_d(\boldsymbol{\mu}_{\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{y}})$

Application: e.g. random number generation: draw from $N_d(\boldsymbol{0}, \boldsymbol{I}_d)$ (easy!) then convert to multivariate normal by tranformation (see Worksheet 4).

2.2 Nonlinear transformations

2.2.1 General transformation

$$\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x})$$

with $\boldsymbol{h}$ an arbitrary vector-valued function

- linear case: $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{a} + \boldsymbol{B}\boldsymbol{x}$

2.2.2 Delta method

Assume that we know the mean $\mathrm{E}(\boldsymbol{x}) = \boldsymbol{\mu}_{\boldsymbol{x}}$ and variance $\mathrm{Var}(\boldsymbol{x}) = \boldsymbol{\Sigma}_{\boldsymbol{x}}$ of $\boldsymbol{x}$. Is it possible to say something about the mean and variance of the transformed random variable $\boldsymbol{y}$?

$$\mathrm{E}(\boldsymbol{y}) = \mathrm{E}(\boldsymbol{h}(\boldsymbol{x})) = ?$$

$$\mathrm{Var}(\boldsymbol{y}) = \mathrm{Var}(\boldsymbol{h}(\boldsymbol{x})) = ?$$

In general, for a transformation $\boldsymbol{h}(\boldsymbol{x})$ the exact mean and variance of the transformed variable cannot be obtained analytically.

However, we can find a **linear approximation** and then compute its mean and variance. This approximation is called the "Delta Method", or the "law of propagation of errors", and is credited to Gauss.[1]

Linearisation of $\boldsymbol{h}(\boldsymbol{x})$ is achieved by a Taylor series approximation of first order of $\boldsymbol{h}(\boldsymbol{x})$ around $\boldsymbol{x}_0$:

$$\boldsymbol{h}(\boldsymbol{x}) \approx \boldsymbol{h}(\boldsymbol{x}_0) + \underbrace{J_{\boldsymbol{h}}(\boldsymbol{x}_0)}_{\text{Jacobian matrix}} (\boldsymbol{x} - \boldsymbol{x}_0) = \underbrace{\boldsymbol{h}(\boldsymbol{x}_0) - J_{\boldsymbol{h}}(\boldsymbol{x}_0)\,\boldsymbol{x}_0}_{\boldsymbol{a}} + \underbrace{J_{\boldsymbol{h}}(\boldsymbol{x}_0)}_{\boldsymbol{B}}\boldsymbol{x}$$

If $h(\boldsymbol{x})$ is scalar-valued then **gradient** $\nabla h(\boldsymbol{x})$ is given by the vector of partial correlations

$$\nabla h(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial h(\boldsymbol{x})}{\partial x_1} \\ \vdots \\ \frac{\partial h(\boldsymbol{x})}{\partial x_d} \end{pmatrix}$$

where ∇ is the nabla operator.

The **Jacobian matrix** is the **generalisation of the gradient** if $\boldsymbol{h}(\boldsymbol{x})$ is vector-valued:

$$J_{\boldsymbol{h}}(\boldsymbol{x}) = \begin{pmatrix} \nabla h_1(\boldsymbol{x})^T \\ \nabla h_2(\boldsymbol{x})^T \\ \vdots \\ \nabla h_m(\boldsymbol{x})^T \end{pmatrix} = \begin{pmatrix} \frac{\partial h_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial h_1(\boldsymbol{x})}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial h_m(\boldsymbol{x})}{\partial x_d} \end{pmatrix}$$

Note that in the Jacobian matrix by convention the gradient for each individual component of $\boldsymbol{h}(\boldsymbol{x})$ is contained in the *row* of the matrix so the number of rows corresponds to the dimension of $\boldsymbol{h}(\boldsymbol{x})$ and the number of columns to the dimension of $\boldsymbol{x}$.

First order approximation of $\boldsymbol{h}(\boldsymbol{x})$ around $\boldsymbol{x}_0 = \boldsymbol{\mu}_{\boldsymbol{x}}$ yields $\boldsymbol{a} = \boldsymbol{h}(\boldsymbol{\mu}_{\boldsymbol{x}}) - J_{\boldsymbol{h}}(\boldsymbol{\mu}_{\boldsymbol{x}})\,\boldsymbol{\mu}_{\boldsymbol{x}}$ and $\boldsymbol{B} = J_{\boldsymbol{h}}(\boldsymbol{\mu}_{\boldsymbol{x}})$ and leads directly to the **multivariate Delta method**:

$$\mathrm{E}(\boldsymbol{y}) \approx \boldsymbol{h}(\boldsymbol{\mu}_{\boldsymbol{x}})$$

$$\mathrm{Var}(\boldsymbol{y}) \approx J_{\boldsymbol{h}}(\boldsymbol{\mu}_{\boldsymbol{x}})\,\boldsymbol{\Sigma}_{\boldsymbol{x}}\,J_{\boldsymbol{h}}(\boldsymbol{\mu}_{\boldsymbol{x}})^T$$

The **univariate Delta method** is a special case:

$$\mathrm{E}(y) \approx h(\mu_x)$$

[1] Gorroochurn, P. 2020. Who Invented the Delta Method, Really? The Mathematical Intelligencer **42**:46–49. https://doi.org/10.1007/s00283-020-09982-0

$$\text{Var}(y) \approx \sigma_x^2 \, h'(\mu_x)^2$$

Note that the Delta approximation breaks down if $\text{Var}(\boldsymbol{y})$ is singular, for example if the first derivative (or gradient or Jacobian matrix) at $\boldsymbol{\mu_x}$ is zero.

Example 2.7. Variance of the odds ratio

The proportion $\hat{p} = \frac{n_1}{n}$ resulting from n repeats of a Bernoulli experiment has expectation $\text{E}(\hat{p}) = p$ and variance $\text{Var}(\hat{p}) = \frac{p(1-p)}{n}$. What are the (approximate) mean and the variance of the corresponding odds ratio $\widehat{OR} = \frac{\hat{p}}{1-\hat{p}}$?

With $h(x) = \frac{x}{1-x}$, $\widehat{OR} = h(\hat{p})$ and $h'(x) = \frac{1}{(1-x)^2}$ we get using the Delta method $\text{E}(\widehat{OR}) \approx h(p) = \frac{p}{1-p}$ and $\text{Var}(\widehat{OR}) \approx h'(p)^2 \text{Var}(\hat{p}) = \frac{p}{n(1-p)^3}$.

Example 2.8. Log-transform as variance stabilisation

Assume x has some mean $\text{E}(x) = \mu$ and variance $\text{Var}(x) = \sigma^2\mu^2$, i.e. the standard deviation $\text{SD}(x)$ is proportional to the mean μ. What are the (approximate) mean and the variance of the log-transformed variable $\log(x)$?

With $h(x) = \log(x)$ and $h'(x) = \frac{1}{x}$ we get using the Delta method $\text{E}(\log(x)) \approx h(\mu) = \log(\mu)$ and $\text{Var}(\log(x)) \approx h'(\mu)^2 \text{Var}(x) = \left(\frac{1}{\mu}\right)^2 \sigma^2\mu^2 = \sigma^2$. Thus, after applying the log-transform the variance does not depend any more on the mean!

2.2.3 Transformation of a probability density function under a general invertible transformation

Assume $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{y}(\boldsymbol{x})$ is invertible: $\boldsymbol{h}^{-1}(\boldsymbol{y}) = \boldsymbol{x}(\boldsymbol{y})$

$\boldsymbol{x} \sim F_{\boldsymbol{x}}$ with probability density function $f_{\boldsymbol{x}}(\boldsymbol{x})$

The density $f_{\boldsymbol{y}}(\boldsymbol{y})$ of the transformed random vector $\boldsymbol{y}$ is then given by

$$f_{\boldsymbol{y}}(\boldsymbol{y}) = |\det(J_{\boldsymbol{x}}(\boldsymbol{y}))| \; f_{\boldsymbol{x}}(\boldsymbol{x}(\boldsymbol{y}))$$

where $J_{\boldsymbol{x}}(\boldsymbol{y})$ is the Jacobian matrix of the inverse transformation.

Special cases:

- Univariate version: $f_y(y) = \left|\frac{dx(y)}{dy}\right| f_x(x(y))$
- Linear transformation $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{a} + \boldsymbol{B}\boldsymbol{x}$, with $\boldsymbol{x}(\boldsymbol{y}) = \boldsymbol{B}^{-1}(\boldsymbol{y} - \boldsymbol{a})$ and $J_{\boldsymbol{x}}(\boldsymbol{y}) = \boldsymbol{B}^{-1}$:

$$f_{\boldsymbol{y}}(\boldsymbol{y}) = |\det(\boldsymbol{B})|^{-1} \, f_{\boldsymbol{x}}\left(\boldsymbol{B}^{-1}(\boldsymbol{y} - \boldsymbol{a})\right)$$

2.2.4 Normalising flows

In this module we will focus mostly on linear transformations as these underpin much of classical multivariate statistics, but it is important to keep in mind for later study the importance of nonlinear transformations

In machine learning (sequences of) invertible nonlinear transformations are known as "normalising flows". They are used both in a generative way (building complex models from simple models) and for simplification and dimension reduction.

If you are interested in normalising flows then a good start to learn more are the review papers by Kobyzev et al (2021)[2] and Papamakarios et al. (2021).[3]

2.3 General whitening transformations

2.3.1 Overview

Whitening transformations are a special and widely used class of invertible location-scale transformations.

Terminology: whitening refers to the fact that after the transformation the covariance matrix is spherical, isotropic, white ($\boldsymbol{I}_d$)

Whitening is **useful in preprocessing**, as they allow to **turn multivariate models into uncorrelated univariate models** (via decorrelation property). Some whitening transformations **reduce the dimension in an optimal way** (via compression property).

The *Mahalanobis* transform is a specific example of a **whitening transformation**. It is also know as "zero-phase component analysis" or short ZCA transform.

In so-called latent variable models whitening procedures link observed (correlated) variables and latent variables (which typically are uncorrelated and standardised):

Whitening $\downarrow$ $\boldsymbol{x}$ $\uparrow$ $\boldsymbol{z}$

Observed variable (can be measured)
external, typically correlated

Unobserved "latent" variable (cannot be directly measured)
internal, typically chosen to be uncorrelated

[2] Kobyzev et al. 2021. *Normalizing Flows: Introduction and Ideas*. IEEE Trans. Pattern Anal. Mach. Intell. **43**:3964-3979

[3] Papamakarios et al. 2021. *Normalizing Flows for Probabilistic Modeling and Inference*. JMLR **22**:1-64

2.3.2 Whitening transformation and whitening constraint

Starting point:

Random vector $\boldsymbol{x} \sim F_{\boldsymbol{x}}$ **not necessarily from multivariate normal**.

$\boldsymbol{x}$ has mean $\mathrm{E}(\boldsymbol{x}) = \boldsymbol{\mu}$ and a positive definite (invertible) covariance matrix $\mathrm{Var}(\boldsymbol{x}) = \boldsymbol{\Sigma}$.

Note that in the following we leave out the subscript $\boldsymbol{x}$ for the covariance of $\boldsymbol{x}$ unless it is needed for clarification.

The covariance can be split into positive variances $\boldsymbol{V}$ and a positive definite invertible correlation matrix $\boldsymbol{P}$ so that $\boldsymbol{\Sigma} = \boldsymbol{V}^{1/2}\boldsymbol{P}\boldsymbol{V}^{1/2}$.

Whitening transformation:

$$\underbrace{\boldsymbol{z}}_{d\times 1 \text{ vector}} = \underbrace{\boldsymbol{W}}_{d\times d \text{ whitening matrix}} \underbrace{\boldsymbol{x}}_{d\times 1 \text{ vector}}$$

Objective: choose $\boldsymbol{W}$ so that $\mathrm{Var}(\boldsymbol{z}) = \boldsymbol{I}_d$

For Mahalanobis/ZCA whitening we already know that $\boldsymbol{W}^{\mathrm{ZCA}} = \boldsymbol{\Sigma}^{-1/2}$.

In general, the whitening matrix $\boldsymbol{W}$ needs to satisfy a constraint:

$$\begin{aligned} & \mathrm{Var}(\boldsymbol{z}) && = \boldsymbol{I}_d \\ \Longrightarrow\ & \mathrm{Var}(\boldsymbol{W}\boldsymbol{x}) && = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{W}^T = \boldsymbol{I}_d \\ \Longrightarrow\ & \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{W} \end{aligned}$$

$$\Longrightarrow \text{constraint on whitening matrix: } \boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{\Sigma}^{-1}$$

Clearly, the ZCA whitening matrix satisfies this constraint: $(\boldsymbol{W}^{ZCA})^T\boldsymbol{W}^{ZCA} = \boldsymbol{\Sigma}^{-1/2}\boldsymbol{\Sigma}^{-1/2} = \boldsymbol{\Sigma}^{-1}$

2.3.3 Parameterisation of whitening matrix

Covariance-based parameterisation of whitening matrix:

A general way to specify a valid whitening matrix is

$$\boldsymbol{W} = \boldsymbol{Q}_1\boldsymbol{\Sigma}^{-1/2}$$

where $\boldsymbol{Q}_1$ is an orthogonal matrix.[4]

Recall that an orthogonal matrix $\boldsymbol{Q}$ has the property that $\boldsymbol{Q}^{-1} = \boldsymbol{Q}^T$ and and as a consequence $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{I}$.

[4] In this Chapter we will make frequent use of orthogonal matrices. Check the Appendix to review their most important properties.

As a result, the above W satisfies the whitening constraint:

$$W^T W = \Sigma^{-1/2} \underbrace{Q_1^T Q_1}_{I} \Sigma^{-1/2} = \Sigma^{-1}$$

Note the converse is also true: any whitening whitening matrix, i.e. any W satisfying the whitening constraint, can be written in the above form as $Q_1 = W\Sigma^{1/2}$ is orthogonal by construction.

$\Longrightarrow$ instead of choosing W, **we choose the orthogonal matrix Q_1!**

- recall that orthogonal matrices geometrically represent rotations (plus reflections).
- it is now clear that there are infinitely many whitening procedures, because there are infinitely many rotations! This also means we need to find ways to choose/select among whitening procedures.
- for the Mahalanobis/ZCA transformation $Q_1^{\text{ZCA}} = I$
- **whitening** can be interpreted as **Mahalanobis transformation** followed by further **rotation-reflection**

Correlation-based parameterisation of whitening matrix:

Instead of working with the covariance matrix Σ, we can express W also in terms of the corresponding correlation matrix $P = (\rho_{ij}) = V^{-1/2}\Sigma V^{-1/2}$ where V is the diagonal matrix containing the variances.

Specifically, we can specify the whitening matrix as

$$W = Q_2 P^{-1/2} V^{-1/2}$$

It is easy to verify that this W also satisfies the whitening constraint:

$$\begin{aligned} W^T W &= V^{-1/2} P^{-1/2} \underbrace{Q_2^T Q_2}_{I} P^{-1/2} V^{-1/2} \\ &= V^{-1/2} P^{-1} V^{-1/2} = \Sigma^{-1} \end{aligned}$$

Conversely, any whitening matrix W can also be written in this form as $Q_2 = WV^{1/2}P^{1/2}$ is orthogonal by construction.

- **Another interpretation of whitening**: first **standardising** ($V^{-1/2}$), then **decorrelation** ($P^{-1/2}$), followed by **rotation-reflection** (Q_2)
- for Mahalanobis/ZCA transformation $Q_2^{\text{ZCA}} = \Sigma^{-1/2} V^{1/2} P^{1/2}$

Both forms to write W using Q_1 and Q_2 are equally valid (and interchangeable).

Note that for the same W

$$Q_1 \neq Q_2 \text{ Two different orthogonal matrices!}$$

and also

$$\underbrace{\boldsymbol{\Sigma}^{-1/2}}_{\text{Symmetric}} \neq \underbrace{\boldsymbol{P}^{-1/2}\boldsymbol{V}^{-1/2}}_{\text{Not Symmetric}}$$

even though

$$\boldsymbol{\Sigma}^{-1/2}\boldsymbol{\Sigma}^{-1/2} = \boldsymbol{\Sigma}^{-1} = \boldsymbol{V}^{-1/2}\boldsymbol{P}^{-1/2}\boldsymbol{P}^{-1/2}\boldsymbol{V}^{-1/2}$$

2.3.4 Cross-covariance and cross-correlation for general whitening transformations

A useful criterion to characterise and to distinguish among whitening transformations is the cross-covariance and cross-correlation matrix between the original variable $\boldsymbol{x}$ and the whitened variable $\boldsymbol{z}$:

a) **Cross-covariance $\boldsymbol{\Phi} = \Sigma_{xz}$** between $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$\begin{aligned}\boldsymbol{\Phi} = \mathrm{Cov}(\boldsymbol{x}, \boldsymbol{z}) &= \mathrm{Cov}(\boldsymbol{x}, \boldsymbol{W}\boldsymbol{x}) \\ &= \boldsymbol{\Sigma}\boldsymbol{W}^T \\ &= \boldsymbol{\Sigma}\,\boldsymbol{\Sigma}^{-1/2}\boldsymbol{Q}_1^T \\ &= \boldsymbol{\Sigma}^{1/2}\boldsymbol{Q}_1^T\end{aligned}$$

In component notation we write $\boldsymbol{\Phi} = (\phi_{ij})$ where the row index i refers to $\boldsymbol{x}$ and the column index j to $\boldsymbol{z}$.

Cross-covariance is linked with Q_1! Thus, choosing cross-covariance determines $\boldsymbol{Q}_1$ (and vice versa).

Note that the above cross-covariance matrix $\boldsymbol{\Phi}$ satisfies the condition $\boldsymbol{\Phi}\boldsymbol{\Phi}^T = \boldsymbol{\Sigma}$.

The whitening matrix expressed in terms of cross-covariance is $\boldsymbol{W} = \boldsymbol{\Phi}^T\boldsymbol{\Sigma}^{-1}$, so as required $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Sigma}^{-1} = \boldsymbol{\Sigma}^{-1}$. Furthermore, $\boldsymbol{\Phi}$ is the *inverse* of the whitening matrix, as $\boldsymbol{W}^{-1} = \left(\boldsymbol{Q}_1\boldsymbol{\Sigma}^{-1/2}\right)^{-1} = \boldsymbol{\Sigma}^{1/2}\boldsymbol{Q}_1^{-1} = \boldsymbol{\Sigma}^{1/2}\boldsymbol{Q}_1^T = \boldsymbol{\Phi}$.

b) **Cross-correlation $\boldsymbol{\Psi} = \boldsymbol{P}_{xz}$** between $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$\begin{aligned}\boldsymbol{\Psi} = \mathrm{Cor}(\boldsymbol{x}, \boldsymbol{z}) &= \boldsymbol{V}^{-1/2}\boldsymbol{\Phi} \\ &= \boldsymbol{V}^{-1/2}\boldsymbol{\Sigma}\boldsymbol{W}^T \\ &= \boldsymbol{V}^{-1/2}\boldsymbol{\Sigma}\boldsymbol{V}^{-1/2}\boldsymbol{P}^{-1/2}\boldsymbol{Q}_2^T \\ &= \boldsymbol{P}^{1/2}\boldsymbol{Q}_2^T\end{aligned}$$

In component notation we write $\boldsymbol{\Psi} = (\psi_{ij})$ where the row index i refers to $\boldsymbol{x}$ and the column index j to $\boldsymbol{z}$.

Cross-correlation is linked with Q_2! Hence, choosing cross-correlation determines Q_2 (and vice versa). The whitening matrix expressed in terms of cross-correlation is $W = \Psi^T P^{-1} V^{-1/2}$.

Note that the factorisation of the cross-covariance $\Phi = \Sigma^{1/2} Q_1^T$ and the cross-correlation $\Psi = P^{1/2} Q_2^T$ into the product of a positive definite symmetric matrix and an orthogonal matrix are examples of a **polar decomposition**.

2.3.5 Inverse whitening transformation and loadings

Inverse transformation:

Recall that $z = Wx$. Therefore, the reverse transformation going from the whitened to the original variable is $x = W^{-1} z$. This can be expressed also in terms of cross-covariance and cross-correlation. With $W^{-1} = \Phi$ we get

$$x = \Phi z \,.$$

Furthermore, since $\Psi = V^{-1/2} \Phi$ we have $W^{-1} = V^{1/2} \Psi$ and hence

$$V^{-1/2} x = \Psi z \,.$$

The reverse whitening transformation is also known as colouring transformation (the previously discussed inverse Mahalanobis transform is one example).

Definition of loadings:

Loadings are the coefficients of the linear transformation from the latent variable back to the observed variable. If the variables are standardised to unit variance then the loadings are also called *correlation loadings*.

Hence, the cross-covariance matrix Φ plays the role of *loadings* linking the latent variable z with the original x. Similarly, the cross-correlation matrix Ψ contains the *correlation loadings* linking the (already standardised) latent variable z with the standardised x.

In the convention we use here the rows correspond to the original variables and the columns to the whitened latent variables.

Multiple correlation coefficients from z back to x:

We consider the backtransformation from the whitened variable z to the original variables x and note that the components of z are all uncorrelated with $P_z = I$. The squared multiple correlation coefficient $\mathrm{MCor}(x_i, z)$ between each x_i and all

$\boldsymbol{z}$ is therefore just the sum of the corresponding squared correlations $\text{Cor}(x_i, z_j)^2$:

$$\begin{aligned}\text{MCor}(x_i, \boldsymbol{z})^2 = \boldsymbol{P}_{x_i\boldsymbol{z}}\boldsymbol{P}_{\boldsymbol{z}}^{-1}\boldsymbol{P}_{\boldsymbol{z}x_i} = \\ \sum_{j=1}^{d} \text{Cor}(x_i, z_j)^2 \\ = \sum_{j=1}^{d} \psi_{ij}^2 = 1\end{aligned}$$

As shown earlier for a general linear one-to-one- transformation (which includes whitening as special case) the squared multiple correlation must be 1 because there is no error. We can confirm this by computing the **row sums of squares** of the cross-correlation matrix $\boldsymbol{\Psi}$ in matrix notation

$$\begin{aligned}\text{Diag}\left(\boldsymbol{\Psi}\boldsymbol{\Psi}^T\right) &= \text{Diag}\left(\boldsymbol{P}^{1/2}\boldsymbol{Q}_2^T\boldsymbol{Q}_2\boldsymbol{P}^{1/2}\right) \\ &= \text{Diag}(\boldsymbol{P}) \\ &= (1, \ldots, 1)^T\end{aligned}$$

from which it is clear that the choice of $\boldsymbol{Q}_2$ is not relevant.

Similarly, the **row sums of squares** of the cross-covariance matrix $\boldsymbol{\Phi}$ equal the variances of the original variables, regardless of $\boldsymbol{Q}_1$:

$$\sum_{j=1}^{d} \phi_{ij}^2 = \text{Var}(x_i)$$

or in matrix notation

$$\begin{aligned}\text{Diag}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^T\right) &= \text{Diag}\left(\boldsymbol{\Sigma}^{1/2}\boldsymbol{Q}_1^T\boldsymbol{Q}_1\boldsymbol{\Sigma}^{1/2}\right) \\ &= \text{Diag}(\boldsymbol{\Sigma}) \\ &= (\text{Var}(x_1), \ldots, \text{Var}(x_d)^T\end{aligned}$$

2.3.6 Summaries of cross-covariance $\boldsymbol{\Phi}$ and cross-correlation $\boldsymbol{\Psi}$ resulting from whitening transformations

Matrix trace:

A simply summary of a matrix is its trace. For the cross-covariance matrix $\boldsymbol{\Phi}$ the trace is the sum of all covariances between corresponding elements in $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$\text{Tr}(\boldsymbol{\Phi}) = \sum_{i=1}^{d} \text{Cov}(x_i, z_i) = \sum_{i=1}^{d} \phi_{ii} = \text{Tr}\left(\boldsymbol{\Sigma}^{1/2}\boldsymbol{Q}_1^T\right)$$

Likewise, for the cross-correlation matrix $\boldsymbol{\Psi}$ the trace is the sum of all correlations between corresponding elements in $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$\text{Tr}(\boldsymbol{\Psi}) = \sum_{i=1}^{d} \text{Cor}(x_i, z_i) = \sum_{i=1}^{d} \psi_{ii} = \text{Tr}\left(\boldsymbol{P}^{1/2}\boldsymbol{Q}_2^T\right)$$

In both cases the value of the trace depends on $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$. Interestingly, there is unique choice such that the trace is maximised.

Specifically, to maximise $\text{Tr}(\boldsymbol{\Phi})$ we conduct the following steps:

i) Apply eigendecomposition to $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$. Note that $\boldsymbol{\Lambda}$ is diagonal with positive eigenvalues $\lambda_i > 0$ as $\boldsymbol{\Sigma}$ is positive definite and $\boldsymbol{U}$ is an orthogonal matrix.
ii) The objective function becomes

$$\begin{aligned}
\text{Tr}(\boldsymbol{\Phi}) &= \text{Tr}\left(\boldsymbol{\Sigma}^{1/2}\boldsymbol{Q}_1^T\right) \\
&= \text{Tr}\left(\boldsymbol{U}\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^T\boldsymbol{Q}_1^T\right) \\
&= \text{Tr}\left(\boldsymbol{\Lambda}^{1/2}\,\boldsymbol{U}^T\boldsymbol{Q}_1^T\boldsymbol{U}\right) \\
&= \text{Tr}\left(\boldsymbol{\Lambda}^{1/2}\,\boldsymbol{B}\right) \\
&= \sum_{i=1}^{d} \lambda_i^{1/2} b_{ii}.
\end{aligned}$$

Note that the product of two orthogonal matrices is itself an orthogonal matrix. Therefore, $\boldsymbol{B} = \boldsymbol{U}^T\boldsymbol{Q}_1^T\boldsymbol{U}$ is an orthogonal matrix and $\boldsymbol{Q}_1 = \boldsymbol{U}\boldsymbol{B}^T\boldsymbol{U}^T$.
iii) As $\lambda_i > 0$ and all $b_{ii} \in [-1, 1]$ the objective function is maximised for $b_{ii} = 1$, i.e. for $\boldsymbol{B} = \boldsymbol{I}$.
iv) In turn this implies that $\text{Tr}(\boldsymbol{\Phi})$ is maximised for $\boldsymbol{Q}_1 = \boldsymbol{I}$.

Similarly, to maximise $\text{Tr}(\boldsymbol{\Psi})$ we

- decompose $\boldsymbol{P} = \boldsymbol{G}\boldsymbol{\Theta}\boldsymbol{G}^T$ and then, following the above,
- find that $\text{Tr}(\boldsymbol{\Psi}) = \text{Tr}\left(\boldsymbol{\Theta}^{1/2}\,\boldsymbol{G}^T\boldsymbol{Q}_2^T\boldsymbol{G}\right)$ is maximised for $\boldsymbol{Q}_2 = \boldsymbol{I}$.

Squared Frobenius norm and total variation:

Another way to summarise and dissect the association between $\boldsymbol{x}$ and the corresponding whitened $\boldsymbol{z}$ is the squared Frobenius norm and the total variation based on $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$.

The squared Frobenius norm (Euclidean) norm is the sum of squared elements of a matrix.

If we consider the squared Frobenius norm of the cross-covariance matrix, i.e. the sum of squared covariances between $\boldsymbol{x}$ and $\boldsymbol{z}$,

$$||\boldsymbol{\Phi}||_F^2 = \sum_{i=1}^{d}\sum_{j=1}^{d}\phi_{ij}^2 = \mathrm{Tr}(\boldsymbol{\Phi}^T\boldsymbol{\Phi}) = \mathrm{Tr}(\boldsymbol{\Sigma})$$

we find that this equals the **total variation** of $\boldsymbol{\Sigma}$ and that it does not depend on $\boldsymbol{Q}_1$. Likewise, computing the squared Frobenius norm of the cross-correlation matrix, i.e. the sum of squared correlations between $\boldsymbol{x}$ and $\boldsymbol{z}$,

$$||\boldsymbol{\Psi}||_F^2 = \sum_{i=1}^{d}\sum_{j=1}^{d}\psi_{ij}^2 = \mathrm{Tr}\left(\boldsymbol{\Psi}^T\boldsymbol{\Psi}\right) = \mathrm{Tr}\left(\boldsymbol{P}\right) = d$$

yields the total variation of $\boldsymbol{P}$ which also does not depend on $\boldsymbol{Q}_2$. Note this is because the squared Frobenius norm is invariant against rotations and reflections.

Proportion of total variation:

We can now compute the contribution of each whitened component z_j to the total variation. The sum of squared covariances of each z_j with all $x_1, \ldots, x_d$ is

$$h_j = \sum_{i=1}^{d}\mathrm{Cov}(x_i, z_j)^2 = \sum_{i=1}^{d}\phi_{ij}^2$$

with $\sum_{j=1}^{d} h_j = \mathrm{Tr}\left(\boldsymbol{\Sigma}\right)$ the total variation. In vector notation the contributions are written as the **column sums of squares** of $\boldsymbol{\Phi}$

$$\boldsymbol{h} = (h_1, \ldots, h_d)^T = \mathrm{Diag}(\boldsymbol{\Phi}^T\boldsymbol{\Phi}) = \mathrm{Diag}\left(\boldsymbol{Q}_1\boldsymbol{\Sigma}\boldsymbol{Q}_1^T\right).$$

The relative contribution of z_j versus the total variation is

$$\frac{h_j}{\mathrm{Tr}\left(\boldsymbol{\Sigma}\right)}.$$

Crucially, in contrast to total variation, the contributions h_j depend on the choice of $\boldsymbol{Q}_1$.

Similarly, the sum of squared correlations of each z_j with all $x_1, \ldots, x_d$ is

$$k_j = \sum_{i=1}^{d}\mathrm{Cor}(x_i, z_j)^2 = \sum_{i=1}^{d}\psi_{ij}^2$$

with $\sum_{i=j}^{d} k_j = \mathrm{Tr}(\boldsymbol{P}) = d$. In vector notation this correspoinds to the **column sums of squares** of $\boldsymbol{\Psi}$

$$\boldsymbol{k} = (k_1, \ldots, k_d)^T = \mathrm{Diag}\left(\boldsymbol{\Psi}^T\boldsymbol{\Psi}\right) = \mathrm{Diag}\left(\boldsymbol{Q}_2\boldsymbol{P}\boldsymbol{Q}_2^T\right).$$

The relative contribution of z_j with regard to the total variation of the correlation $\boldsymbol{P}$ is

$$\frac{k_j}{\mathrm{Tr}(\boldsymbol{P})} = \frac{k_j}{d}\,.$$

As above, the contributions k_j depend on the choice of $\boldsymbol{Q}_2$.

Maximising the proportion of total variation:

Interestingly, it is possible to choose a unique whitening transformation such that the contributions are maximised, i.e. that the sum of the m largest contributions of h_j and k_j is as large as possible.

Specifically, we note that $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}^T\boldsymbol{\Psi}$ are symmetric real matrices. For these type of matrices we know from Schur's theorem (1923) that the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$ **majorise** the diagonal elements $p_1 \geq p_2 \geq \ldots \geq p_d$. More precisely,

$$\sum_{i=1}^{m} \lambda_i \geq \sum_{i=1}^{m} p_i\,,$$

i.e. the sum of the largest m eigenvalues is larger than or equal to the sum of the m largest diagonal elements. The maximum (and equality) is only achieved if the matrix is diagonal, as in this case the diagonal elements are equal to the eigenvalues.

Therefore, the optimal solution to problem of maximising the relative contributions is obtained by computing the eigendecompositions $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ and $\boldsymbol{P} = \boldsymbol{G}\boldsymbol{\Theta}\boldsymbol{G}^T$ and diagonalise $\boldsymbol{\Phi}^T\boldsymbol{\Phi} = \boldsymbol{Q}_1\boldsymbol{\Sigma}\boldsymbol{Q}_1^T$ and $\boldsymbol{\Psi}^T\boldsymbol{\Psi} = \boldsymbol{Q}_2\boldsymbol{P}\boldsymbol{Q}_2^T$ by setting $\boldsymbol{Q}_1 = \boldsymbol{U}^T$ and $\boldsymbol{Q}_2 = \boldsymbol{G}^T$, respectively. This yields for the maximised contributions

$$\boldsymbol{h} = \mathrm{Diag}\,(\boldsymbol{\Lambda}) = (\lambda_1, \ldots, \lambda_d)^T$$

and

$$\boldsymbol{k} = \mathrm{Diag}\,(\boldsymbol{\Theta}) = (\theta_1, \ldots, \theta_d)^T$$

with eigenvalues λ_i and θ_i arranged in decreasing order.

2.4 Natural whitening procedures

We now introduce several strategies (maximise correlation between individual components, maximise compression, structural constraints) to select an optimal whitening procedure.

Specifically, we discuss the following whitening transformations:

- **Mahalanobis** whitening, also known as **ZCA** (zero-phase component analysis) whitening in machine learning (based on covariance)
- **ZCA-cor** whitening (based on correlation)
- **PCA** whitening (based on covariance)

- **PCA-cor** whitening (based on correlation)
- **Cholesky** whitening

Thus, in the following we consider three main types (ZCA, PCA, Cholesky) of whitening.

In the following $\boldsymbol{x}_c = \boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}}$ and $\boldsymbol{z}_c = \boldsymbol{z} - \boldsymbol{\mu}_{\boldsymbol{z}}$ denote the mean-centered variables.

2.4.1 ZCA whitening

Aim: remove correlations and standardise but otherwise make sure that the whitened vector $\boldsymbol{z}$ does not differ too much from the original vector $\boldsymbol{x}$. Specifically, each latent component z_i should be as close as as possible to the corresponding original variable x_i:

$$\begin{aligned} x_1 &\leftrightarrow z_1 \\ x_2 &\leftrightarrow z_2 \\ x_3 &\leftrightarrow z_3 \\ &\vdots \end{aligned}$$

One possible way to implement this is to compute the expected squared difference between the two centered random vectors $\boldsymbol{z}_c$ and $\boldsymbol{x}_c$.

ZCA objective function: **minimise** $\mathrm{E}\left(||\boldsymbol{x}_c - \boldsymbol{z}_c||_F^2\right)$ to find an optimal whitening procedure.

The ZCA objective function can be simplified as follows:

$$\begin{aligned} \mathrm{E}\left(||\boldsymbol{x}_c - \boldsymbol{z}_c||_F^2\right) &= \mathrm{E}\left(||\boldsymbol{x}_c||_F^2\right) - 2\mathrm{E}\left(\mathrm{Tr}\left(\boldsymbol{x}_c \boldsymbol{z}_c^T\right)\right) + \mathrm{E}\left(||\boldsymbol{z}_c||_F^2\right) \\ &= \mathrm{Tr}(\mathrm{E}(\boldsymbol{x}_c \boldsymbol{x}_c^T)) - 2\mathrm{Tr}(\mathrm{E}(\boldsymbol{x}_c \boldsymbol{z}_c^T)) + \mathrm{Tr}(\mathrm{E}(\boldsymbol{z}_c \boldsymbol{z}_c^T)) \\ &= \mathrm{Tr}(\mathrm{Var}(\boldsymbol{x})) - 2\mathrm{Tr}(\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{z})) + \mathrm{Tr}(\mathrm{Var}(\boldsymbol{z})) \\ &= \mathrm{Tr}(\boldsymbol{\Sigma}) - 2\mathrm{Tr}(\boldsymbol{\Phi}) + d \end{aligned}$$

The same objective function can also be obtained by putting a diagonal constraint on the cross-covariance $\boldsymbol{\Phi}$. Specifically, we are looking for the $\boldsymbol{\Phi}$ that is closest to the diagonal matrix $\boldsymbol{I}$ by **minimising**

$$\begin{aligned} ||\boldsymbol{\Phi} - \boldsymbol{I}||_F^2 &= ||\boldsymbol{\Phi}||_F^2 - 2\mathrm{Tr}(\boldsymbol{\Phi}^T \boldsymbol{I}) + ||\boldsymbol{I}||_F^2 \\ &= \mathrm{Tr}(\boldsymbol{\Sigma}) - 2\mathrm{Tr}(\boldsymbol{\Phi}) + d \end{aligned}$$

This will force the off-diagonal elements of $\boldsymbol{\Phi}$ to be close to zero and thus leads to sparsity in the cross-covariance matrix.

The only term in the above that depends on the whitening transformation is $-2\mathrm{Tr}(\boldsymbol{\Phi})$ as $\boldsymbol{\Phi}$ is a function of $\boldsymbol{Q}_1$. Therefore we can use the following alternative objective:

ZCA equivalent objective: **maximise** $\mathrm{Tr}(\boldsymbol{\Phi}) = \mathrm{Tr}(\boldsymbol{\Sigma}^{1/2} \boldsymbol{Q}_1^T)$ to find the optimal $\boldsymbol{Q}_1$

Solution:

From the earlier discussion we know that the optimal matrix is

$$\boldsymbol{Q}_1^{\text{ZCA}} = \boldsymbol{I}$$

The corresponding whitening matrix for ZCA is therefore

$$\boldsymbol{W}^{\text{ZCA}} = \boldsymbol{\Sigma}^{-1/2}$$

and the cross-covariance matrix is

$$\boldsymbol{\Phi}^{\text{ZCA}} = \boldsymbol{\Sigma}^{1/2}$$

and the cross-correlation matrix

$$\boldsymbol{\Psi}^{\text{ZCA}} = \boldsymbol{V}^{-1/2}\boldsymbol{\Sigma}^{1/2}$$

Note that $\boldsymbol{\Sigma}^{1/2}$ is a symmetric positive definite matrix, hence its diagonal elements are all positive. As a result, the diagonals of $\boldsymbol{\Phi}^{\text{ZCA}}$ and $\boldsymbol{\Psi}^{\text{ZCA}}$ are positive, i.e. $\text{Cov}(x_i, z_i) > 0$ and $\text{Cor}(x_i, z_i) > 0$. Hence, for ZCA two corresponding components x_i and z_i are always positively correlated!

Proportion of total variation:

For ZCA with $\boldsymbol{Q}_1 = \boldsymbol{I}$ we find that $\boldsymbol{h} = \text{Diag}(\boldsymbol{\Sigma}) = \sum_{j=1}^{d} \text{Var}(x_j)$ with $h_i = \text{Var}(x_i)$ hence for ZCA the proportion of total variation contributed by the latent component z_i is the ratio $\frac{\text{Var}(x_i)}{\sum_{j=1}^{d} \text{Var}(x_j)}$.

Summary:

- ZCA/Mahalanobis transform is the unique transformation that minimises the expected total squared component-wise difference between $\boldsymbol{x}_c$ and $\boldsymbol{z}_c$.
- In ZCA corresponding components in the whitened and original variables are always positively correlated. This facilitates the interpretation of the whitened variables.
- Use ZCA aka Mahalanobis whitening if you want to "just" remove correlations.

2.4.2 ZCA-Cor whitening

Aim: same as above but remove scale in $\boldsymbol{x}$ first before comparing to $\boldsymbol{z}$.

ZCA-cor objective function: **minimise** $\text{E}\left(||\boldsymbol{V}^{-1/2}\boldsymbol{x}_c - \boldsymbol{z}_c||_F^2\right)$ to find an optimal whitening procedure.

This can be simplified as follows:

$$\begin{aligned}
\mathrm{E}\left(||\boldsymbol{V}^{-1/2}\boldsymbol{x}_c - \boldsymbol{z}_c||_F^2\right) &= \mathrm{E}\left(||\boldsymbol{V}^{-1/2}\boldsymbol{x}_c||_F^2\right) - 2\mathrm{E}\left(\mathrm{Tr}\left(\boldsymbol{V}^{-1/2}\boldsymbol{x}_c\boldsymbol{z}_c^T\right)\right) + \mathrm{E}\left(||\boldsymbol{z}_c||_F^2\right) \\
&= \mathrm{Tr}(\mathrm{E}(\boldsymbol{V}^{-1/2}\boldsymbol{x}_c\boldsymbol{x}_c^T\boldsymbol{V}^{-1/2})) - 2\mathrm{Tr}(\mathrm{E}(\boldsymbol{V}^{-1/2}\boldsymbol{x}_c\boldsymbol{z}_c^T)) + \mathrm{Tr}(\mathrm{E}(\boldsymbol{z}_c\boldsymbol{z}_c^T)) \\
&= \mathrm{Tr}(\mathrm{Cor}(\boldsymbol{x},\boldsymbol{x})) - 2\mathrm{Tr}(\mathrm{Cor}(\boldsymbol{x},\boldsymbol{z})) + \mathrm{Tr}(\mathrm{Var}(\boldsymbol{z})) \\
&= d - 2\mathrm{Tr}(\boldsymbol{\Psi}) + d \\
&= 2d - 2\mathrm{Tr}(\boldsymbol{\Psi})
\end{aligned}$$

The same objective function can also be obtained by putting a diagonal constraint on the cross-correlation $\boldsymbol{\Psi}$. Specifically, we are looking for the $\boldsymbol{\Psi}$ that is closest to the diagonal matrix $\boldsymbol{I}$ by **minimising**

$$\begin{aligned}
||\boldsymbol{\Psi} - \boldsymbol{I}||_F^2 &= ||\boldsymbol{\Psi}||_F^2 - 2\mathrm{Tr}(\boldsymbol{\Psi}^T\boldsymbol{I}) + ||\boldsymbol{I}||_F^2 \\
&= d - 2\mathrm{Tr}(\boldsymbol{\Psi}) + d \\
&= 2d - 2\mathrm{Tr}(\boldsymbol{\Psi})
\end{aligned}$$

This will force the off-diagonal elements of $\boldsymbol{\Psi}$ to be close to zero and thus leads to sparsity in the cross-correlation matrix.

The only term in the above that depends on the whitening transformation is $-2\mathrm{Tr}(\boldsymbol{\Psi})$ as $\boldsymbol{\Psi}$ is a function of $\boldsymbol{Q}_2$. Thus we can use the following alternative objective instead:

ZCA-cor equivalent objective: **maximise** $\mathrm{Tr}(\boldsymbol{\Psi}) = \mathrm{Tr}(\boldsymbol{P}^{1/2}\boldsymbol{Q}_2^T)$ to find optimal $\boldsymbol{Q}_2$

Solution: same as above for ZCA but using correlation instead of covariance:

From the earlier discussion we know that the optimal matrix is

$$\boldsymbol{Q}_2^{\text{ZCA-Cor}} = \boldsymbol{I}$$

The corresponding whitening matrix for ZCA-cor is therefore

$$\boldsymbol{W}^{\text{ZCA-Cor}} = \boldsymbol{P}^{-1/2}\boldsymbol{V}^{-1/2}$$

and the cross-covariance matrix is

$$\boldsymbol{\Phi}^{\text{ZCA-Cor}} = \boldsymbol{V}^{1/2}\boldsymbol{P}^{1/2}$$

and the cross-correlation matrix is

$$\boldsymbol{\Psi}^{\text{ZCA-Cor}} = \boldsymbol{P}^{1/2}$$

For the ZCA-cor transformation we also have $\mathrm{Cov}(x_i, z_i) > 0$ and $\mathrm{Cor}(x_i, z_i) > 0$ so that two corresponding components x_i and z_i are always positively correlated!

Proportion of total variation:

For ZCA-cor with $\boldsymbol{Q}_2 = \boldsymbol{I}$ we find that $\boldsymbol{k} = \text{Diag}(\boldsymbol{P}) = d$ with all $k_i = 1$. Thus, in ZCA-cor each whitened component z_i contributes equally to the total variation $\text{Tr}(\boldsymbol{P}) = d$, with relative proportion $\frac{1}{d}$.

Summary:

- ZCA-cor whitening is the unique whitening transformation maximising the total correlation between corresponding elements in $\boldsymbol{x}$ and $\boldsymbol{z}$.
- ZCA-cor leads to interpretable $\boldsymbol{z}$ because each individual element in $\boldsymbol{z}$ is (typically strongly) positively correlated with the corresponding element in the original $\boldsymbol{x}$.
- As ZCA-cor is explicitly constructed to maximise the total pairwise correlations it achieves higher total correlation than ZCA.
- If $\boldsymbol{x}$ is standardised to $\text{Var}(x_i) = 1$ then ZCA and ZCA-cor are identical.

2.4.3 PCA whitening

Aim: remove correlations and at the same time compress information into a few latent variables. Specifically, we would like that the first latent component z_1 is maximally linked with all variables in $\boldsymbol{x}$, followed by the second component z_2 and so on:

$$\begin{aligned} x_1, x_2, \ldots, x_d &\rightarrow z_1 \\ x_1, x_2, \ldots, x_d &\rightarrow z_2 \\ &\vdots \\ x_1, x_2, \ldots, x_d &\rightarrow z_d \end{aligned}$$

One way to measure the total association of the latent component z_j with all the original $x_1, \ldots, x_d$ is the sum of the corresponding squared covariances

$$h_j = \sum_{i=1}^{d} \text{Cov}(x_i, z_j)^2 = \sum_{i=1}^{d} \phi_{ij}^2$$

or equivalently the **column sum of squares** of $\boldsymbol{\Phi}$

$$\boldsymbol{h} = (h_1, \ldots, h_d)^T = \text{Diag}(\boldsymbol{\Phi}^T\boldsymbol{\Phi}) = \text{Diag}\left(\boldsymbol{Q}_1\boldsymbol{\Sigma}\boldsymbol{Q}_1^T\right)$$

Each h_j is the contribution of z_j to $\text{Tr}\left(\boldsymbol{Q}_1\boldsymbol{\Sigma}\boldsymbol{Q}_1^T\right) = \text{Tr}(\boldsymbol{\Sigma})$ i.e. to the total variation based on $\boldsymbol{\Sigma}$. As $\text{Tr}(\boldsymbol{\Sigma})$ is constant this implies that there are only $d-1$ independent h_j.

In PCA-whitening we wish to concentrate most of the contributions to the total variation based on $\boldsymbol{\Sigma}$ in a small number of latent components.

PCA whitening objective function: find an optimal optimal $\boldsymbol{Q}_1$ so that the resulting set $h_1 \geq h_2 \ldots \geq h_d$ in $\boldsymbol{h} = \text{Diag}\left(\boldsymbol{Q}_1\boldsymbol{\Sigma}\boldsymbol{Q}_1^T\right)$ majorizes any other set of relative contributions.

Solution:

Following the earlier discussion we apply Schur's theorem and find the optimal solution by diagonalising $\mathbf{\Phi}^T\mathbf{\Phi}$ through eigendecomposition of $\mathbf{\Sigma} = \boldsymbol{U}\mathbf{\Lambda}\boldsymbol{U}^T$. Hence, the optimal value for the $\boldsymbol{Q}_1$ matrix is

$$\boldsymbol{Q}_1^{\text{PCA}} = \boldsymbol{U}^T$$

However, recall that $\boldsymbol{U}$ is not uniquely defined — you are free to change the columns signs. The corresponding whitening matrix is

$$\boldsymbol{W}^{\text{PCA}} = \boldsymbol{U}^T\mathbf{\Sigma}^{-1/2} = \mathbf{\Lambda}^{-1/2}\boldsymbol{U}^T$$

and the cross-covariance matrix is

$$\mathbf{\Phi}^{\text{PCA}} = \boldsymbol{U}\mathbf{\Lambda}^{1/2}$$

and the cross-correlation matrix is

$$\mathbf{\Psi}^{\text{PCA}} = \boldsymbol{V}^{-1/2}\boldsymbol{U}\mathbf{\Lambda}^{1/2}$$

Identifiability:

Note that all of the above (i.e. $\boldsymbol{Q}_1^{\text{PCA}}, \boldsymbol{W}^{\text{PCA}}, \mathbf{\Phi}^{\text{PCA}}, \mathbf{\Psi}^{\text{PCA}}$) is not unique due to the sign ambiguity in the columns of $\boldsymbol{U}$.

Therefore, for identifiability reasons we may wish to impose a further constraint on $\boldsymbol{Q}_1^{\text{PCA}}$ or equivalently $\mathbf{\Phi}^{\text{PCA}}$. A useful condition is to require (for the given ordering of the original variables!) that $\boldsymbol{Q}_1^{\text{PCA}}$ has a positive diagonal or equivalently that $\mathbf{\Phi}^{\text{PCA}}$ has a positive diagonal. This implies that $\text{Diag}(\boldsymbol{U}) > 0$ and $\text{Diag}(\mathbf{\Psi}^{\text{PCA}}) > 0$, hence all pairs x_i and z_i are positively correlated.

It is particularly important to pay attention to the sign ambiguity when comparing different computer implementations of PCA whitening (and the related PCA approach).

Note that the actual objective of PCA whitening $\text{Diag}(\mathbf{\Phi}^T\mathbf{\Phi})$ is not affected by the sign ambiguity since the column signs of $\mathbf{\Phi}$ do not matter.

Proportion of total variation:

In PCA whitening the contribution h_i^{PCA} of each latent component z_i to the total variation based on the covariance $\text{Tr}(\mathbf{\Sigma}) = \sum_{j=1}^d \lambda_j$ is $h_i^{\text{PCA}} = \lambda_i$. The fraction $\frac{\lambda_i}{\sum_{j=1}^d \lambda_j}$ is the relative contribution of each element in $\boldsymbol{z}$ to explain the total variation.

Thus, low ranking components z_i with small $h_i^{\text{PCA}} = \lambda_i$ may be discarded. In this way PCA whitening achieves both compression and dimension reduction.

Summary:

- PCA whitening is a whitening transformation that maximises compression with the sum of squared cross-covariances as underlying optimality criterion.
- There are sign ambiguities in the PCA whitened variables which are inherited from the sign ambiguities in eigenvectors.
- If a positive-diagonal condition on the orthogonal matrices is imposed then these sign ambiguities are fully resolved and corresponding components z_i and x_i are always positively correlated.

2.4.4 PCA-cor whitening

Aim: same as for PCA whitening but remove scale in $\boldsymbol{x}$ first. This means we use squared correlations rather than squared covariances to measure compression, i.e.

$$k_j = \sum_{i=1}^{d} \text{Cor}(x_i, z_j)^2 = \sum_{i=1}^{d} \psi_{ij}^2$$

or in vector notation the **column sum of squares** of $\boldsymbol{\Psi}$

$$\boldsymbol{k} = (k_1, \ldots, k_d)^T = \text{Diag}\left(\boldsymbol{\Psi}^T\boldsymbol{\Psi}\right) = \text{Diag}\left(\boldsymbol{Q}_2\boldsymbol{P}\boldsymbol{Q}_2^T\right)$$

Each k_j is the contribution of z_j to $\text{Tr}\left(\boldsymbol{Q}_2\boldsymbol{P}\boldsymbol{Q}_2^T\right) = \text{Tr}(\boldsymbol{P}) = d$ i.e. the total variation based on $\boldsymbol{P}$. As $\text{Tr}(\boldsymbol{P}) = d$ is constant this implies that there are only $d - 1$ independent k_j.

In PCA-cor-whitening we wish to concentrate most of the contributions to the total variation based on $\boldsymbol{P}$ in a small number of latent components.

PCA-cor whitening objective function: find an optimal optimal $\boldsymbol{Q}_2$ so that the resulting set $k_1 \geq k_2 \ldots \geq k_d$ in $\boldsymbol{k} = \text{Diag}\left(\boldsymbol{Q}_2\boldsymbol{P}\boldsymbol{Q}_2^T\right)$ majorizes any other set of relative contributions.

Solution:

Following the earlier discussion we apply Schur's theorem and find the optimal solution by diagonalising $\boldsymbol{\Psi}^T\boldsymbol{\Psi}$ through eigendecomposition of $\boldsymbol{P} = \boldsymbol{G}\boldsymbol{\Theta}\boldsymbol{G}^T$. Hence, the optimal value for the $\boldsymbol{Q}_2$ matrix is

$$\boldsymbol{Q}_2^{\text{PCA-Cor}} = \boldsymbol{G}^T$$

Again $\boldsymbol{G}$ is not uniquely defined — you are free to change signs of the columns. The corresponding whitening matrix is

$$\boldsymbol{Q}_2^{\text{PCA-Cor}} = \boldsymbol{G}^T$$

The corresponding whitening matrix is

$$\boldsymbol{W}^{\text{PCA-Cor}} = \boldsymbol{\Theta}^{-1/2}\boldsymbol{G}^T\boldsymbol{V}^{-1/2}$$

and the cross-covariance matrix is

$$\mathbf{\Phi}^{\text{PCA-Cor}} = \boldsymbol{V}^{1/2}\boldsymbol{G}\mathbf{\Theta}^{1/2}$$

and the cross-correlation matrix is

$$\mathbf{\Psi}^{\text{PCA-Cor}} = \boldsymbol{G}\mathbf{\Theta}^{1/2}$$

Identifiability:

As with PCA whitening, there are sign ambiguities in the above because the column signs of $\boldsymbol{G}$ can be freely chosen. For identifiability we may wish to impose further constraints on $\boldsymbol{Q}_2^{\text{PCA-Cor}}$ or equivalently on $\mathbf{\Psi}^{\text{PCA-Cor}}$. A useful condition is to require (for the given ordering of the original variables!) that the diagonal elements of $\boldsymbol{Q}_2^{\text{PCA-Cor}}$ are all positive or equivalently that $\mathbf{\Psi}^{\text{PCA-Cor}}$ has a positive diagonal. This implies that $\text{Diag}(\boldsymbol{G}) > 0$ and $\text{Diag}(\mathbf{\Phi}^{\text{PCA-Cor}}) > 0$.

Note that the actual objective of PCA-cor whitening $\text{Diag}(\mathbf{\Psi}^T\mathbf{\Psi})$ is not affected by the sign ambiguity since the column signs of $\mathbf{\Psi}$ do not matter.

Proportion of total variation:

In PCA-cor whitening the contribution $k_i^{\text{PCA-Cor}}$ of each latent component z_i to the total variation based on the correlation $\text{Tr}(\boldsymbol{P}) = d$ is $k_i^{\text{PCA-Cor}} = \theta_i$. The fraction $\frac{\theta_i}{d}$ is the relative contribution of each element in $\boldsymbol{z}$ to explain the total variation.

Summary:

- PCA-cor whitening is a whitening transformation that maximises compression with the sum of squared cross-correlations as underlying optimality criterion.
- There are sign ambiguities in the PCA-cor whitened variables which are inherited from the sign ambiguities in the eigenvectors.
- If a positive-diagonal condition on the orthogonal matrices is imposed then these sign ambiguities are fully resolved and corresponding components z_i and x_i are always positively correlated.
- If $\boldsymbol{x}$ is standardised to $\text{Var}(x_i) = 1$, then PCA and PCA-cor whitening are identical.

2.4.5 Cholesky whitening

Aim in Cholesky whitening:

Find a whitening transformation such that the cross-covariance $\mathbf{\Phi}$ and cross-correlation $\mathbf{\Psi}$ have lower triangular structure. Specifically, we wish that the original variable x_1 is linked with the first latent variable z_1 only, the second

original variable x_2 is linked to z_1 and z_2 only, and so on, and the last variable x_d is linked with all latent variables $z_1, \ldots, z_d$:

$$\begin{array}{cc} x_1 & \to z_1 \\ x_1, x_2 & \to z_2 \\ \vdots & \\ x_1, x_2, \ldots, x_d & \to z_d \end{array}$$

Thus, Cholesky whitening imposes a structural (sparsity!) constraint on the loadings, where the non-zero coefficients are all in the lower half whereas in the upper half the coefficients all vanish.

Cholesky matrix decomposition:

In order to find such a whitening transformation we use the Cholesky decomposition

The Cholesky decomposition of a square matrix $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$ requires a positive definite $\boldsymbol{A}$ and is unique. $\boldsymbol{L}$ is a lower triangular matrix with positive diagonal elements. Its inverse $\boldsymbol{L}^{-1}$ is also lower triangular with positive diagonal elements. If $\boldsymbol{D}$ is a diagonal matrix with positive elements then $\boldsymbol{D}\boldsymbol{L}$ is also a lower triangular matrix with a positive diagonal and the Cholesky factor for the matrix $\boldsymbol{D}\boldsymbol{A}\boldsymbol{D}$.

Solution: Apply a Cholesky decomposition to $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$

The resulting whitening matrix is

$$\boldsymbol{W}^{\text{Chol}} = \boldsymbol{L}^{-1}$$

By construction, $\boldsymbol{W}^{\text{Chol}}$ is a lower triangular matrix with positive diagonal. The whitening constraint is satisfied as $(\boldsymbol{W}^{\text{Chol}})^T\boldsymbol{W}^{\text{Chol}} = (\boldsymbol{L}^{-1})^T\boldsymbol{L}^{-1} = (\boldsymbol{L}^T)^{-1}\boldsymbol{L}^{-1} = (\boldsymbol{L}\boldsymbol{L}^T)^{-1} = \boldsymbol{\Sigma}^{-1}$.

The cross-covariance matrix is the inverse of the whitening matrix

$$\boldsymbol{\Phi}^{\text{Chol}} = \boldsymbol{L}$$

and the cross-correlation matrix is

$$\boldsymbol{\Psi}^{\text{Chol}} = \boldsymbol{V}^{-1/2}\boldsymbol{L}$$

Both $\boldsymbol{\Phi}^{\text{Chol}}$ and $\boldsymbol{\Psi}^{\text{Chol}}$ are lower triangular matrices with positive diagonal elements. Hence two corresponding components x_i and z_i are always positively correlated!

Finally, the corresponding orthogonal matrices are

$$\boldsymbol{Q}_1^{\text{Chol}} = \boldsymbol{\Phi}^T\boldsymbol{\Sigma}^{-1/2} = \boldsymbol{L}^T\boldsymbol{\Sigma}^{-1/2}$$

and

$$\boldsymbol{Q}_2^{\text{Chol}} = \boldsymbol{\Psi}^T \boldsymbol{P}^{-1/2} = \boldsymbol{L}^T \boldsymbol{V}^{-1/2} \boldsymbol{P}^{-1/2}$$

Application to correlation instead of covariance:

We may also apply the Cholesky decomposition to the correlation rather than the covariance matrix. However, this does *not* lead to a different whitening transform (unlike for ZCA and PCA):

The Cholesky factor for $\boldsymbol{P} = \boldsymbol{V}^{-1/2} \boldsymbol{\Sigma} \boldsymbol{V}^{-1/2}$ is $\boldsymbol{V}^{-1/2} \boldsymbol{L}$. The corresponding whitening matrix is $(\boldsymbol{V}^{-1/2} \boldsymbol{L})^{-1} \boldsymbol{V}^{-1/2} = \boldsymbol{L}^{-1} = \boldsymbol{W}^{\text{Chol}}$.

This is also intuively clear as the covariance and correlation loadings are closely linked, in particular they share the same triangular shape.

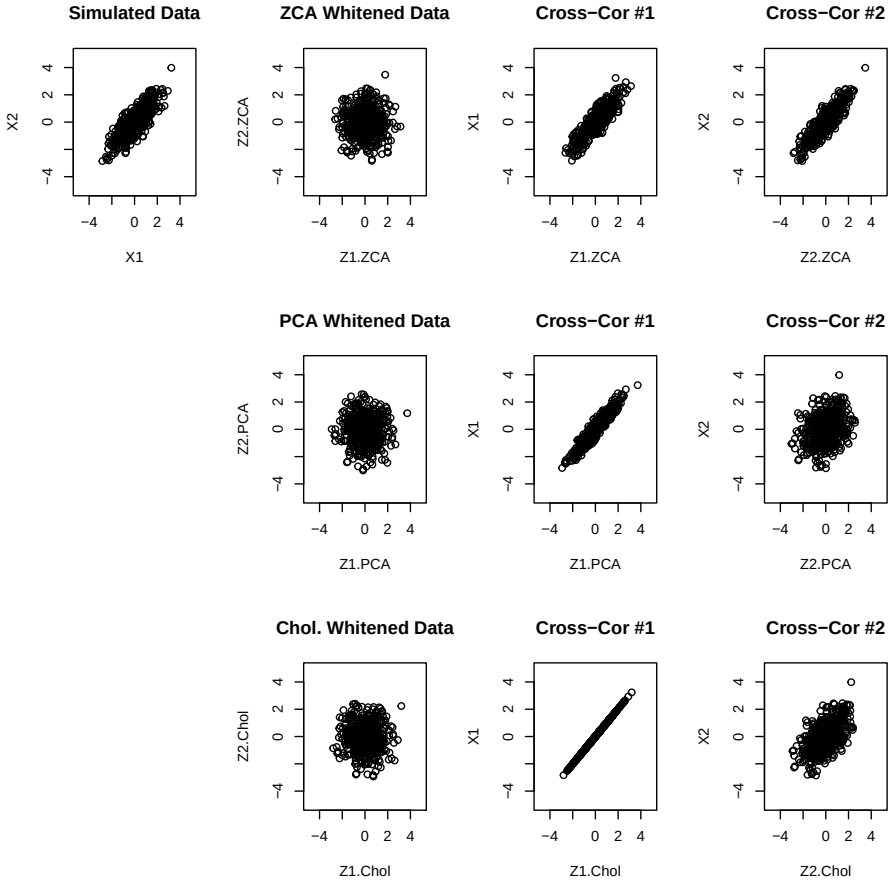
Dependence on the input order:

Cholesky whitening depends on the ordering of input variables. Each ordering of the original variables will yield a different triangular constraint and thus a different Cholesky whitening transform. For example, by inverting the ordering to $x_d, x_{d-1}, \ldots, x_1$ we effectively enforce an upper triangular shape.

An alternative formulation of Cholesky whitening decomposes the precision matrix rather than the covariance matrix. This yields the upper triangular structure directly and is otherwise fully equivalent to Cholesky whitening based on decomposing the covariance.

2.4.6 Comparison of whitening procedures - simulated data

For comparison, here are the results of ZCA, PCA and Cholesky whitening applied to a simulated bivariate normal data set with correlation $\rho = 0.8$.



In column 1 you can see the simulated data as scatter plot.

Column 2 shows the scatter plots of the whitened data — as expect all three methods remove correlation and produce an isotropic covariance.

However, the three approaches differ in the cross-correlations. Columns 3 and 4 show the cross-correlations between the first two corresponding components (x_1 and z_1, and x_2 and z_2) for ZCA, PCA and Cholesky whitening. As expected, in ZCA both pairs show strong correlation, but this is not the case for PCA and Cholesky whitening.

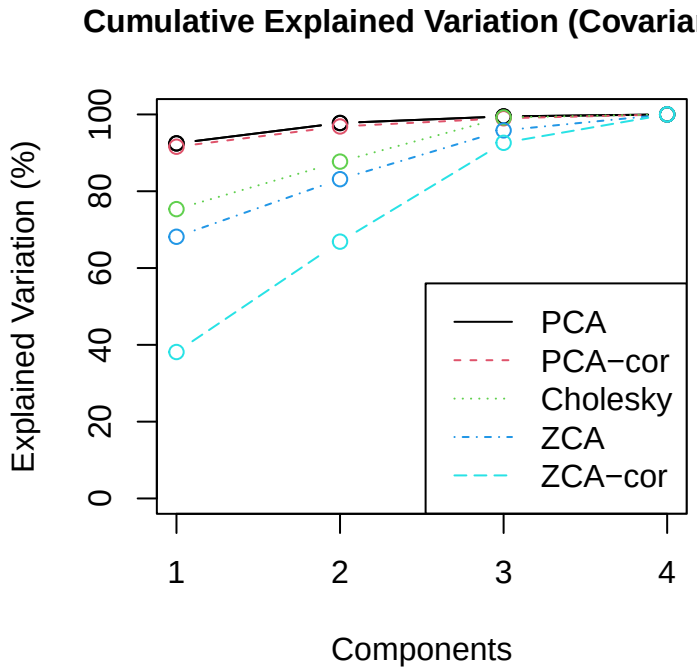
Note that for Cholesky whitening the first component z_1 is perfectly positively correlated with the original component x_1 because the whitening matrix is lower triangular with a positive diagonal and hence z_1 is just x_1 multiplied with a positive constant.

2.4.7 Comparison of whitening procedures - iris flowers

As an example we consider the well known iris flower data set. It consists of botanical measures (sepal length, sepal width, petal length and petal width) for 150 iris flowers comprising three species (Iris setosa, Iris versicolor, Iris virginica). Hence this data set has dimension $d = 4$ and sample size $n = 150$.

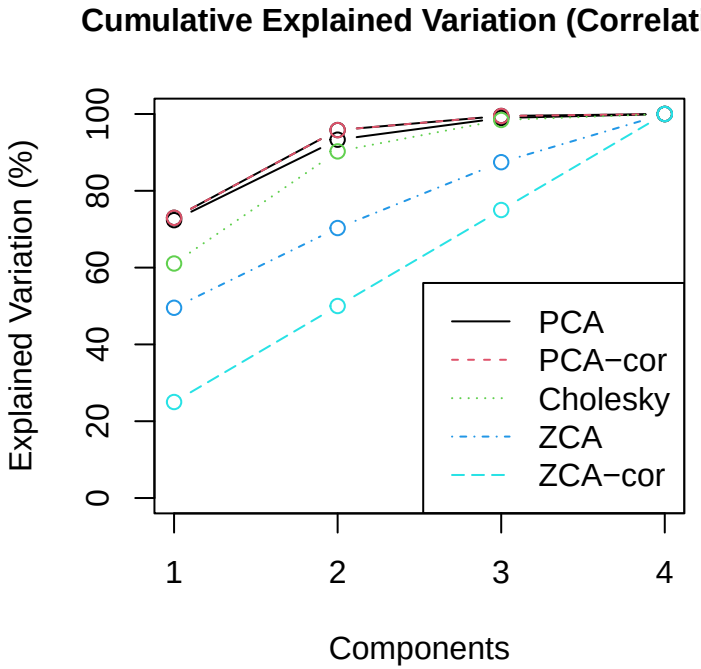
We apply all discussed whitening transforms to this data, and then sort the whitened components by their relative contribution to the total variation. For Cholesky whitening we used the input order for the shape constraint.

Here are the results for explained variation based on covariance loadings:



As expected, the two PCA whitening approaches compress the data most. On the other end of the spectrum, the ZCA whitening methods are the two least compressing approaches. Cholesky whitening is a compromise between ZCA and PCA in terms of compression.

Similar results are obtained based on correlation loadings - note how ZCA-cor provides equal weight for each latent variable.



2.4.8 Recap

Method	Type of usage
ZCA, ZCA-cor:	pure decorrelate, maintain similarity to original data set, interpretability
PCA, PCA-cor:	compression, find effective dimension, reduce dimensionality, feature identification
Cholesky	triangular shaped W, $\mathbf{\Phi}$ and $\mathbf{\Psi}$, sparsity

If data are standardised then $\mathbf{\Phi}$ and $\mathbf{\Psi}$ will be the same and hence ZCA will become ZCA-cor and PCA becomes PCA-cor. The triangular shape constraint of Cholesky whitening depends on the ordering of the original variables.

Related methods not discussed in this course:

- Factor models: essentially whitening plus an additional error term, factors have rotational freedom just like in whitening
- Partial Least Squares (PLS): similar to Principal Components Analysis (PCA) but in a regression setting (with the choice of latent variables

depending on the response)

- Nonlinear dimension reduction methods such as SNE, tSNE, UMAP.

2.5 Principal Component Analysis (PCA)

2.5.1 PCA transformation

Principal component analysis was proposed in 1933 by Harald Hotelling[5] and is very closely related to **PCA whitening**. The underlying mathematics was developed earlier in 1901 by Karl Pearson[6] for the problem of orthogonal regression.

Assume random vector $\boldsymbol{x}$ with $\text{Var}(\boldsymbol{x}) = \boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$. PCA is a particular orthogonal transformation of the original $\boldsymbol{x}$ such that the resulting components are orthogonal:

$$\underbrace{\boldsymbol{t}^{\text{PCA}}}_{\text{Principal components}} = \underbrace{\boldsymbol{U}^T}_{\text{Orthogonal matrix}} \boldsymbol{x}$$

$$\text{Var}(\boldsymbol{t}^{\text{PCA}}) = \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_d \end{pmatrix}$$

Note that while principal components are *orthogonal* they do *not* have unit variance but the variance of principal components t_i equals the eigenvalues λ_i.

Thus PCA itself is *not* a whitening procedure but it is very closely linked to PCA whitening which is obtained by standardising the principal components: $\boldsymbol{z}^{\text{PCA}} = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{t}^{\text{PCA}}$

Compression properties:

The total variation is $\text{Tr}(\text{Var}(\boldsymbol{t}^{\text{PCA}})) = \text{Tr}(\boldsymbol{\Lambda}) = \sum_{j=1}^{d} \lambda_j$. With principle components the fraction $\frac{\lambda_i}{\sum_{j=1}^{d} \lambda_j}$ can be interpreted as the proportion of variation contributed by each component in $\boldsymbol{t}^{\text{PCA}}$ to the total variation. Thus, low ranking components in $\boldsymbol{t}^{\text{PCA}}$ with low variation may be discarded, thus leading to a reduction in dimension.

[5] Hotelling, H. 1933. Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology **24**:417–441 (Part 1) and **24**:498–520 (Part 2). https://doi.org/10.1037/h0071325 and https://doi.org/10.1037/h0070888

[6] Pearson, K. 1901. On lines and planes of closest fit to systems of points in space. Philosophical Magazine **2**:559–572. https://doi.org/10.1080/14786440109462720

2.5.2 Application to data

Written in terms of a data matrix $\boldsymbol{X}$ instead of a random vector $\boldsymbol{x}$ PCA becomes:

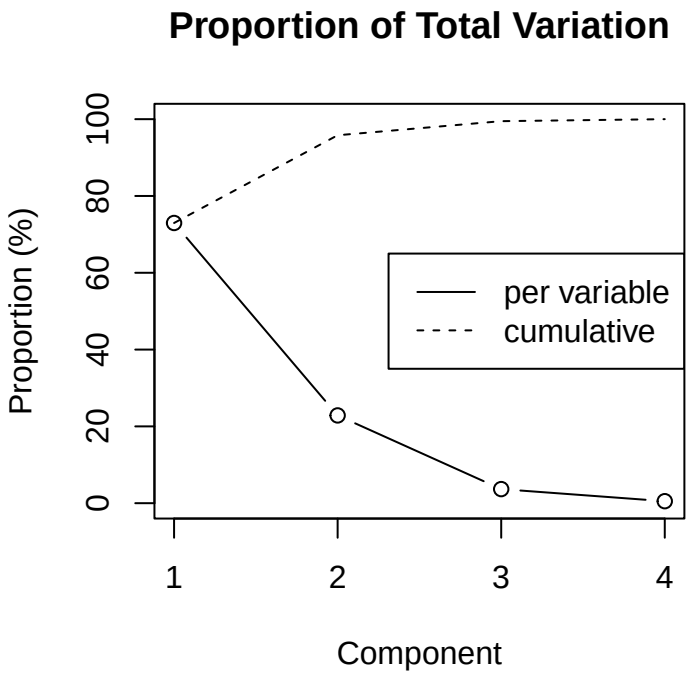
$$\underbrace{\boldsymbol{T}}_{\text{Sample version of principal components}} = \underbrace{\boldsymbol{X}}_{\text{Data matrix}} \boldsymbol{U}$$

There are now two ways to obtain $\boldsymbol{U}$:

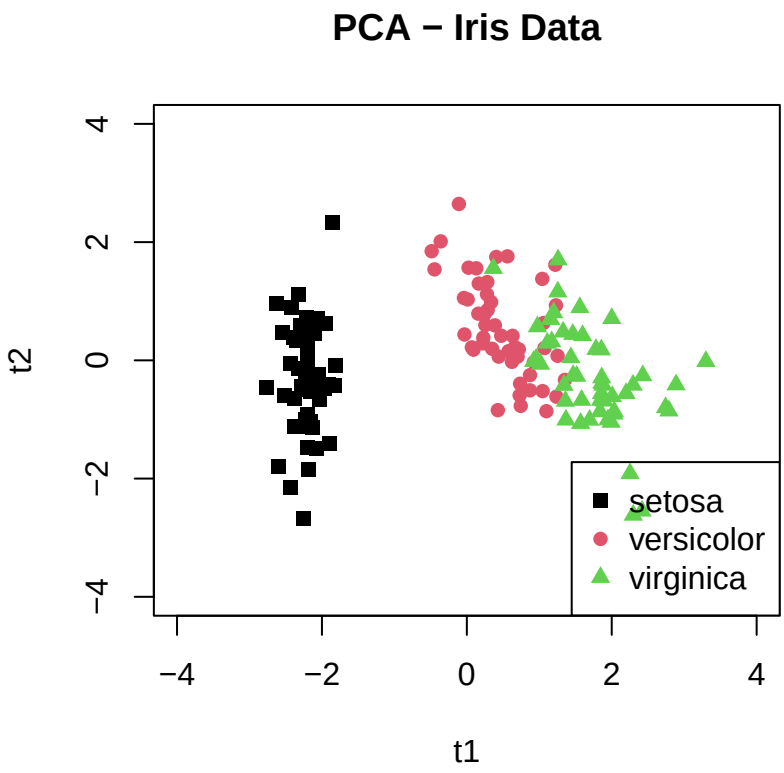
1. Estimate the covariance matrix, e.g. by $\hat{\boldsymbol{\Sigma}} = \frac{1}{n}\boldsymbol{X}_c^T\boldsymbol{X}_c$ where $\boldsymbol{X}_c$ is the column-centred data matrix; then apply the eigenvalue decomposition on $\hat{\boldsymbol{\Sigma}}$ to get $\boldsymbol{U}$.
2. Compute the singular value decomposition of $\boldsymbol{X}_c = \boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^T$. As $\hat{\boldsymbol{\Sigma}} = \frac{1}{n}\boldsymbol{X}_c^T\boldsymbol{X}_c = \boldsymbol{U}(\frac{1}{n}\boldsymbol{D}^2)\boldsymbol{U}^T$ you can just use $\boldsymbol{U}$ from the SVD of $\boldsymbol{X}_c$ and there is no need to compute the covariance.

2.5.3 Iris flower data example

We first standardise the data, then compute PCA components and plot the proportion of total variation contributed by each component. This shows that only two PCA components are needed to achieve 95% of the total variation:



A scatter plot plot of the the first two principal components is also informative:



This shows that there groupings among the 150 flowers, corresponding to the species, and that these groups can be characterised by the the principal components.

2.5.4 PCA correlation loadings

In an earlier section we have learned that for a general whitening transformation the cross-correlations $\boldsymbol{\Psi} = \mathrm{Cor}(\boldsymbol{x}, \boldsymbol{z})$ play the role of correlation loadings in the inverse transformation:

$$\boldsymbol{V}^{-1/2}\boldsymbol{x} = \boldsymbol{\Psi}\boldsymbol{z}\,,$$

i.e. they are the coefficients linking the whitened variable $\boldsymbol{z}$ with the standardised original variable $\boldsymbol{x}$. This relationship holds therefore also for PCA-whitening with $\boldsymbol{z}^{\mathrm{PCA}} = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T\boldsymbol{x}$ and $\boldsymbol{\Psi}^{\mathrm{PCA}} = \boldsymbol{V}^{-1/2}\boldsymbol{U}\boldsymbol{\Lambda}^{1/2}$.

The classical PCA is not a whitening approach because $\mathrm{Var}(\boldsymbol{t}^{\mathrm{PCA}}) \neq \boldsymbol{I}$. However, we can still compute cross-correlations between $\boldsymbol{x}$ and the principal components

$\boldsymbol{t}^{\text{PCA}}$, resulting in

$$\text{Cor}(\boldsymbol{x}, \boldsymbol{t}^{\text{PCA}}) = \boldsymbol{V}^{-1/2}\boldsymbol{U}\boldsymbol{\Lambda}^{1/2} = \boldsymbol{\Psi}^{\text{PCA}}$$

Note these are the same as the cross-correlations for PCA-whitening since $\boldsymbol{t}^{\text{PCA}}$ and $\boldsymbol{z}^{\text{PCA}}$ only differ in scale.

The inverse PCA transformation is

$$\boldsymbol{x} = \boldsymbol{U}\boldsymbol{t}^{\text{PCA}}$$

In terms of standardised PCA components $z^{\text{PCA}} = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{t}^{\text{PCA}}$ and standardised original components it becomes

$$\boldsymbol{V}^{-1/2}\boldsymbol{x} = \boldsymbol{\Psi}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{t}^{\text{PCA}}$$

Thus the cross-correlation matrix $\boldsymbol{\Psi}$ plays the role of *correlation loadings* also in classical PCA, i.e. they are the coefficients linking the standardised PCA components with the standardised original components.

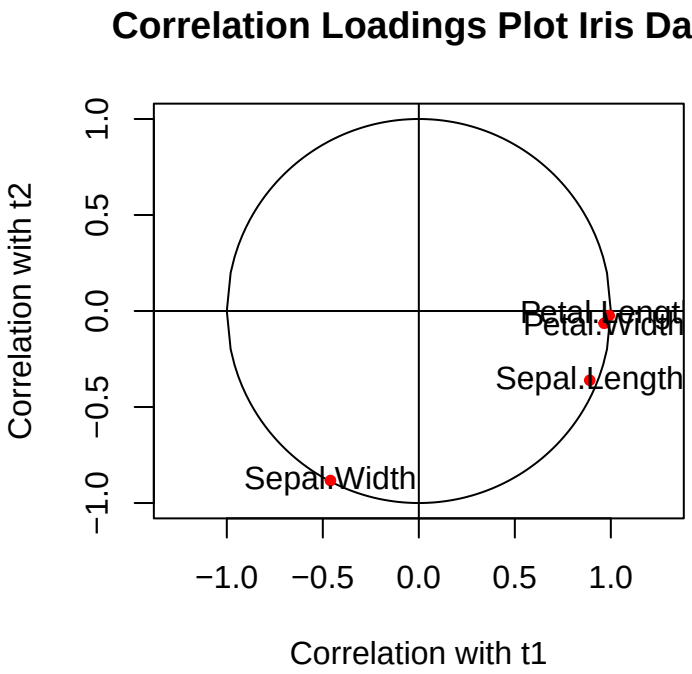
2.5.5 PCA correlation loadings plot

In PCA and PCA-cor whitening as well as in classical PCA the aim is compression, i.e. to find latent variables such that most of the total variation is contributed by a small number of components.

In order to be able to better interpret the top ranking PCA component we can use a visual device called *correlation loadings plot*. For this we compute the correlation between the PCA components 1 and 2 (t_1^{PCA} and t_2^{PCA}) with all original variables $x_1, \ldots, x_d$.

For each original variable x_i we therefore have two numbers between -1 and 1, the correlation $\text{Cor}(x_i, t_1^{\text{PCA}}) = \psi_{i1}$ and $\text{Cor}(x_i, t_2^{\text{PCA}}) = \psi_{i2}$ that we use as coordinates to draw a point in a plane. Recall that the row sums of squares of the correlation loadings $\boldsymbol{\Psi}$ are all identical to 1. Hence, the sum of the squared loadings from just the first two components is also at most 1. Thus, by construction, all points have to lie within a unit circle around the origin.
The original variables most strongly influenced by the two latent variables will have strong correlation and thus lie near the outer circle, whereas variables that are not influenced by the two latent variables will lie near the origin.

As an example, here is the correlation loadings plot showing the cross-correlation between the first two PCA components and all four variables of the iris flower data set discussed earlier.



The interpretation of this plot is discussed in Worksheet 5.

Chapter 3

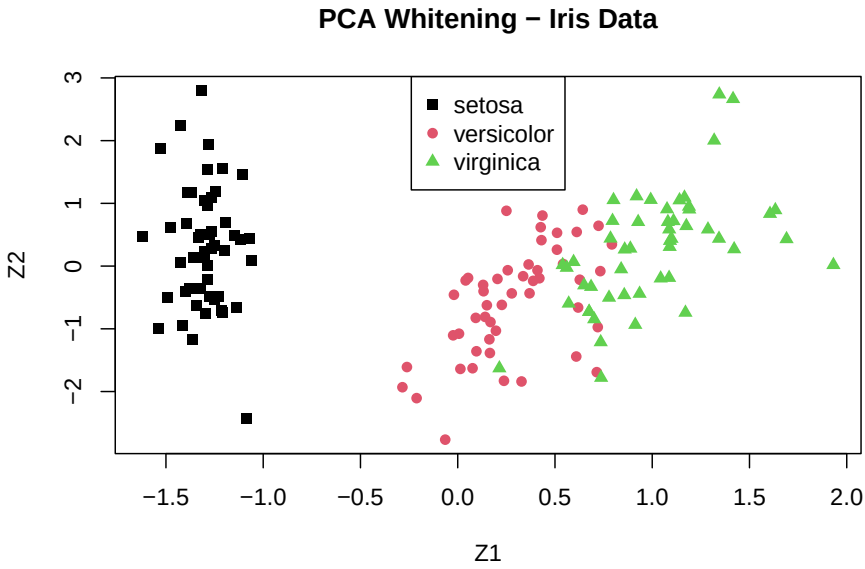
Unsupervised learning and clustering

3.1 Challenges in unsupervised learning

3.1.1 Objective

We observe data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ for n objects (or subjects). Each sample $\boldsymbol{x}_i$ is a vector of dimension d. Thus, for each of the n objects / subjects we have measurements on d variables. The aim of unsupervised learning is to identify patters relating the objects/subjects based on the information available in $\boldsymbol{x}_i$. Note that in unsupervised learning we use *only* the information in $\boldsymbol{x}_i$ and nothing else.

For illustration consider the first two principal components of the Iris flower data (see e.g. Worksheet 5):



Clearly there is a group structure among the samples that is linked to particular patterns in the first two principal components.

Note that in this plot we have used additional information, the class labels (setosa, versicolor, virginica), to highlighting the true underlying structure (the three flower species).

In **unsupervised learning** the class labels are (assumed to be) unknown, and the aim is to infer the clustering and thus the classes labels.[1]

There are many methods for clustering and unsupervise learning, both purely algorithmic as well as probabilistic. In this chapter we will study a few of the most commonly used approaches.

3.1.2 Questions and problems

In order to implement unsupervised learning we need to address a number of questions:

- how do we define clusters?
- how do we learn / infer clusters?
- how many clusters are there? (this is surprisingly difficult!)
- how can we assess the uncertainty of clusters?

Once we know the clusters we are also interested in:

[1] In contrast, in **supervised learning** (to be discussed in a subsequent chapter) the class labels are known for a subset of the data (the training data set) and are required to learn a prediction function.

- which features define / separate each cluster?

(note this is a feature / variable selection problem, discussed in in supervised learning).

Many of these problems and questions are highly specific to the data at hand. Correspondingly, there are many different types of methods and models for clustering and unsupervised learning.

In terms of representing the data, unsupervised learning tries to balance between the following two extremes:

1) all objects are grouped into a single cluster (low complexity model)
2) all objects are put into their own cluster (high complexity model)

In practise, the aim is to find a compromise, i.e. a model that captures the structure in the data with appropriate complexity — not too low and not too complex.

3.1.3 Why is clustering difficult?

Partioning problem (combinatorics): How many partitions of n objects (say flowers) into K groups (say species) exists?

Answer:

$$S(n, K) = \left\{ \begin{array}{c} n \\ K \end{array} \right\}$$

this is the "Sterling number of the second type".

For large n:

$$S(n, K) \approx \frac{K^n}{K!}$$

Example:

n	K	Number of possible partitions
15	3	$\approx$ 2.4 million (10^6)
20	4	$\approx$ 2.4 billion (10^9)
$\vdots$		
100	5	$\approx 6.6 \times 10^{76}$

These are enormously big numbers even for relatively small problems!

$\Longrightarrow$ Clustering / partitioning / structure discovery is not easy!

$\Longrightarrow$ We cannot expect perfect answers or a single "true" clustering

In fact, as a model of the data many differnt clusterings may fit the data equally well.

$\Longrightarrow$ We need to assesse the uncertainty of the clustering

This can be done as part of probabilistic modelling or by resampling (e.g., bootstrap).

3.1.4 Common types of clustering methods

There are very many different clustering algorithms!

We consider the following two broad types of methods:

1) **Algorithmic clustering methods** (these are not explicitly based on a probabilistic model)

- K-means
- PAM
- hierarchical clustering (distance or similarity-based, divise and agglomerative)

  pros: fast, effective algorithms to find at least some grouping **cons:** no probabilistic interpretation, blackbox methods

2) **Model-based clustering** (based on a probabilistic model)

- mixture models (e.g. Gaussian mixture models, GMMs, non-hierarchical)
- graphical models (e.g. Bayesian networks, Gaussian graphical models GGM, trees and networks)

  pros: full probabilistic model with all corresponding advantages
  cons: computationally very expensive, sometimes impossible to compute exactly.

3.2 Hierarchical clustering

3.2.1 Tree-like structures

Often, categorisations of objects are nested, i.e. there sub-categories of categories etc. These can be naturally represented by **tree-like hierarchical structures**.

In many branches of science hierarchical clusterings are widely employed, for example in evolutionary biology: see e.g.

- Tree of Life explaining the biodiversity of life
- phylogenetic trees among species (e.g. vertebrata)
- population genetic trees to describe human evolution
- taxonomic trees for plant species
- etc.

Note that when visualising hierarchical structures typically the corresponding tree is depicted facing downwards, i.e. the root of the tree is shown on the top, and the tips/leaves of the tree are shown at the bottom!

In order to obtain such a hierarchical clustering from data two opposing strategies are commonly used:

1) **divisive or recursive partitioning algorithms**
   - grow the tree from the root downwards
   - first determine the main two clusters, then recursively refine the clusters further.
2) **agglomerative algorithms**
   - grow the tree from the leaves upwards
   - successively form partitions by first joining individual object together, then recursively join groups of items together, until all is merged.

In the following we discuss a number of popular hierarchical agglomerative clustering algorithms that are based on the pairwise distances / similarities (a $n \times n$ matrix) among all data points.

3.2.2 Agglomerative hierarchical clustering algorithms

A general algorithm for agglomerative construction of a hierarchical clustering works as follows:

Initialisation:

Compute a dissimilarity / distance matrix between all pairs of objects where "objects" are single data points at this stage but later are also be sets of data points.

Iterative procedure:

1) identify the pair of objects with the smallest distance. These two objects are then merged together into one set. Create an internal node in the tree to represent this set.
2) update the distance matrix by computing the distances between the new set and all other objects. If the new set contains all data points the procedure terminates. The final node created is the root node.

For actual implementation of this algorithm two key ingredients are needed:

1) a distance measure $d(\boldsymbol{a}, \boldsymbol{b})$ between two individual elementary data points $\boldsymbol{a}$ and $\boldsymbol{b}$.

   This is typically one of the following:

   - Euclidean distance $d(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{d}(a_i - b_i)^2} = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^T(\boldsymbol{a} - \boldsymbol{b})}$
   - Squared Euclidean distance $d(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a} - \boldsymbol{b})^T(\boldsymbol{a} - \boldsymbol{b})$
   - Manhattan distance $d(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{d} |a_i - b_i|$

- Maximum norm $d(\boldsymbol{a}, \boldsymbol{b}) = \max_{i \in \{1,\ldots,d\}} |a_i - b_i|$

In the end, making the correct choice of distance will require subject knowledge about the data!

2) a distance measure $d(A, B)$ between two sets of objects $A = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_{n_A}\}$ and $B = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_{n_B}\}$ of size n_A and n_B, respectively.

To determine the distance $d(A, B)$ between these two sets the following measures are often employed:

- **complete linkage** (max. distance): $d(A, B) = \max_{\boldsymbol{a}_i \in A, \boldsymbol{b}_i \in B} d(\boldsymbol{a}_i, \boldsymbol{b}_i)$
- **single linkage** (min. distance): $d(A, B) = \min_{\boldsymbol{a}_i \in A, \boldsymbol{b}_i \in B} d(\boldsymbol{a}_i, \boldsymbol{b}_i)$
- **average linkage** (avg. distance): $d(A, B) = \frac{1}{n_A n_B} \sum_{\boldsymbol{a}_i \in A} \sum_{\boldsymbol{b}_i \in B} d(\boldsymbol{a}_i, \boldsymbol{b}_i)$

3.2.3 Ward's clustering method

Another agglomerative hierarchical procedure is **Ward's minimum variance approach**[2] (see also[3]). In this approach in each iteration the two sets A and B are merged that lead to the **smallest increase in within-group variation**. The centroids of the two sets is given by $\boldsymbol{\mu}_A = \frac{1}{n_A} \sum_{\boldsymbol{a}_i \in A} \boldsymbol{a}_i$ and $\boldsymbol{\mu}_B = \frac{1}{n_B} \sum_{\boldsymbol{b}_i \in B} \boldsymbol{b}_i$.

The within-group sum of squares for group A is

$$w_A = \sum_{\boldsymbol{a}_i \in A} (\boldsymbol{a}_i - \boldsymbol{\mu}_A)^T (\boldsymbol{a}_i - \boldsymbol{\mu}_A)$$

and is computed here on the basis of the difference of the observations $\boldsymbol{a}_i$ relative to their mean $\boldsymbol{\mu}_A$. However, since we typically only have pairwise distances available we don't know the group means so this formula can't be applied. Fortunately, it is also possible to compute w_A using only the pairwise differences using

$$w_A = \frac{1}{n_A} \sum_{\boldsymbol{a}_i, \boldsymbol{a}_j \in A, i<j} (\boldsymbol{a}_i - \boldsymbol{a}_j)^T (\boldsymbol{a}_i - \boldsymbol{a}_j)$$

This trick is employed in Ward's clustering method by constructing a distance measure between two sets A and B as

$$d(A, B) = w_{A \cup B} - w_A - w_B$$

and using as the distance between two elementary data points $\boldsymbol{a}$ and $\boldsymbol{b}$ the squared Euclidean distance

$$d(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a} - \boldsymbol{b})^T (\boldsymbol{a} - \boldsymbol{b}) .$$

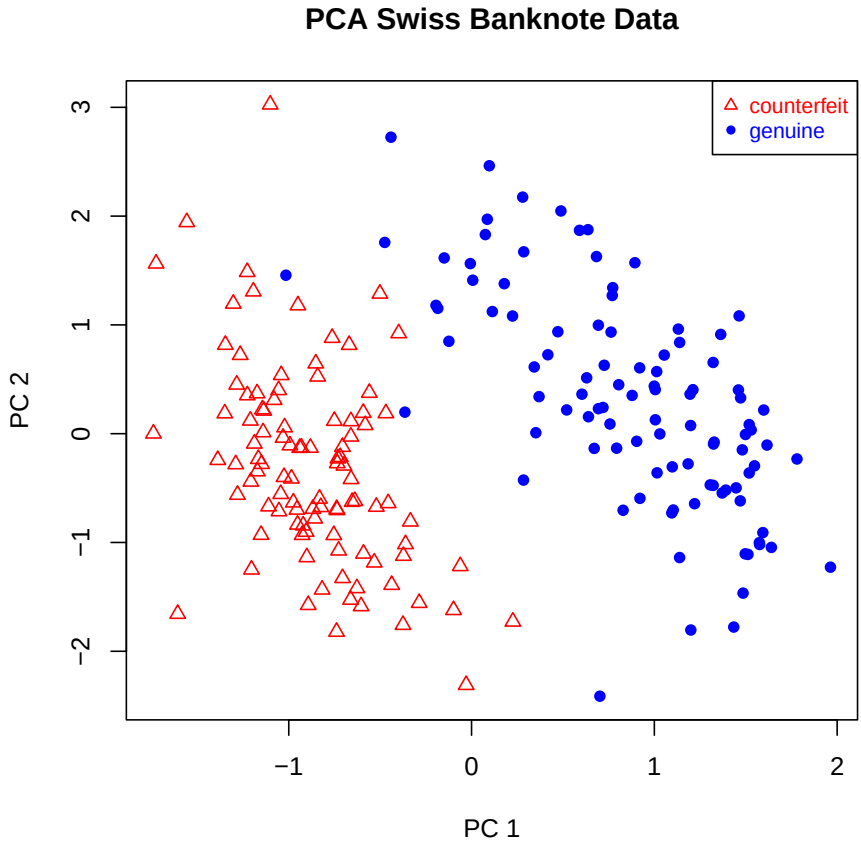
[2] Ward, J.H. 1963. Hierarchical grouping to optimize an objective function. JASA **58**:236–244. https://doi.org/10.1080/01621459.1963.10500845

[3] F. Murtagh and P Legendre. 2014. Ward's hierarchical agglomerative clustering method: which algorithms implement Ward's criterion? J. Classif. *31*:274–295. https://doi.org/10.1007/s00357-014-9161-z

3.2.4 Application to Swiss banknote data set

This data set is reports 6 pysical measurements on 200 Swiss bank notes. Of the 200 notes 100 are genuine and 100 are counterfeit. The measurements are: length, left width, right width, bottom margin, top margin, diagonal length of the bank notes.

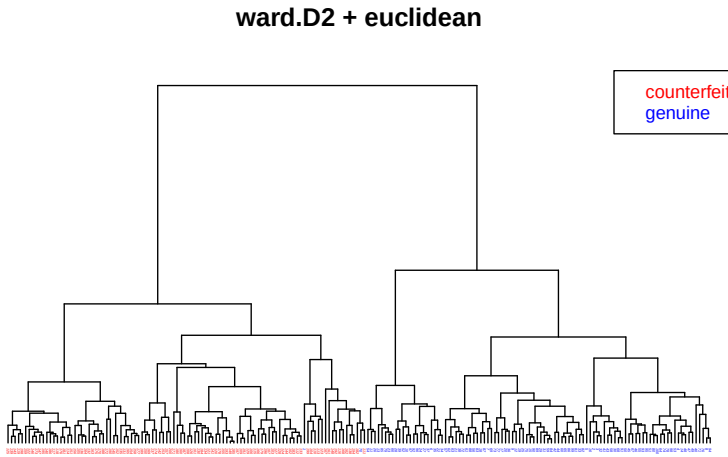
Plotting the first to PCAs of this data shows that there are indeed two well defined groups, and that these groups correspond precisely to the genuine and counterfeit banknotes:



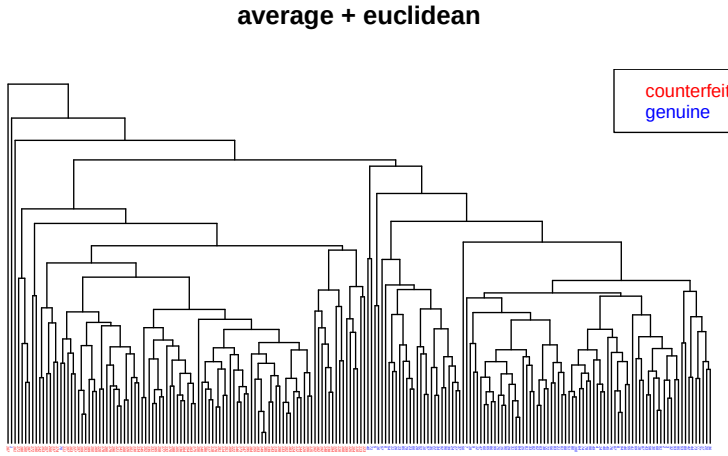
We now compare the hierarchical clusterings of the Swiss bank note data using four different methods using Euclidean distance.

An interactive R Shiny web app of this analysis (which also allows to explore further distance measures) is available online at https://minerva.it.manchester.ac.uk/shiny/strimmer/hclust/ .

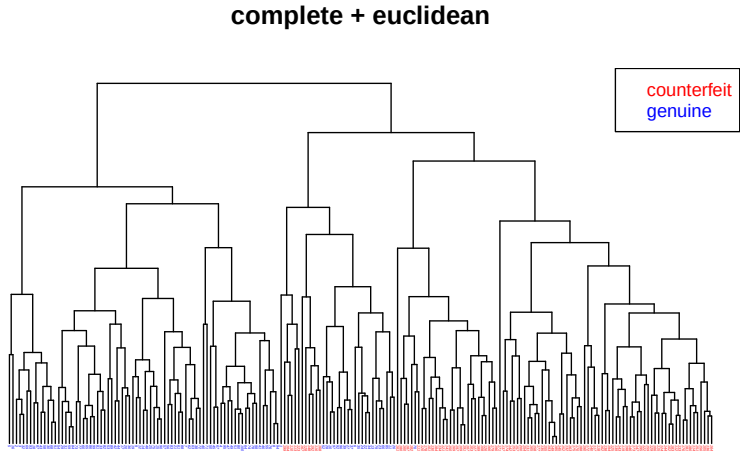
Ward.D2 (=Ward's method):



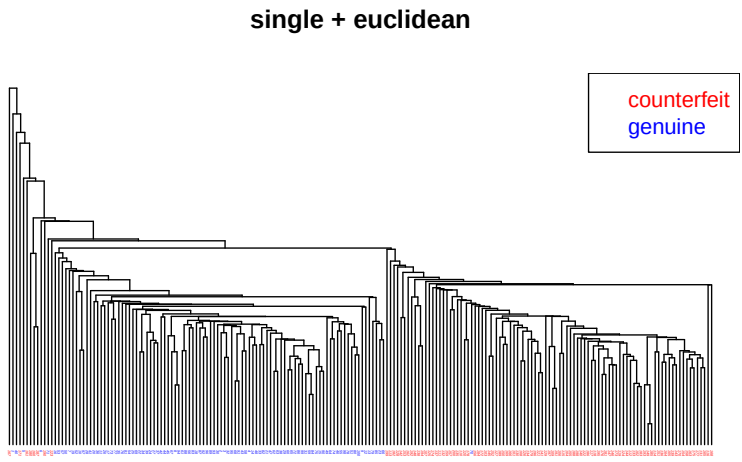
Average linkage:



Complete linkage:



Single linkage:



Result:

- All four trees / hierarchical clusterings are quite different!
- The Ward.D2 method is the only one that finds the correct grouping (except for a single error).

3.2.5 Assessment of the uncertainty of hierarchical clusterings

In practical application of hierarchical clustering methods is is essential to evaluate the stability and uncertainty of the obtained groupings. This is often done as follows using the "bootstrap":

- Sampling with replacement is used to generate a number of so-called bootstrap data sets (say $B = 200$) similar to the original one. Specifically, we create new data matrices by repeatedly randomly selecting columns (variables) from the original data matrix for inclusion in the bootstrap data matrix. Note that we sample columns as our aim is to cluster the samples.
- Subsequently, a hierarchical clustering is computed for each of the bootstrap data sets. As a result, we now have an "ensemble" of B bootstrap trees.
- Finally, analysis of the clusters (bipartions) shown in all the bootstrap trees allows to count the clusters that appear frequently, and also those that appear less frequently. These counts provide a measure of the stability of the clusterings appearing in the original tree.
- Additionally, from the bootstrap tree we can also compute a consensus tree containing the most stable clusters. This an be viewed as an "ensemble average" of all the bootstrap trees.

A disadvantage of this procedure is that bootstrapping trees is computationally very expensive, as the original procedure is already time consuming but now needs to be repeated a large number of times.

3.3 K-means clustering

3.3.1 Set-up

- We assume that there are K groups (i.e. K is known in advance).
- For each group $k \in \{1, \ldots, K\}$ we assume a group mean $\boldsymbol{\mu}_k$.
- Aim: partition the data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ into K non-overlapping groups.
- Each of the n data points $\boldsymbol{x}_i$ is assigned to exactly one of the K groups.
- Maximise the homogeneity within each group (i.e. each group should contain similar objects).
- Maximise the heterogeneity between the different groups (i.e each group should differ from the other groups).

3.3.2 Algorithm

After running K-means we will get estimates of $\hat{\boldsymbol{\mu}}_k$ of the group means, as well allocations $y_i \in \{1, \ldots, K\}$ of each data point $\boldsymbol{x}_i$ to one of the classes.

Initialisation:

At the start of the algorithm the n observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are randomly allocated with equal probability to one of the K groups. The resulting assignment is $y_1, \ldots, y_n$, with each $y_i = \{1, \ldots, K\}$. With $G_k = \{i | y_i = k\}$ we denote the set of indices of the data points in cluster k, and with $n_k = |G_k|$ the number of samples in cluster k.

Iterative refinement:

1) Estimate the group means by

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i \in G_k} \boldsymbol{x}_i$$

2) Update the group allocations y_i. Specifically, assign each data point $\boldsymbol{x}_i$ to the group k with the nearest $\hat{\boldsymbol{\mu}}_k$. The distance is measured in terms of the Euclidean norm:

$$\begin{aligned} y_i &= \arg\min_k \left| \boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k \right|_2 \\ &= \arg\min_k \left(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k \right)^T \left(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k \right) \end{aligned}$$

Steps 1 and 2 are repeated until the algorithm converges (i.e. until the group allocations don't change any more) or until a specified upper limit of iterations is reached.

3.3.3 Properties

K-means has been proposed in the 1950 to 1970s by various authors in diverse contexts.[4] Despite its simplicity K-means is, perhaps surprisingly, a very effective clustering algorithm. The main reason for this is the close connection of K-means with probabilistic clustering based on Gaussian mixture models (for details see later section).

Since the clustering depends on the initialisation it is often useful to run K-means several times with different starting group allocations of the data points. Furthermore, non-random or non-uniform initialisations can lead to improved and faster convergence, see the K-means++ algorithm.

The clusters constructed in K-means have linear boundaries and thus form a Voronoi tessellation around the cluster means. Again, this can be explained by the close link of K-means with a particular Gaussian mixture model.

[4] H.-H. Bock. 2008. Origins and extensions of the k-means algorithm in cluster analysis. JEHPS **4**, no. 2. https://www.jehps.net/Decembre2008/Bock.pdf

3.3.4 Choosing the number of clusters

Once the K-means algorithm has run we can assess the homogeneity and heterogeneity of the resulting clusters:

a) the total within-group sum of squares SSW (in R: `tot.withinss`), or total unexplained sum of squares:

$$SSW = \sum_{k=1}^{K} \sum_{i \in G_k} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^T (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)$$

This quantity decreases with K and is zero for $K = n$. The K-means algorithm tries to minimise this quantity but it will typically only find a local minimum rather than the global one.

b) the between-group sum of squares SSB (in R: `betweenss`), or explained sum of squares:

$$SSB = \sum_{k=1}^{K} n_k (\hat{\boldsymbol{\mu}}_k - \hat{\boldsymbol{\mu}}_0)^T (\hat{\boldsymbol{\mu}}_k - \hat{\boldsymbol{\mu}}_0)$$

where $\hat{\boldsymbol{\mu}}_0 = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i = \frac{1}{n} \sum_{k=1}^{K} n_k \hat{\boldsymbol{\mu}}_k$ is the global mean of the samples. SSB increases with the number of clusters K until for $K = n$ it becomes equal to the total sum of squares SST.

c) the total sum of squares

$$SST = \sum_{i=1}^{n} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_0)^T (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_0) \,.$$

By construction $SST = SSB + SSW$ for any K (i.e. SST is a constant independent of K).

Dividing the sum of squares by the sample size n we get

- $T = \frac{SST}{n}$ as the *total variation*,
- $B = \frac{SSW}{n}$ as the *explained variation* and
- $W = \frac{SSW}{n}$ as the total *unexplained variation* ,
- with $T = B + W$.

In order to decide on the optimal number of clusters we run K-means for different settings for K and then choose the smallest K for which the explained variation B is not significantly worse compared to a clustering with a substantially larger K (see example below).

3.3.5 K-medoids aka PAM

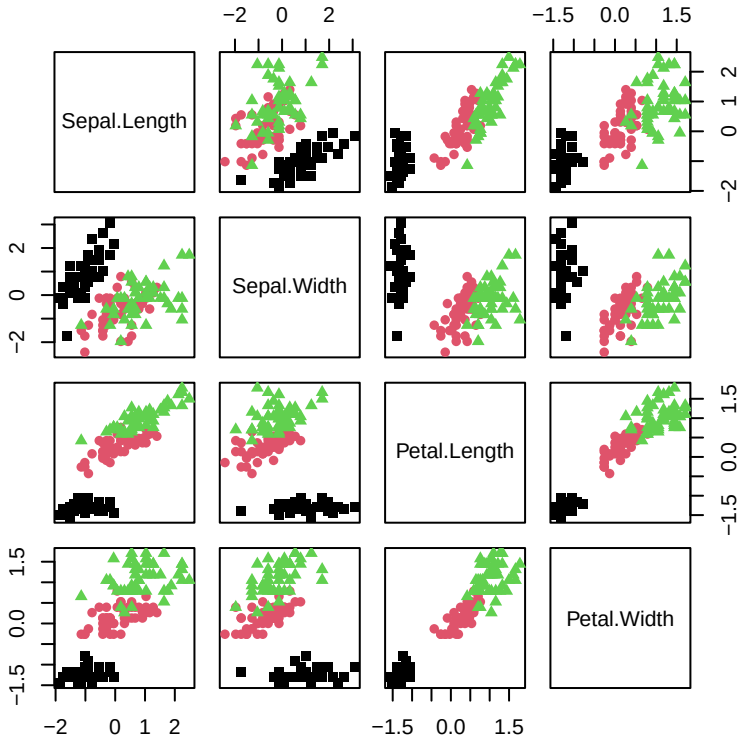
A closely related clustering method is K-medoids or PAM ("Partitioning Around Medoids").

This works exactly like K-means, only that

- instead of the estimated group means $\hat{\mu}_k$ one member of each group is selected as its representative (the so-called "medoid")
- instead of squared Euclidean distance other dissimilarity measures are also allowed.

3.3.6 Application of K-means to Iris data

Scatter plots of Iris data:



The R output from a K-means analysis with known true number of clusters specified ($K = 3$) is:

```
kmeans.out = kmeans(X.iris, 3)
kmeans.out
```

```
## K-means clustering with 3 clusters of sizes 53, 50, 47
##
## Cluster means:
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
```

```
## 1  -0.05005221 -0.88042696    0.3465767   0.2805873
## 2  -1.01119138  0.85041372   -1.3006301  -1.2507035
## 3   1.13217737  0.08812645    0.9928284   1.0141287
##
## Clustering vector:
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 1 1 1 3 1 1 1 1 1 1 1 1 3 1 1 1 1 3 1 3
##  [75] 1 3 3 3 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 3 1 3 3 3 3 1 3 1 1 3
## [112] 3 3 1 1 3 3 3 3 1 3 1 3 1 3 3 1 3 3 3 3 3 3 1 1 3 3 3 1 3 3 3 1 3 3 3 3
## [149] 3 1
##
## Within cluster sum of squares by cluster:
## [1] 44.08754 47.35062 47.45019
##  (between_SS / total_SS =  76.7 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.with
## [6] "betweenss"    "size"         "iter"         "ifault"
```

The corresponding total within-group sum of squares (SSW, `tot.withinss`) is

```
kmeans.out$tot.withinss
```

```
## [1] 138.8884
```

and the between-group sum of squares (SSB, `betweenss`) is

```
kmeans.out$betweenss
```

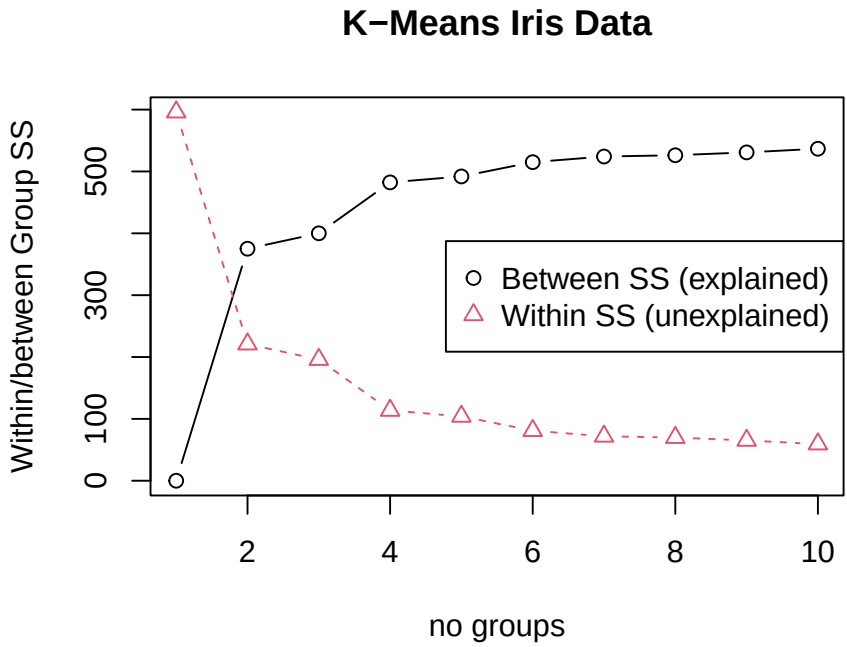
```
## [1] 457.1116
```

By comparing with the known class assignments we can assess the accuracy of K-means clustering:

```
table(L.iris, kmeans.out$cluster)
```

```
##
## L.iris         1  2  3
##   setosa       0 50  0
##   versicolor 39  0 11
##   virginica  14  0 36
```

For choosing K we run K-means several times and compute within and between cluster variation in dependence of K:



Thus, $K = 3$ clusters seem appropriate since the the explained variation does not significantly improve (and the unexplained variation does not significantly decrease) with a further increase of the number of clusters.

3.3.7 Arbitrariness of cluster labels and label switching

It is important to realise that in unsupervised learning and clustering the labels of each group are assigned in an arbitrary fashion. Recall that for K groups there are $K!$ possibilities to attach the labels, corresponding to the number of permutations of K groups.

Thus, different runs of a clustering algorithm such as K-means may return the same clustering (groupings of samples) but with different labels. This phenomenon is called "label switching" and makes it difficult to automatise comparison of clusterings. In particular, one cannot simply rely on the automatically assigned group label, instead one needs to compare the actual members of the clusters.

A way to resolve the problem of label switching is to relabel the clusters using additional information, such as requiring that some samples are in specific groups (e.g.: sample 1 is always in group labelled "1"), and/or linking labels to orderings or constraints on the group characteristics (e.g.: the group with label "1" has always a smaller mean that group with label "2").

3.4 Mixture models

3.4.1 Finite mixture model

- K groups / classes / categories, with finite K known in advance.
- Probability of class k: $\Pr(k) = \pi_k$ with $\sum_{k=1}^{K} \pi_k = 1$.
- Each class $k \in C = \{1, \ldots, K\}$ is modelled by its own distribution F_k with own parameters $\boldsymbol{\theta}_k$.
- Density of class k: $f_k(\boldsymbol{x}) = f(\boldsymbol{x}|k)$.
- The conditional means and variances for each class $k \in C$ are $\mathrm{E}(\boldsymbol{x}|k) = \boldsymbol{\mu}_k$ and $\mathrm{Var}(\boldsymbol{x}|k) = \boldsymbol{\Sigma}_k$.
- The resulting mixture density for the observed variable $\boldsymbol{x}$ is

$$f_{\text{mix}}(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k f_k(\boldsymbol{x})$$

Very often one uses **multivariate normal components** $f_k(\boldsymbol{x}) = N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
$\Longrightarrow$ **Gaussian mixture model** (GMM)

Mixture models are fundamental not just in clustering but for many other applications (e.g. classification).

Note: don't confuse *mixture model* with *mixed model* (= terminology for a *random effects* regression model).

3.4.2 Total mean and variance of mixture model

Using the law of total expectation we obtain the mean of the mixture density as follows:

$$\begin{aligned}\mathrm{E}(\boldsymbol{x}) &= \mathrm{E}(\mathrm{E}(\boldsymbol{x}|k)) \\ &= \sum_{k=1}^{K} \pi_k \boldsymbol{\mu}_k \\ &= \boldsymbol{\mu}_0\end{aligned}$$

Similarly, using the law of total variance we compute the marginal variance:

$$\underbrace{\mathrm{Var}(\boldsymbol{x})}_{\text{total}} = \underbrace{\mathrm{Var}(\mathrm{E}(\boldsymbol{x}|k))}_{\text{explained / between-group}} + \underbrace{\mathrm{E}(\mathrm{Var}(\boldsymbol{x}|k))}_{\text{unexplained / within-group}}$$

$$\boldsymbol{\Sigma}_0 = \sum_{k=1}^{K} \pi_k (\boldsymbol{\mu}_k - \boldsymbol{\mu}_0)(\boldsymbol{\mu}_k - \boldsymbol{\mu}_0)^T + \sum_{k=1}^{K} \pi_k \boldsymbol{\Sigma}_k$$

Thus, the **total variance** decomposes into the **explained (between group) variance** and the **unexplained (within group) variance**. This is the same decomposition as in linear regression (see MATH20802 Statistical Methods).

3.4.3 Total variation

The **total variation** is given by the **trace of the covariance matrix**. The above decomposition for the total variation is

$$\begin{aligned}\mathrm{Tr}(\boldsymbol{\Sigma}_0) &= \sum_{k=1}^{K} \pi_k \mathrm{Tr}((\boldsymbol{\mu}_k - \boldsymbol{\mu}_0)(\boldsymbol{\mu}_k - \boldsymbol{\mu}_0)^T) + \sum_{k=1}^{K} \pi_k \mathrm{Tr}(\boldsymbol{\Sigma}_k) \\ &= \sum_{k=1}^{K} \pi_k (\boldsymbol{\mu}_k - \boldsymbol{\mu}_0)^T (\boldsymbol{\mu}_k - \boldsymbol{\mu}_0) + \sum_{k=1}^{K} \pi_k \mathrm{Tr}(\boldsymbol{\Sigma}_k)\end{aligned}$$

If the covariances are replaced by their empirical estimates we obtain the $T = B + W$ decomposition of total variation familiar from K-means:

$$T = \mathrm{Tr}\left(\hat{\boldsymbol{\Sigma}}_0\right) = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_0)^T (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_0)$$

$$B = \frac{1}{n} \sum_{k=1}^{K} n_k (\hat{\boldsymbol{\mu}}_k - \hat{\boldsymbol{\mu}}_0)^T (\hat{\boldsymbol{\mu}}_k - \hat{\boldsymbol{\mu}}_0)$$

$$W = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in G_k} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^T (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)$$

3.4.4 Univariate mixture

For a univariate mixture ($d = 1$) with $K = 2$ components we get

$$\mu_0 = \pi_1 \mu_1 + \pi_2 \mu_2 \,,$$
$$\sigma^2_{\text{within}} = \pi_1 \sigma_1^2 + \pi_2 \sigma_2^2 = \sigma^2_{\text{pooled}} \,,$$

also know as pooled variance, and

$$\begin{aligned}\sigma^2_{\text{between}} &= \pi_1 (\mu_1 - \mu_0)^2 + \pi_2 (\mu_2 - \mu_0)^2 \\ &= \pi_1 \pi_2^2 (\mu_1 - \mu_2)^2 + \pi_2 \pi_1^2 (\mu_1 - \mu_2)^2 \\ &= \pi_1 \pi_2 (\mu_1 - \mu_2)^2 \\ &= \left(\frac{1}{\pi_1} + \frac{1}{\pi_2}\right)^{-1} (\mu_1 - \mu_2)^2\end{aligned} \,.$$

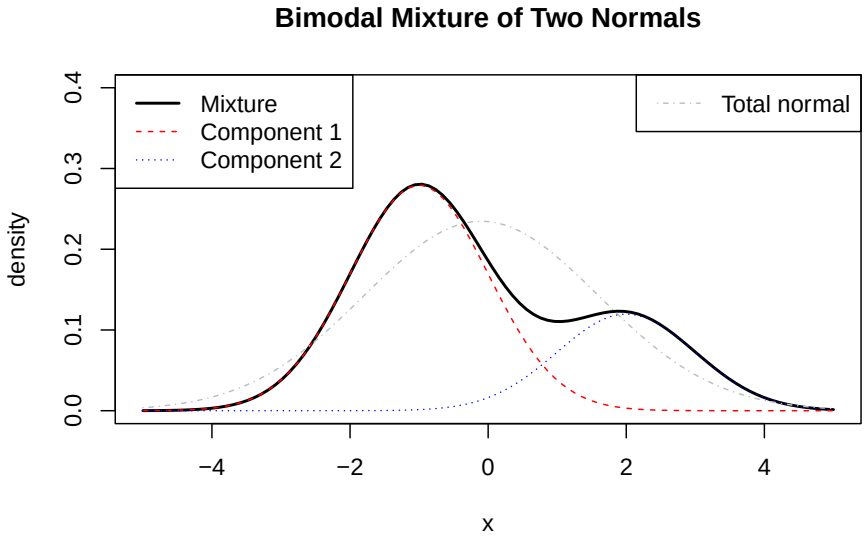
The ratio of the between-group variance and the within-group variance is proportional (by factor of n) to the squared pooled-variance t-score:

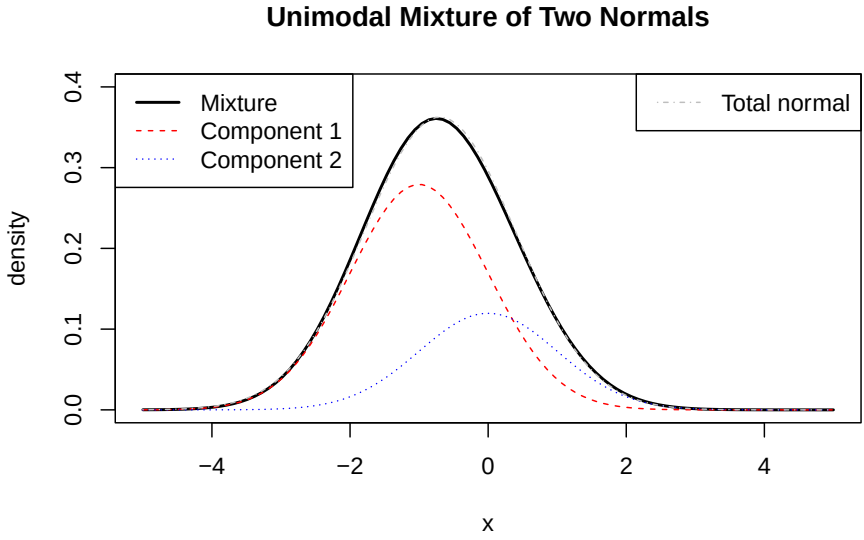
$$\frac{\sigma^2_{\text{between}}}{\sigma^2_{\text{within}}} = \frac{(\mu_1 - \mu_2)^2}{\left(\frac{1}{\pi_1} + \frac{1}{\pi_2}\right) \sigma^2_{\text{pooled}}} = \frac{t^2_{\text{pooled}}}{n}$$

If you are familiar with ANOVA (e.g. linear models course) you will recognise this ratio as the F-score.

3.4.5 Example of mixtures

Mixtures can take on many different shapes and forms, so it is instructive to study a few examples. An interactive tool to visualise two component normal mixture is available online as R Shiny web app at https://minerva.it.manchester.ac.uk/shiny/strimmer/mixture/ .

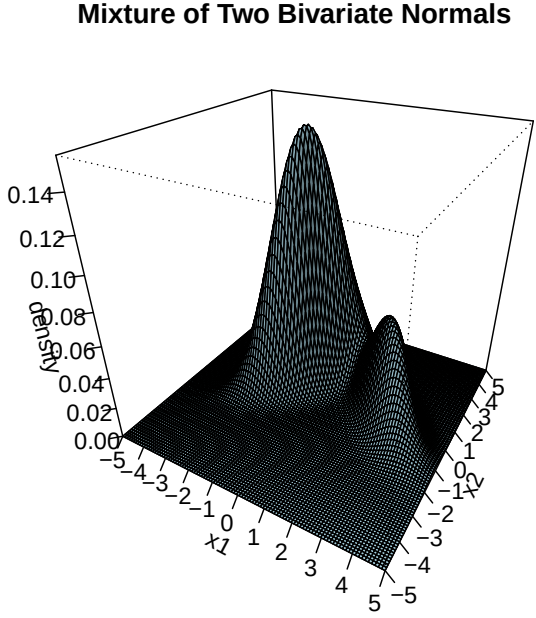




The first plot shows the bimodal density of a mixture distribution consisting of two normals with $\pi_1 = 0.7$, $\mu_1 = -1$, $\mu_2 = 2$ and the two variances equal to 1 ($\sigma_1^2 = 1$ and $\sigma_2^2 = 1$). Because the two components are well-separated there are two clear modes. The plot also shows the density of a normal distribution with the same total mean ($\mu_0 = -0.1$) and variance ($\sigma_0^2 = 2.89$) as the mixture distribution. Clearly the total normal and the mixture density are very different.

However, a two-component mixtures can also be unimodal. For example, if the mean of the second component is adjusted to $\mu_2 = 0$ then there is only a single mode and the total normal density with $\mu_0 = -0.7$ and $\sigma_0^2 = 1.21$ is now almost inistinguishable in form from the mixture density. Thus, in this case it will be very hard (or even impossible) to identify the two peaks from data.

Most mixtures we consider in this course are multivariate. For illustration, here is a plot of a mixture of two bivariate normals, with $\pi_1 = 0.7$, $\boldsymbol{\mu}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, $\boldsymbol{\Sigma}_1 = \begin{pmatrix} 1 & 0.7 \\ 0.7 & 1 \end{pmatrix}$, $\boldsymbol{\mu}_2 = \begin{pmatrix} 2.5 \\ 0.5 \end{pmatrix}$ and $\boldsymbol{\Sigma}_2 = \begin{pmatrix} 1 & -0.7 \\ -0.7 & 1 \end{pmatrix}$:



3.4.6 Sampling from a mixture model

Assuming we know how to sample from the component densities $f_k(\boldsymbol{x})$ of the mixture model it is straightforward to set up a procedure for sampling from the mixture $f_{\text{mix}}(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k f_k(\boldsymbol{x})$ itself.

This is done in a two-step process:

1. Draw from categorical distribution with parameter $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)^T$:

   $$\boldsymbol{z} \sim \text{Cat}(\boldsymbol{\pi})$$

   Here the vector $\boldsymbol{z} = (z_1, \ldots, z_K)^T$ indicates a hard group 0/1 allocation, with all components $z_{\neq k} = 0$ except for a single entry $z_k = 1$.

2. Subsequently, sample from the component k selected in step 1:

   $$\boldsymbol{x} \sim F_k$$

This two-stage sampling approach is also known as hierarchical generative model for a mixture distribution. This generative view is not only useful for simulating data from a mixture model but also highlights the role of the latent variable (the class allocation).

3.5 Fitting mixture models to data and inferring the latent states

In the following we denote by

- $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T$ the data matrix containing the observations of n independent and identically distributed samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, and
- $\boldsymbol{y} = (y_1, \ldots, y_n)^T$ the associated group memberships, as well as
- the parameters $\boldsymbol{\theta}$ which for a Gaussian mixture model are $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K\}$.

3.5.1 Observed and latent variables

When we observe data from a mixture model we collect samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. Associated with each observed $\boldsymbol{x}_i$ is the corresponding underlying class allocation $y_1, \ldots, y_n$ where each y_i takes on a value from $C = \{1, \ldots, K\}$. Crucially, the class allocations y_i are unknown and cannot be directly observed, thus are latent.

- The **joint density** for observed and unobserved variables:

  $$f(\boldsymbol{x}, y) = f(\boldsymbol{x}|y)\Pr(y) = f_y(\boldsymbol{x})\pi_y$$

The mixture density is therefore a **marginal density** as it arises from the joint density $f(\boldsymbol{x}, y)$ by marginalising over the discrete variable y.

- Marginalisation: $f(\boldsymbol{x}) = \sum_{y \in C} f(\boldsymbol{x}, y)$

3.5.2 Complete data likelihood and observed data likelihood

If we know $\boldsymbol{y}$ in advance, i.e. if we know which sample belongs to a particular group, we can construct a *complete data log-likelihood* based on the joint distribution $f(\boldsymbol{x}, y) = \pi_y f_y(\boldsymbol{x})$. The log-likelihood for $\boldsymbol{\theta}$ given the both $\boldsymbol{X}$ and $\boldsymbol{y}$ is

$$\log L(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}) = \sum_{i=1}^{n} \log f(\boldsymbol{x}_i, y_i) = \sum_{i=1}^{n} \log \left(\pi_{y_i} f_{y_i}(\boldsymbol{x}_i)\right)$$

On the other hand, typically we do not know $\boldsymbol{y}$ and therefore use the marginal or mixture density $f(\boldsymbol{x})$ to construct the *observed data log-likelihood* (sometimes also called *incomplete data log-likelihood*) $f(\boldsymbol{x}|\boldsymbol{\theta})$ as

$$\begin{aligned}\log L(\boldsymbol{\theta}|\boldsymbol{X}) &= \sum_{i=1}^{n} \log f(\boldsymbol{x}_i|\boldsymbol{\theta}) \\ &= \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k f_k(\boldsymbol{x}_i)\right)\end{aligned}$$

The *observed data log-likelihood* can also be computed from the complete data likelihood function by marginalising over $\boldsymbol{y}$

$$\begin{aligned}\log L(\boldsymbol{\theta}|\boldsymbol{X}) &= \log \sum_{\boldsymbol{y}} L(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y}) \\ &= \log \sum_{y_1,\ldots,y_K} \prod_{i=1}^{n} f(\boldsymbol{x}_i, y_i) \\ &= \log \prod_{i=1}^{n} \sum_{k=1}^{K} f(\boldsymbol{x}_i, k) \\ &= \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} f(\boldsymbol{x}_i, k)\right)\end{aligned}$$

Clustering with a mixture model can be viewed as an *incomplete* or *missing* data problem (see also part II of MATH20802 Statistical Methods).

Specifically, we face the problem of

- fitting the model using only the observed data $\boldsymbol{X}$ and
- simultaneously inferring the class allocations $\boldsymbol{y}$, i.e. states of the latent variable.

3.5.3 Fitting the mixture model to the observed data

For large sample size n the standard way to fit a mixture model is to employ maximum likelihood to find the MLEs of the parameters of the mixture model.

The direct way to fit a mixture model by maximum likelihood is to **maximise the observed data log-likelihood function** with regard to $\boldsymbol{\theta}$:

$$\hat{\boldsymbol{\theta}}^{ML} = \arg\max_{\boldsymbol{\theta}} \log L(\boldsymbol{\theta}|\boldsymbol{X})$$

Unfortunately, in practise evaluation and optimisation of the log-likelihood function can be difficult due to a number of reasons:

- The form of the observed data log-likelihood function prevents analytic simplifications (note the sum inside the logarithm) and thus can be difficult to compute.
- Because of the symmetries due to exchangeability of cluster labels the likelihood function is multimodal and thus hard to optimise. Note this is also linked to the general problem of label switching and non-identifiability of cluster labels — see the discussion for K-means clustering.
- Further identifiability issues can arise if (for instance) two neighboring components of the mixture model are largely overlapping and thus are too close to each other to be discriminated as two different modes. In other words, it is difficult to determine the number of classes.
- Furthermore, the likelihood in Gaussian mixture models is singular if one of the fitted covariance matrices becomes singular. However, this can be easily adressed by using some form of regularisation (Bayes, penalised ML, etc.) or simply by requiring sufficient sample size per group.

3.5.4 Predicting the group allocation of a given sample

In probabilistic clustering the aim is to infer the latent states $y_1, \ldots, y_n$ for all observed samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

Assuming that the mixture model is known (either in advance or after fitting it) Bayes' theorem allows predict the probability that an observation $\boldsymbol{x}_i$ falls in group $k \in \{1, \ldots, K\}$:

$$q_i(k) = \Pr(k|\boldsymbol{x}_i) = \frac{\pi_k f_k(\boldsymbol{x}_i)}{f(\boldsymbol{x}_i)}$$

Thus, for each of the n samples we get a probability mass function over the K classes with $\sum_{k=1}^{K} q_i(k) = 1$.

The posterior probabilities in $q_i(k)$ provide a so-called *soft assignment* of the sample $\boldsymbol{x}_i$ to all classes rather than a $0/1$ *hard assignment* to a specific class (as for example in the K-means algorithm).

To obtain at a hard clustering and to infer the most probable latent state we select the class with the highest probability

$$y_i = \arg\max_{k} q_i(k)$$

Thus, in probabilistic clustering we directly obtain an assessment of the uncertainty of the class assignment for a sample $\boldsymbol{x}_i$ (which is not the case in simple algorithmic clustering such K-means). We can use this information to check whether there are several classes with equal or similar probability. This will be the case, e.g., if $\boldsymbol{x}_i$ lies near the boundary between two neighbouring classes.

Using the interactive Shiny app for the univariate normal component mixture (online at https://minerva.it.manchester.ac.uk/shiny/strimmer/mixture/) you can explore the posterior probabilities of each class.

3.6 Application of Gaussian mixture models

3.6.1 Choosing the number of classes

In an application of a GMM we need to select a suitable value for K, i.e. the number of classe.

Since GMMs operate in a likelihood framework we can use penalised likelihood model selection criteria to choose among different models (i.e. GMMs with different numbers of classes).

The most popular choices are AIC (Akaike Information Criterion) and BIC (Bayesian Information criterion) defined as follows:

$$\text{AIC} = -2 \log L + 2K$$

$$\text{BIC} = -2 \log L + K \log(n)$$

In order to choose a suitable model we evaluate different models with different K and then choose the model that minimises AIC or BIC

Note that in both criteria more complex models with more parameters (in this case groups) are penalised over simpler models in order to prevent overfitting.

Another way of choosing optimal numbers of clusters is by cross-validation (see later chapter on supervised learning).

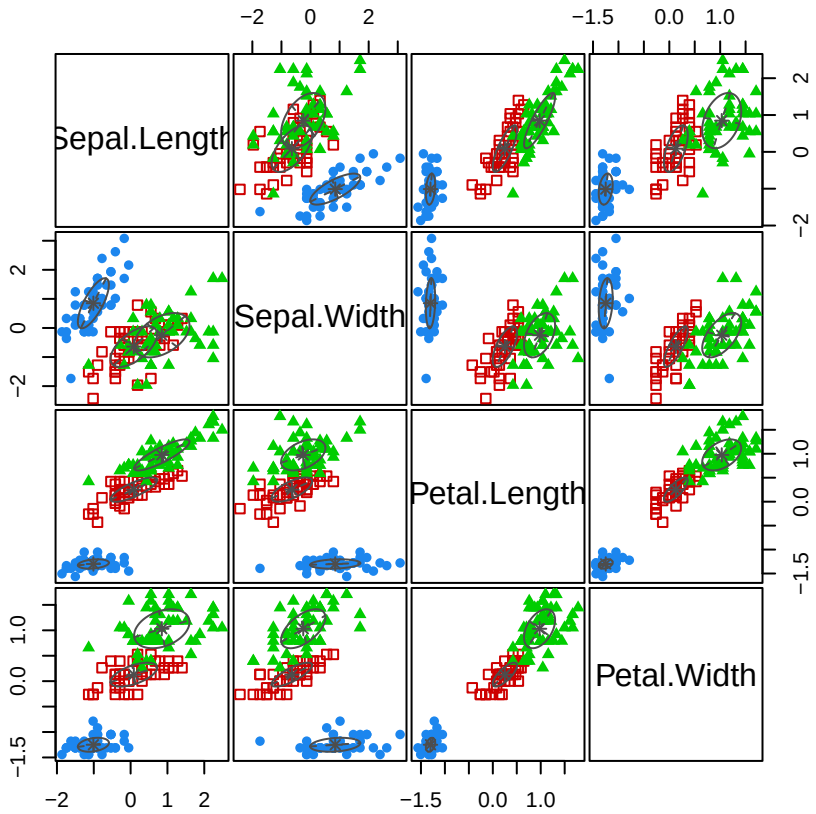
3.6.2 Application of GMMs to Iris flower data

We now explore the application of Gaussian mixture models to the Iris flower data set we also investigated with PCA and K-means.

First, we fit a GMM with 3 clusters, using the R software "mclust."[5]

[5]L. Scrucca L. et. al. 2016. mclust 5: Clustering, classification and density estimation using Gaussian finite mixture models. The R Journal 8:205–233. See https://journal.r-project.org/archive/2016/RJ-2016-021/ and https://mclust-org.github.io/mclust/

```
data(iris)
X.iris = scale((iris[, 1:4]), scale=TRUE) # center and standardise
L.iris = iris[, 5]

library("mclust")
gmm3 = Mclust(X.iris, G=3, verbose=FALSE)
plot(gmm3, what="classification")
```



The "mclust" software has used the following model when fitting the mixture:

```
gmm3$modelName
```

```
## [1] "VVV"
```

Here "VVV" is the name used by the "mclust" software for a model allowing for an individual unrestricted covariance matrix $\mathbf{\Sigma}_k$ for each class k.

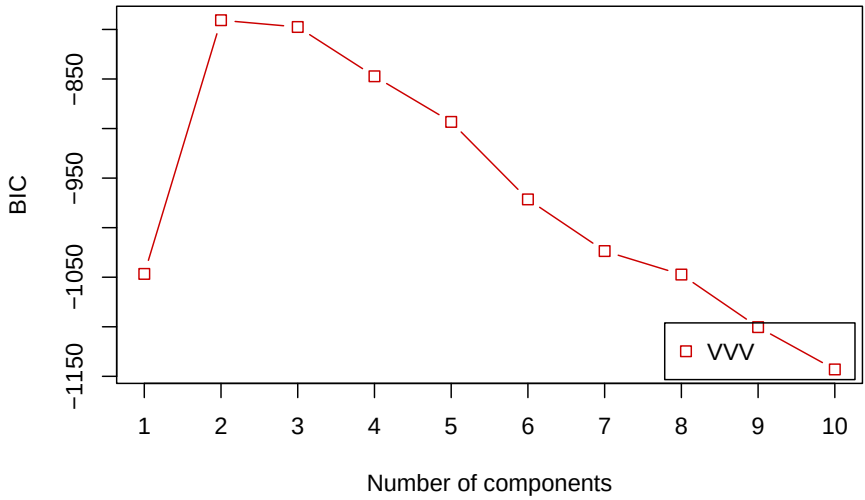
This GMM has a substantially lower misclassification error compared to K-means with the same number of clusters:

```
table(gmm3$classification, L.iris)
```

```
##    L.iris
##     setosa versicolor virginica
##  1     50          0         0
##  2      0         45         0
##  3      0          5        50
```

Note that in "mclust" the BIC criterion is defined with the opposite sign ($\text{BIC}_{\text{mclust}} = 2 \log L - K \log(n)$), thus we need to find the *maximum* value rather than the smallest value.

If we compute BIC for various numbers of groups we find that the model with the best $\text{BIC}_{\text{mclust}}$ is a model with 2 clusters but the model with 3 cluster has nearly as good a BIC:



3.7 The EM algorithm

3.7.1 Motivation

As discussed above, the observed data log-likelihood can be difficult to maximise directly due to its form as a log marginal likelihood. Intriguingly, it is possible to optimise it indirectly using the complete data log-likelihood!

This method is called the EM algorithm[6] and has been formaly proposed by Arthur Dempster and others in 1977. It iteratively estimates both the parameters of the mixture model parameters and the latent states The key idea behind the EM algorithm is to exploits the simplicity of the complete data likelihood and to obtain estimates of $\boldsymbol{\theta}$ by imputing the missing group allocations and then subsequently iteratively refining both the imputations and the estimates of $\boldsymbol{\theta}$.

More precisely, in the EM (=expectation-maximisation) algorithm we alternate between

1) updating the soft allocations of each sample using the current estimate of the parameters $\boldsymbol{\theta}$ (obtained in step 2)
2) updating the parameter estimates by maximising the *expected* complete data log-likelihood. The expectation is taken with regard to the distribution over the latent states (obtained in step 1). Thus the complete data log-likelihood is averaged over the soft class assignments.

3.7.2 The EM algorithm

Specifically, the EM algorithm proceeds as follows:

1) Initialisation:

- Start with a guess of the parameters $\hat{\boldsymbol{\theta}}^{(1)}$, then continue with "E" Step, Part A.
- Alternatively, start with a guess of the soft allocations for each sample $q_i(k)^{(1)}$, collected in the matrix $\boldsymbol{Q}^{(1)}$, then continue with "E" Step, Part B. This may be derived from some prior information, e.g., from running K-means. Caveat: some particular initialisations correspond to invariant states and hence should be avoided (see further below).

2) **E "expectation" step**

- Part A: Use Bayes' theorem to compute new probabilities of allocation to class k for all the samples $\boldsymbol{x}_i$:

$$q_i(k)^{(b+1)} \leftarrow \frac{\hat{\pi}_k^{(b)} f_k(\boldsymbol{x}_i|\hat{\boldsymbol{\theta}}^{(b)})}{f(\boldsymbol{x}_i|\hat{\boldsymbol{\theta}}^{(b)})}$$

  Note that to obtain $q_i(k)^{(b+1)}$ the current estimate $\hat{\boldsymbol{\theta}}^{(b)}$ of the parameters of the mixture model is required.

- Part B: Construct the *expected* complete data log-likelihood function for $\boldsymbol{\theta}$

[6] Dempster, A. P, N. M. Laird and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. JRSS B **39**:1–38. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x

using the soft allocations $q_i(k)^{(b+1)}$ collected in the matrix $\boldsymbol{Q}^{(b+1)}$:

$$G(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{Q}^{(b+1)}) = \sum_{i=1}^{n} \sum_{k=1}^{K} q_i(k)^{(b+1)} \log\left(\pi_k f_k(\boldsymbol{x}_i)\right)$$

Note that in the case that the soft allocations $\boldsymbol{Q}^{(b+1)}$ turn into hard $0/1$ allocations then $G(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{Q}^{(b+1)})$ becomes equivalent to the complete data log-likelihood.

3) **M "maximisation" step** — Maximise the expected complete data log-likelihood to update the estimates of mixture model parameters:

$$\hat{\boldsymbol{\theta}}^{(b+1)} \leftarrow \arg\max_{\boldsymbol{\theta}} G(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{Q}^{(b+1)})$$

4) Continue with 2) "E" Step until the series $\hat{\boldsymbol{\theta}}^{(1)}, \hat{\boldsymbol{\theta}}^{(2)}, \hat{\boldsymbol{\theta}}^{(3)}, \ldots$ has converged.

Since maximisation of the expected complete data log-likelihood is typically much easier (and often also analytically tractable) the EM algorithm is often preferred over direct maximisation of the observed data log-likelihood.

Note that to avoid singularities in the expected log-likelihood function we may need to adopt regularisation (i.e. penalised maximum likelihood or Bayesian learning) for estimating the parameters in the M-step.

3.7.3 EM algorithm for multivariate normal mixture model

For a Gaussian mixture model (GMM) both steps in the EM algorithm can be expressed analytically:

E-step:

Update the soft allocations:

$$q_i(k)^{(b+1)} = \frac{\hat{\pi}_k^{(b)} N(\boldsymbol{x}_i|\hat{\boldsymbol{\mu}}_k^{(b)}, \hat{\boldsymbol{\Sigma}}_k^{(b)})}{\hat{f}^{(b)}\left(\boldsymbol{x}_i|\hat{\pi}_1^{(b)}, \ldots, \hat{\pi}_K^{(b)}, \hat{\boldsymbol{\mu}}_1^{(b)}, \ldots, \hat{\boldsymbol{\mu}}_K^{(b)}, \hat{\boldsymbol{\Sigma}}_1^{(b)}, \ldots, \hat{\boldsymbol{\Sigma}}_K^{(b)}\right)}$$

M-step:

The number of samples assigned to class k in the current step is

$$n_k^{(b+1)} = \sum_{i=1}^{n} q_i(k)^{(b+1)}$$

Note this is not necessarily an integer because of the soft allocations of samples to groups!

The updated estimate of the group probabilities is

$$\hat{\pi}_k^{(b+1)} = \frac{n_k^{(b+1)}}{n}$$

The updated estimate of the mean is

$$\hat{\boldsymbol{\mu}}_k^{(b+1)} = \frac{1}{n_k^{(b+1)}} \sum_{i=1}^{n} q_i(k)^{(b+1)} \boldsymbol{x}_i$$

and the updated covariance estimate is

$$\hat{\boldsymbol{\Sigma}}_k^{(b+1)} = \frac{1}{n_k^{(b+1)}} \sum_{i=1}^{n} q_i(k)^{(b+1)} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_k^{(b+1)}\right) \left(\boldsymbol{x}_i - \boldsymbol{\mu}_k^{(b+1)}\right)^T$$

Note that if $q_i(k)$ is a hard allocation (so that for any i only one class has weight 1 and all others weight 0) then all estimators above reduce to the usual empirical estimators.

In Worksheet 8 you can find a simple R implementation of the EM algorithm for univariate normal mixtures.

Similar analytical expressions as in the normal case can also be found in more general mixtures where the components are exponential families.

3.7.4 Connection with K-means clustering method

The K-means algorithm is very closely related to the EM algorithm and probabilistic clustering with a specific Gaussian mixture models.

Specifically, we assume a simplified model where the probabilities π_k of all classes are equal (i.e. $\pi_k = \frac{1}{K}$) and where the covariances $\boldsymbol{\Sigma}_k$ are all of the same spherical form $\sigma^2 \boldsymbol{I}$. Thus, the covariance does not depend on the group, there is no correlation between the variables and the variance of all variables is the same.

First, we consider the "E" step. Using the mixture model above the soft assignment for the class allocation becomes

$$\log(q_i(k)) = -\frac{1}{2\sigma^2}(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^T(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k) + \text{const}$$

where const does not depend on k. This can turned into a hard class allocation by

$$\begin{aligned} y_i &= \arg\max_k \log(q_i(k)) \\ &= \arg\min_k (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^T(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k) \end{aligned}$$

which is exactly the K-means rule to allocate of samples to groups.

Second, in the "M" step we compute the parameters of the model. If the class allocations are hard the expected log-likelihood becomes the observed data likelihood and the MLE of the group mean is the average of samples in that group.

Thus, K-means can be viewed as an EM type algorithm to provide hard classification based on a simple restricted Gaussian mixture model.

3.7.5 Why the EM algorithm works — an entropy point of view

The iterative (soft) imputation of the latent states in the EM algorithm is intuitive. However, it is not immediately clear why the expected observed log-likelihood needs to be maximised rather than, e.g., the observed log-likelihood with hard allocations. Furthermore, we need to show that maximising the marginal likelihood and applying the EM algorithm with Bayesian imputation of the latent states both lead to the same fitted mixture model.

Intriguingly, the EM algorithm is easiest to understand from an entropy point of view, considering the entropy foundations of maximum likelihood and Bayesian learning — for details see part I and II of MATH20802 Statistical Methods.

First, recall that the method of maximum likelihood results from minimising the KL divergence between an empirical distribution Q_x representing the observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and the model family $F_x^{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$:

$$\hat{\boldsymbol{\theta}}^{ML} = \arg\min_{\boldsymbol{\theta}} D_{\mathrm{KL}}(Q_x, F_x^{\boldsymbol{\theta}})$$

The KL divergence decomposes into a cross-entropy and an entropy part

$$D_{\mathrm{KL}}(Q_x, F_{\boldsymbol{\theta}}) = H(Q_x, F_x^{\boldsymbol{\theta}}) - H(Q_x)$$

hence minimising the KL divergence with regard to $\boldsymbol{\theta}$ is the same as maximising the function

$$\begin{aligned} -nH(Q_x, F_x^{\boldsymbol{\theta}}) &= n\mathrm{E}_{Q_x}(\log f(\boldsymbol{x}|\boldsymbol{\theta})) \\ &= \sum_{i=1}^{n} \log f(\boldsymbol{x}_i|\boldsymbol{\theta}) \\ &= \log L(\boldsymbol{\theta}|\boldsymbol{X}) \end{aligned}$$

which is indeed the observed data log-likelihood for $\boldsymbol{\theta}$.

Second, we recall the chain rule for the KL divergence. Specifically, the KL divergence for the joint model forms an upper bound of the KL divergence for

the marginal model:

$$D_{\mathrm{KL}}(Q_{x,y}, F^{\boldsymbol{\theta}}_{x,y}) = D_{\mathrm{KL}}(Q_x, F^{\boldsymbol{\theta}}_x) + \underbrace{D_{\mathrm{KL}}(Q_{y|x}, F^{\boldsymbol{\theta}}_{y|x})}_{\geq 0}$$
$$\geq D_{\mathrm{KL}}(Q_x, F^{\boldsymbol{\theta}}_x)$$

Unlike for $\boldsymbol{x}$ we do not have observations about y. Nonetheless, we can model the joint distribution $Q_{x,y} = Q_x Q_{y|x}$ by assuming a distribution $Q_{y|x}$ over the latent variable.

The EM algorithm arises from iteratively decreasing the joint KL divergence $D_{\mathrm{KL}}(Q_x Q_{y|x}, F^{\boldsymbol{\theta}}_{x,y})$ with regard to both $Q_{y|x}$ and $\boldsymbol{\theta}$:

1) "E" Step: While keeping $\boldsymbol{\theta}$ fixed we vary $Q_{y|x}$ to minimise the joint KL divergence. The minimum is reached at $D_{\mathrm{KL}}(Q_{y|x}, F^{\boldsymbol{\theta}}_{y|x}) = 0$. This is the case for $Q_{y|x} = F^{\boldsymbol{\theta}}_{y|x}$, i.e. when the latent distribution $Q_{y|x}$ representing the soft allocations is computed by conditioning, i.e. using Bayes' theorem.

2) "M" Step: While keeping $Q_{y|x}$ fixed the joint KL divergence is further minimised with regard to $\boldsymbol{\theta}$. This is equivalent to maximising the function $\sum_{k=1}^{K} \sum_{i=1}^{n} q(k|\boldsymbol{x}_i) \log f(\boldsymbol{x}_i, k|\boldsymbol{\theta})$ which is indeed the expected complete data log-likelihood.

Note that in both steps the joint KL divergence always decreases and never increases. Furthermore, at the end of "E" step the joint KL divergence equals the marginal KL divergence. Thus, this procedure implicitly minimises the marginal KL divergence as well, and hence maximises the marginal log-likelihood.

Alternatively, using $H(Q_{x,y}) = H(Q_x) + H(Q_{y|x})$ we can rewrite the above upper bound for the joint KL divergence as an equivalent lower bound for n times the negative marginal cross-entropy:

$$-nH(Q_x, F^{\boldsymbol{\theta}}_x) = \underbrace{-nH(Q_x Q_{y|x}, F^{\boldsymbol{\theta}}_{x,y}) + nH(Q_{y|x})}_{\text{lower bound, ELBO}} + \underbrace{nD_{\mathrm{KL}}(Q_{y|x}, F^{\boldsymbol{\theta}}_{y|x})}_{\geq 0}$$
$$\geq \mathcal{F}\left(Q_x, Q_{y|x}, F^{\boldsymbol{\theta}}_{x,y}\right)$$

The lower bound is known as the "ELBO". The EM algorithm then arises by maximising $\mathcal{F}$ with regard to $Q_{y|x}$ ("E" step") and $\boldsymbol{\theta}$ ("M" step).

The entropy interpretation of the EM algorithm is due to Csiszàr and Tusnàdy (1984)[7] and the ELBO interpretation was introduced by Neal and Hinton (1998).[8]

[7] Csiszàr, I., and G, Tusnàdy. 1984. Information geometry and alternating minimization procedures. In Dudewicz, E. J. et al. (eds.) Recent Results in Estimation Theory and Related Topics Statistics and Decisions, Supplement Issue No. 1.

[8] Neal, R. M., and G. Hinton. 1998. A view of the EM algorithm that justifies incremental, sparse, and other variants. In Jordan, M.I. (eds.). Learning in Graphical Models. pp. 355–368. https://doi.org/10.1007/978-94-011-5014-9_12

3.7.6 Convergence and invariant states

Under mild assumptions the EM algorithm is guaranteed to monotonically converge to local optima of the observed data log-likelihood.[9] Thus the series $\hat{\boldsymbol{\theta}}^{(1)}, \hat{\boldsymbol{\theta}}^{(2)}, \hat{\boldsymbol{\theta}}^{(3)}, \ldots$ converges to the estimate $\hat{\boldsymbol{\theta}}$ found when maximising the observed data log-likelihood. However, the speed of convergence in the EM algorithm can sometimes be slow, and there are also situations in which there is no convergence at all to $\hat{\boldsymbol{\theta}}$ because the EM algorithm remains in an invariant state.

An example of such an invariant state for a Gaussian mixture model is uniform initialisation of the latent variables $q_i(k) = \frac{1}{K}$, where K is the number of classes. With this we get in the M step $n_k = \frac{n}{K}$ and as parameter estimates

$$\hat{\pi}_k = \frac{1}{K}$$

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i = \bar{\boldsymbol{x}}$$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{1}{n}\sum_{i=1}^{n} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T = \hat{\boldsymbol{\Sigma}}$$

Crucially, none of these actually depend on the group k! Thus, in the E step when the next soft allocations are determined this leads to

$$q_i(k) = \frac{\frac{1}{K} N(\boldsymbol{x}_i | \bar{\boldsymbol{x}}, \hat{\boldsymbol{\Sigma}})}{\sum_{j=1}^{K} \frac{1}{K} N(\boldsymbol{x}_i | \bar{\boldsymbol{x}}, \hat{\boldsymbol{\Sigma}})} = \frac{1}{K}$$

After one cycle in the EM algorithm we arrive at the same soft allocation that we started with, and the algorithm is trapped in an invariant state! Therefore uniform initialisation should clearly be avoided!

You can explore this effect in practise in Worksheet 8.

[9] Wu, C.F. 1983. On the convergence properties of the EM algorithm. The Annals of Statistics *11*:95–103. https://doi.org/10.1214/aos/1176346060

Chapter 4

Supervised learning and classification

4.1 Aims of supervised learning

4.1.1 Supervised learning vs. unsupervised learning

Unsupervised learning:

Starting point:

- unlabelled data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

Aim: find labels $y_1, \ldots, y_n$ to attach to each sample $\boldsymbol{x}_i$.

For discrete labels y unsupervised learning is called *clustering*.

Supervised learning:

Starting point:

- labelled *training data*: $\{\boldsymbol{x}_1^{\text{train}}, y_1^{\text{train}}\}, \ldots, \{\boldsymbol{x}_n^{\text{train}}, y_n^{\text{train}}\}$
- In addition, we have unlabelled *test data*: $\boldsymbol{x}^{\text{test}}$

Aim: use training data to learn a function, say $h(\boldsymbol{x})$, to predict the label corresponding to the test data. The predictor function may provide a soft (probabilistic) assignment or a hard assignment of a class label to a test sample.

For y discrete supervised learning is called *classification*. For continuous y the label is called response and supervised learning becomes *regression*.

Thus, supervised learning is a two-step procedure:

1) Learn predictor function $h(\boldsymbol{x})$ using the training data $\boldsymbol{x}_i^{\text{train}}$ plus labels y_i^{train}.

2) Predict the label y^{test} for the test data $\boldsymbol{x}^{\text{test}}$ using the estimated classifier function: $\hat{y}^{\text{test}} = \hat{h}(\boldsymbol{x}^{\text{test}})$.

4.1.2 Terminology

The function $h(\boldsymbol{x})$ that predicts the class y is called a *classifier*.

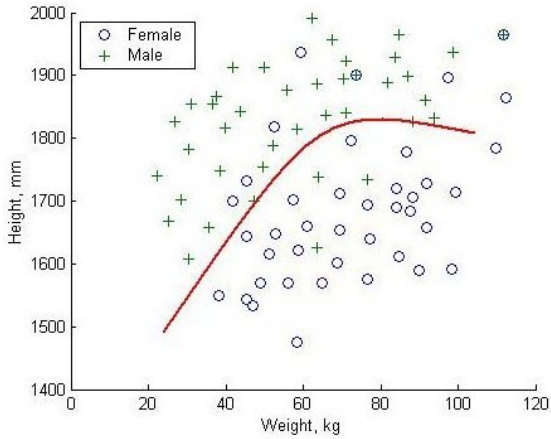
There are many types of classifiers, we focus here primarily on probabilistic classifiers (i.e. those that output probabilities for each possible class/label).

The challenge is to find a classifier that

- explains the current training data well *and*
- that also generalises well to future unseen data.

Note that it is relatively easy to find a predictor that explains the training data but especially in high dimensions (i.e. with many predictors) there is often overfitting and then the predictor does not generalise well!

The *decision boundary* between the classes is defined as the set of all $\boldsymbol{x}$ for which the class assignment by the predictor $h(\boldsymbol{x})$ switches from one class to another.



In general, simple decision boundaries are preferred over complex decision boundaries to avoid overfitting.

Some commonly used probabilistic methods for classifications:

- QDA (quadratic discriminant analysis)
- LDA (linear discriminant analysis)
- DDA (diagonal discriminant analysis),
- Naive Bayes classification
- logistic regression

Common non-probabilistic methods include:

- SVM (support vector machine),
- random forest
- neural networks

Depending on how the classifiers are trainined there are many variations of the above methods, e.g. Fisher discriminant analysis, regularised LDA, shrinkage disciminant analysis etc.

4.2 Bayesian discriminant rule or Bayes classifier

Same setup as with mixture models:

- K groups with K prespecified
- each group has its own distribution F_k with own parameters $\boldsymbol{\theta}_k$
- the density of each class is $f_k(\boldsymbol{x}) = f(\boldsymbol{x}|k)$.
- prior probability of group k is $\Pr(k) = \pi_k$ with $\sum_{k=1}^K \pi_k = 1$
- marginal density is the mixture $f(\boldsymbol{x}) = \sum_{k=1}^K \pi_k f_k(\boldsymbol{x})$

The posterior probability of group k is then

$$\Pr(k|\boldsymbol{x}) = \frac{\pi_k f_k(\boldsymbol{x})}{f(\boldsymbol{x})}$$

This already provides a "soft" classification

$$\boldsymbol{h}(\boldsymbol{x}^{\text{test}}) = (\Pr(k=1|\boldsymbol{x}^{\text{test}}), \ldots, \Pr(k=K|\boldsymbol{x}^{\text{test}}))^T$$

where each possible class $k \in \{1, \ldots, K\}$ is assigned a probability to be the label for the test sample $\boldsymbol{x}$.

The *discriminant function* is the logarithm of the posterior probability:

$$d_k(\boldsymbol{x}) = \log \Pr(k|\boldsymbol{x}) = \log \pi_k + \log f_k(\boldsymbol{x}) - \log f(\boldsymbol{x})$$

Since we use d_k to compare the different classes k we can simplify the discriminant function by dropping all constant terms that do not depend on k — in the above this is the term $\log f(\boldsymbol{x})$. Hence we get for the Bayes discriminant function

$$d_k(\boldsymbol{x}) = \log \pi_k + \log f_k(\boldsymbol{x})\,.$$

For subsequent "hard" classification $h(\boldsymbol{x}^{\text{test}})$ we then select the group/label for which the value of the discriminant function is maximised:

$$\hat{y}^{\text{test}} = h(\boldsymbol{x}^{\text{test}}) = \arg\max_k d_k(\boldsymbol{x}^{\text{test}})\,.$$

The discriminant functions $d_k(\boldsymbol{x})$ can be mapped back to the probabilistic class assignment by using the softargmax function (also known as softmax function):

$$\Pr(k|\boldsymbol{x}) = \frac{\exp(d_k(\boldsymbol{x}))}{\sum_{c=1}^K \exp(d_c(\boldsymbol{x}))} = \frac{\exp(d_k(\boldsymbol{x}) - d_{\max})}{\sum_{c=1}^K \exp(d_c(\boldsymbol{x}) - d_{\max})}\,.$$

Note that subtracting $d_{\max} = \max\{d_1(\boldsymbol{x}), \ldots, d_K(\boldsymbol{x})\}$ avoids numerical overflow problems when computing the exponential by standardising the maximum of the discriminant functions to zero.

We have already encountered the Bayes classifier in the EM algorithm to predict the state of the latent variables (soft assignment) and in the K-means algorithm (hard assignment).

4.3 Normal Bayes classifier

4.3.1 Quadratic discriminant analysis (QDA) and Gaussian assumption

Quadratic discriminant analysis (QDA) is a special case of the Bayes classifier when all densities are multivariate normal with $f_k(\boldsymbol{x}) = N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

This leads to the discriminant function for QDA:

$$d_k^{QDA}(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k) - \frac{1}{2}\log\det(\boldsymbol{\Sigma}_k) + \log(\pi_k)$$

There are a number of noteworthy things here:

- Again terms are dropped that do not depend on k, such as $-\frac{d}{2}\log(2\pi)$.
- Note the appearance of the squared **Mahalanobis distance** between $\boldsymbol{x}$ and $\boldsymbol{\mu}_k$ — $d^{Mahalanobis}(\boldsymbol{x}, \boldsymbol{\mu}|\boldsymbol{\Sigma})^2 = (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})$.
- The **QDA discriminant function is quadratic in $\boldsymbol{x}$** - hence its name! This implies that the **decision boundaries for QDA classification are quadratic** (i.e. parabolas in two dimensional settings).

For Gaussian models specifically it can useful be to multiply the discriminant function by -2 to get rid of the factor $-\frac{1}{2}$, but note that in that case we then need to find the minimum of the discriminant function rather than the maximum:

$$d_k^{QDA(v2)}(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k) + \log\det(\boldsymbol{\Sigma}_k) - 2\log(\pi_k)$$

In the literature you will find both versions of Gaussian discriminant functions so you need to check carefully which convention is used. In the following we will use the first version only.

Decision boundaries for the QDA classifier can be either linear or nonlinear (quadratic curve). The decision boundary between any two classes i and j require that $d_i^{QDA}(\boldsymbol{x}) = d_j^{QDA}(\boldsymbol{x})$, or equivalently $< d_i^{QDA}(\boldsymbol{x}) - d_j^{QDA}(\boldsymbol{x}) = 0$, which is a quadratic equation.

4.3.2 Linear discriminant analysis (LDA)

LDA is a special case of QDA, with the assumption of common overall covariance across all groups: $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$.

This leads to a simplified discriminant function:

$$d_k^{LDA}(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k) + \log(\pi_k)$$

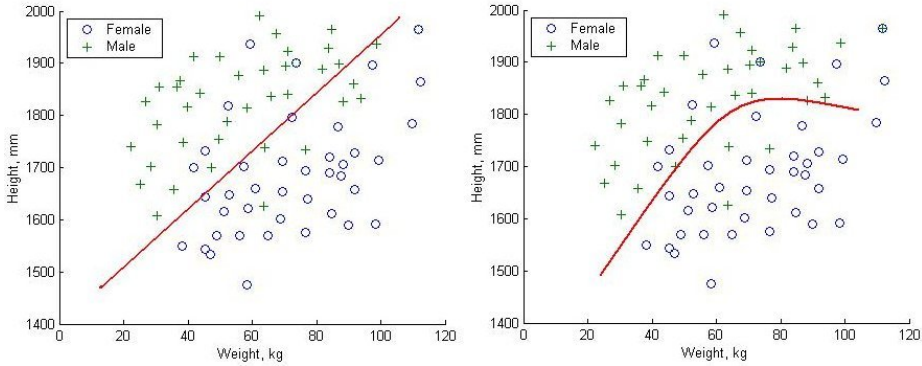
Note that term containing the log-determinant is now gone, and that LDA is essentially now a method that tries to minimize the Mahalanobis distance (while taking also into account the prior class probabilities).

The above function can be further simplified, by noting that the quadratic term $\boldsymbol{x}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{x}$ does not depend on k and hence can be dropped:

$$\begin{aligned} d_k^{LDA}(\boldsymbol{x}) &= \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1} \boldsymbol{x} - \frac{1}{2} \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \log(\pi_k) \\ &= \boldsymbol{b}^T \boldsymbol{x} + a \end{aligned}$$

Thus, the **LDA discriminant function is linear in $\boldsymbol{x}$, and hence the resulting decision boundaries are linear** as well (i.e. straight lines in two-dimensional settings).

Comparison of linear decision boundaries of LDA (left) compared with QDA (right):



Note that logistic regression (cf. GLM module) takes on exactly the above linear form and is indeed closely linked with the LDA classifier.

4.3.3 Diagonal discriminant analysis (DDA)

For DDA we start with the same setting as for LDA, but now we simplify the model even further by additionally requiring a **diagonal covariance** containing only the variances (thus we assume that all correlations among the predictors $x_1, \ldots, x_d$ are zero):

$$\boldsymbol{\Sigma} = \boldsymbol{V} = \begin{pmatrix} \sigma_1^2 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma_d^2 \end{pmatrix}$$

This simplifies the inversion of $\boldsymbol{\Sigma}$ as

$$\boldsymbol{\Sigma}^{-1} = \boldsymbol{V}^{-1} = \begin{pmatrix} \sigma_1^{-2} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_d^{-2} \end{pmatrix}$$

and leads to the discriminant function

$$\begin{aligned} d_k^{DDA}(\boldsymbol{x}) &= \boldsymbol{\mu}_k^T \boldsymbol{V}^{-1} \boldsymbol{x} - \frac{1}{2} \boldsymbol{\mu}_k^T \boldsymbol{V}^{-1} \boldsymbol{\mu}_k + \log(\pi_k) \\ &= \sum_{j=i}^{d} \frac{\mu_{k,j} x_j - \mu_{k,j}^2 / 2}{\sigma_d^2} + \log(\pi_k) \end{aligned}$$

As special case of LDA, the **DDA classifier is a linear classifier** and thus has linear decision boundaries.

The **Bayes classifier** (using any distribution) **assuming uncorrelated predictors** is also known as the **naive Bayes classifier**.

Hence, **DDA is a naive Bayes classifier** assuming underlying Gaussian distributions.

However, don't let you misguide because of the name "naive": in fact DDA and other "naive" Bayes classifier are often very effective classifiers, especially in high-dimensional settings!

4.4 The training step — learning QDA, LDA and DDA classifiers from data

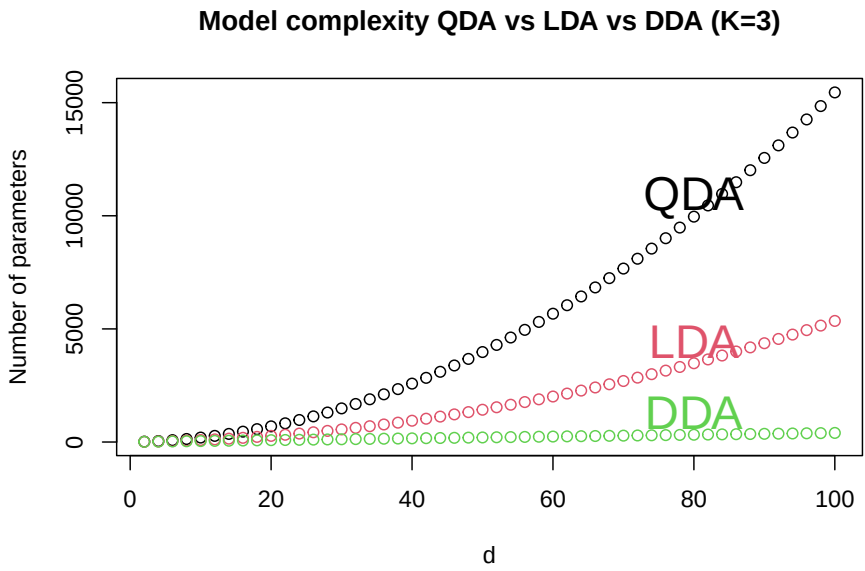
4.4.1 Number of model parameters

In order to predict the class for new data using any of the above discriminant functions we need to first learn the underlying parameters from the training data $\boldsymbol{x}_i^{\text{train}}$ and y_i^{train}:

- For QDA, LDA and DDA we need to learn $\pi_1, \dots, \pi_K$ with $\sum_{k=1}^K \pi_k = 1$ and the mean vectors $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$
- For QDA we additionally require $\boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_K$
- For LDA we need $\boldsymbol{\Sigma}$
- For DDA we estimate $\sigma_1^2, \dots, \sigma_d^2$.

Overall, the total number of parameters to be estimated when learning the discriminant functions from training data is as follows:

- QDA: $K - 1 + Kd + K\frac{d(d+1)}{2}$
- LDA: $K - 1 + Kd + \frac{d(d+1)}{2}$
- DDA: $K - 1 + Kd + d$



4.4.2 Estimating the discriminant / predictor function

For QDA, LDA and DDA we learn the predictor by estimating the parameters of the discriminant function from the training data.

4.4.2.1 Large sample size

If the sample size of the training data set is sufficiently large compared to the model dimensions we can use maximum likelihood (ML) to estimate the model parameters. To be able use ML we need a larger sample size for QDA and LDA (because full covariances need to be estimated) but for DDA relatively small sample size can be sufficient (which explains why "naive" Bayes methods are very popular in practise).

To obtain the parameters estimates we use the known labels y_i^{train} to sort the samples $\boldsymbol{x}_i^{\text{train}}$ into the corresponding classes, and then apply the standard ML estimators. Let $g_k = \{i : y_i^{\text{train}} = k\}$ be the set of all indices of training sample belonging to group k, n_k the sample size in group k

The ML estimates of the class probabilities are the frequencies

$$\hat{\pi}_k = \frac{n_k}{n}$$

and the ML estimate of the group means $k = 1, \ldots, K$ are

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i \in g_k} \boldsymbol{x}_i^{\text{train}} .$$

The ML estimate of the global mean $\boldsymbol{\mu}_0$ (i.e. if we assume there is only a single class and ignore the group labels) is

$$\hat{\boldsymbol{\mu}}_0 = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i^{\text{train}} = \sum_{k=1}^{K} \hat{\pi}_k \hat{\boldsymbol{\mu}}_k$$

Note the global mean is identical to the pooled mean (i.e. weighted average of the individual group means).

The ML estimates for the covariances $\boldsymbol{\Sigma}_k$ for QDA are

$$\widehat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k} \sum_{i \in g_k} (\boldsymbol{x}_i^{\text{train}} - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i^{\text{train}} - \hat{\boldsymbol{\mu}}_k)^T$$

In order to get the ML estimate of the pooled variance $\boldsymbol{\Sigma}$ for use with LDA we compute

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in g_k} (\boldsymbol{x}_i^{\text{train}} - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i^{\text{train}} - \hat{\boldsymbol{\mu}}_k)^T = \sum_{k=1}^{K} \hat{\pi}_k \widehat{\boldsymbol{\Sigma}}_k$$

Note that the pooled variance $\boldsymbol{\Sigma}$ differs (substantially!) from the global variance Σ_0 that results from simply ignoring class labels and that is computed as

$$\widehat{\boldsymbol{\Sigma}}_0^{ML} = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{x}_i^{\text{train}} - \hat{\boldsymbol{\mu}}_0)(\boldsymbol{x}_i^{\text{train}} - \hat{\boldsymbol{\mu}}_0)^T$$

You will recognise the above from the variance decomposition in mixture models, with $\boldsymbol{\Sigma}_0$ being the total variance and the pooled $\boldsymbol{\Sigma}$ the unexplained/with-in group variance.

4.4.2.2 Small sample size

If the dimension d is large compared to the sample size then the number of parameters in the predictor function grows fast. Especially QDA but also LDA is data hungry and ML estimation becomes an ill-posed problem.

As discussed in Section 1.3 in this instance we need to use a regularised estimator for the covariance(s) such as estimators derived in the framework of penalised ML, Bayesian learning, shrinkage estimation etc. This also ensures that the estimated covariance matrices are positive definite (which is automatically guaranteed only for DDA if all variances are positive).

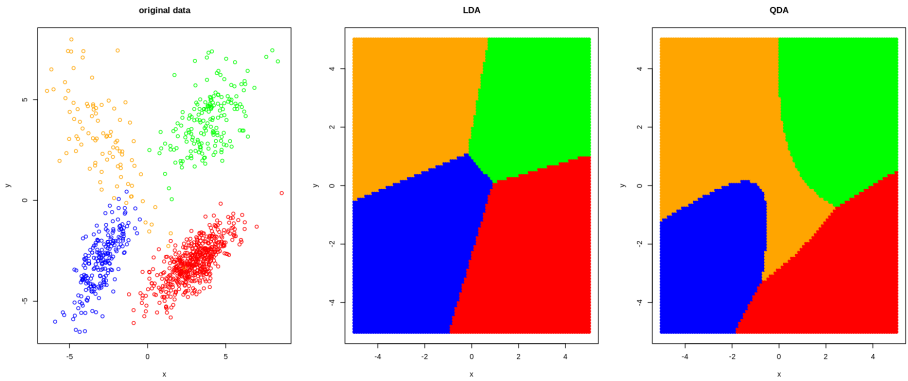
Furthermore, in small sample setting it is advised to reduce the number of parameters of the model. Thus using LDA or DDA is preferred over QDA. This can also prevent overfitting and lead to a predictor that generalises better.

To analyse high-dimensional data in the worksheets we will employ a regularised version of LDA and DDA using Stein-type shrinkage estimation as discussed in Section 1.3 and implemented in the R package "sda".

4.4.3 Comparison of estimated decision boundaries: LDA vs. QDA

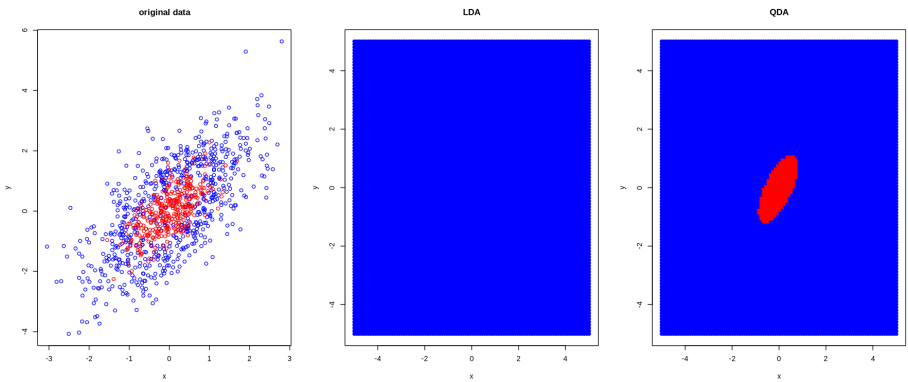
We compare two simple scenarios using simulated data.

Non-nested case ($K = 4$):



Both LDA and QDA clearly separate the 4 classes. Note the curved decision boundaries for QDA and the linear decision boundaries for LDA.

Nested case ($K = 2$):



In the nested case LDA fails to separate the two classes because there is no way to separate two nested classes with a simple linear boundary.

4.5 Quantifying prediction error

Once a classifier has been trained we are naturally interested in its performance to correctly classify previously unseen data points. This is useful for comparing

different types of classifiers and also for comparing the same type of classifier using different sets of predictor variables.

4.5.1 Quantification of prediction error based on validation data

A measure of predictor error compares the predicted label $\hat{y}$ with the true label y for validation data. A validation data set contains both the $\boldsymbol{x}_i$ and the associated label y_i but unlike the training data it has not been used for learning the predictor function.

For continuous response often the squared loss is used:

$$\text{err}(\hat{y}, y) = (\hat{y} - y)^2$$

For binary outcomes one often employs the $0/1$ loss:

$$\text{err}(\hat{y}, y) = \begin{cases} 0, & \text{if } \hat{y} = y \\ 1, & \text{otherwise} \end{cases}$$

Alternatively, any other quantity derived from the confusion matrix (containing TP, TN, FP, FN) can be used.

The mean prediction error is the expectation

$$PE = \text{E}(\text{err}(\hat{y}, y))$$

and thus the empirical mean prediction error is

$$\widehat{PE} = \frac{1}{m} \sum_{i=1}^{m} \text{err}(\hat{y}_i, y_i)$$

where m is the sample size of the validation data set.

More generally, we can also quantify prediction error in the framework of so-called **proper scoring rules**, where the whole probabilistic forecast is taken into account (e.g. the individual probabilities for each class, rather than just the selected most probable class). A commonly used scoring rule is the negative log-probability ("surprise"), and the expected surprise is the cross-entropy (cf. Statistical Methods module). So this leads back to entropy and likelihood (see MATH20802 Statistical Methods).

Once we have an estimate of the prediction error of a model we can use it to compare and choose among a set of candidate models, selecting those with a sufficiently low prediction error.

4.5.2 Estimation of prediction error using cross-validation

Unfortunately, quite often we do not have separate validation data available to evaluate a classifier.

In this case we need to rely on a simple algorithmic procedure called **cross-validation**.

Outline of cross-validation:

1) split the samples in the training data into a number (say K) parts ("folds").
2) use each of the K folds as validation data and the other $K - 1$ folds as training data.
3) average over the resulting K individual estimates of prediction error, to get an overall aggregated predictor error, along with an error.

Note that in each case one part of the data is reserved for validation and *not* used for training the predictor.

We choose K such that the folds are not too small (to allow estimation of prediction error) but also not too large (to make sure that we actually are able to train a reliable classifier from the remaining data). A typical value for K is 5 or 10, so that 80% respectively 90% of the samples are used for training and the other 20 % or 10% for validation.

If $K = n$ there are as many folds as there are samples and the validation data set consists only of a single data point. This is called "leave one out" cross-validation (LOOCV). There are analytic approximations for the prediction error obtained by LOOCV so that this approach is computationally inexpensive for some standard models (including regression).

Further reading:

To study the technical details of cross-validation: read **Section 5.1 Cross-Validation** in James et al. (2021) *An introduction to statistical learning with applications in R (2nd edition)*. Springer.

4.6 Goodness of fit and variable ranking

As in linear regression (cf. "Statistical Methods" module) we are interested in finding out whether the fitted mixture model is an appropriate model, and which particular predictor(s) x_j from $\boldsymbol{x} = (x_1, \ldots, x_d)^T$ are responsible prediction the outcome, i.e. for categorizing a sample into group k.

In order to study these problem it is helpful to rewrite the discriminant function to highlight the influence (or importance) of each predictor.

We focus on linear methods (LDA and DDA) and first look at the simple case $K = 2$ and then generalise to more than two groups.

4.6.1 LDA with $K = 2$ classes

For two classes using the LDA discriminant rule will choose group $k = 1$ if $d_1^{LDA}(\boldsymbol{x}) > d_2^{LDA}(\boldsymbol{x})$, or equivalently, if

$$\Delta_{12}^{LDA} = d_1^{LDA}(\boldsymbol{x}) - d_2^{LDA}(\boldsymbol{x}) > 0$$

Since $d_k(\boldsymbol{x})$ is the log-posterior (plus/minus identical constants) Δ^{LDA} is in fact the **log-posterior odds of class 1 versus class 2** (see Statistical Methods, Bayesian inference).

The difference Δ_{12}^{LDA} is

$$\underbrace{\Delta_{12}^{LDA}}_{\text{log posterior odds}} = \underbrace{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} \left(\boldsymbol{x} - \frac{\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2}{2} \right)}_{\text{log Bayes factor } \log B_{12}} + \underbrace{\log \left(\frac{\pi_1}{\pi_2} \right)}_{\text{log prior odds}}$$

Note that since we only consider simple non-composite models here the log-Bayes factor is identical with the log-likelihood ratio!

The log Bayes factor $\log B_{12}$ is known as the *weight of evidence* in favour of F_1 given $\boldsymbol{x}$. The *expected weight of evidence* assuming $\boldsymbol{x}$ is indeed from F_1 is the Kullback-Leibler discrimination information in favour of group 1, i.e. the KL divergence of from distribution F_2 to F_1:

$$\mathrm{E}_{F_1}(\log B_{12}) = D_{\mathrm{KL}}(F_1, F_2) = \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = \frac{1}{2}\Omega^2$$

This yields, apart of a scale factor, a population version of the Hotelling T^2 statistic defined as

$$T^2 = c^2 (\hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_2)^T \hat{\boldsymbol{\Sigma}}^{-1} (\hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_2)$$

where $c = (\frac{1}{n_1} + \frac{1}{n_2})^{-1/2} = \sqrt{n \pi_1 \pi_2}$ is a sample size dependent factor (for $\mathrm{SD}(\hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_2)$). T^2 is a measure of fit of the underlying two-component mixture.

Using the whitening transformation with $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$ and $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{\Sigma}^{-1}$ we can rewrite the log Bayes factor as

$$\begin{aligned} \log B_{12} &= \left((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \boldsymbol{W}^T \right) \left(\boldsymbol{W} \left(\boldsymbol{x} - \frac{\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2}{2} \right) \right) \\ &= \boldsymbol{\omega}^T \boldsymbol{\delta}(\boldsymbol{x}) \end{aligned}$$

i.e. as the product of two vectors:

- $\boldsymbol{\delta}(\boldsymbol{x})$ is the whitened $\boldsymbol{x}$ (centered around average means) and
- $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_d)^T = \boldsymbol{W}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ gives the weight of each whitened component $\boldsymbol{\delta}(\boldsymbol{x})$ in the log Bayes factor.

A large positive or negative value of ω_j indicates that the corresponding whitened predictor is relevant for choosing a class, whereas small values of ω_j close to zero indicate that the corresponding ZCA whitened predictor is unimportant. Furthermore, $\boldsymbol{\omega}^T\boldsymbol{\omega} = \sum_{j=1}^{d} \omega_j^2 = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = \Omega^2$, i.e. the squared ω_j^2 provide a component-wise decomposition of the overall fit Ω^2.

Choosing ZCA-cor as whitening transformation with $\boldsymbol{W} = \boldsymbol{P}^{-1/2}\boldsymbol{V}^{-1/2}$ we get

$$\boldsymbol{\omega}^{ZCA-cor} = \boldsymbol{P}^{-1/2}\boldsymbol{V}^{-1/2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

A better understanding of $\boldsymbol{\omega}^{ZCA-cor}$ is provided by comparing with the two-sample t-statistic

$$\hat{\boldsymbol{\tau}} = c\hat{\boldsymbol{V}}^{-1/2}(\hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_2)$$

With $\boldsymbol{\tau}$ the population version of $\hat{\boldsymbol{\tau}}$ we can define

$$\boldsymbol{\tau}^{adj} = \boldsymbol{P}^{-1/2}\boldsymbol{\tau} = c\,\boldsymbol{\omega}^{ZCA-cor}$$

as correlation-adjusted t-scores (cat scores). With $(\hat{\boldsymbol{\tau}}^{adj})^T\hat{\boldsymbol{\tau}}^{adj} = T^2$ we can see that the cat scores offer a component-wise decomposition of Hotelling's T^2.

Note the choice of ZCA-cor whitening is to ensure that the whitened components are interpretable and stay maximally correlated to the original variables. However, you may also choose for example PCA whitening in which case the $\boldsymbol{\omega}^T\boldsymbol{\omega}$ provide the variable importance for the PCA whitened variables.

For DDA, which assumes that correlations among predictors vanish, i.e. $\boldsymbol{P} = \boldsymbol{I}_d$, we get

$$\Delta_{12}^{DDA} = \underbrace{\left((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\boldsymbol{V}^{-1/2}\right)}_{c^{-1}\boldsymbol{\tau}^T} \underbrace{\left(\boldsymbol{V}^{-1/2}\left(\boldsymbol{x} - \frac{\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2}{2}\right)\right)}_{\text{centered standardised predictor}} + \log\left(\frac{\pi_1}{\pi_2}\right)$$

Similarly as above, the t-score $\boldsymbol{\tau}$ determines the impact of the standardised predictor in Δ^{DDA}.

Consequently, in DDA we can rank predictors by the squared t-score. Recall that in standard linear regression with uncorrelated predictors we can find the most important predictors by ranking the squared marginal correlations – ranking by (squared) t-scores in DDA is the exact analogy but for discrete response.

4.6.2 Multiple classes

For more than two classes we need to refer to the so-called **pooled centroids formulation** of DDA and LDA (introduced by Tibshirani 2002).

The pooled centroid is given by $\boldsymbol{\mu}_0 = \sum_{k=1}^{K} \pi_k\boldsymbol{\mu}_k$ — this is the centroid if there would be only a single class. The corresponding probability (for a single class) is $\pi_0 = 1$ and the distribution is called F_0.

The LDA discriminant function for this "group 0" is

$$d_0^{LDA}(\boldsymbol{x}) = \boldsymbol{\mu}_0^T \boldsymbol{\Sigma}^{-1} \boldsymbol{x} - \frac{1}{2} \boldsymbol{\mu}_0^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_0$$

and the log posterior odds for comparison of group k with the pooled group 0 is

$$\begin{aligned} \Delta_k^{LDA} &= d_k^{LDA}(\boldsymbol{x}) - d_0^{LDA}(\boldsymbol{x}) \\ &= \log B_{k0} + \log(\pi_k) \\ &= \boldsymbol{\omega}_k^T \boldsymbol{\delta}_k(\boldsymbol{x}) + \log(\pi_k) \end{aligned}$$

with

$$\boldsymbol{\omega}_k = \boldsymbol{W}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_0)$$

and

$$\boldsymbol{\delta}_k(\boldsymbol{x}) = \boldsymbol{W}(\boldsymbol{x} - \frac{\boldsymbol{\mu}_k + \boldsymbol{\mu}_0}{2})$$

The expected log Bayes factor is

$$\mathrm{E}_{F_k}(\log B_{k0}) = KL(F_k || F_0) = \frac{1}{2}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_0)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_k - \boldsymbol{\mu}_0) = \frac{1}{2} \Omega_k^2$$

With scale factor $c_k = (\frac{1}{n_k} - \frac{1}{n})^{-1/2} = \sqrt{n \frac{\pi_k}{1-\pi_k}}$ (for SD($\hat{\boldsymbol{\mu}}_k - \hat{\boldsymbol{\mu}}_0$), with the minus sign before $\frac{1}{n}$ due to correlation between $\hat{\boldsymbol{\mu}}_k$ and pooled mean $\hat{\boldsymbol{\mu}}_0$) we get as correlation-adjusted t-score for comparing mean of group k with the pooled mean

$$\boldsymbol{\tau}_k^{adj} = c_k \boldsymbol{\omega}_k^{ZCA-cor} .$$

For the two class case ($K = 2$) we get with $\boldsymbol{\mu}_0 = \pi_1 \boldsymbol{\mu}_1 + \pi_2 \boldsymbol{\mu}_2$ for the mean difference $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) = \pi_2(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ and with $c_1 = \sqrt{n \frac{\pi_1}{\pi_2}}$ this yields

$$\boldsymbol{\tau}_1^{adj} = \sqrt{n \pi_1 \pi_2} \boldsymbol{P}^{-1/2} \boldsymbol{V}^{-1/2} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2),$$

i.e. the exact same score as in the two-class setting.

4.7 Variable selection

In the previous we saw that in DDA the natural score for **ranking features** with regard to their relevance in separating the classes is the (squared) t-score, and for LDA a whitened version such as the squared correlation-adjusted t-score (based on ZCA-cor whitening) may be used. Once such a ranking has been established the question of a **suitable cutoff** arises, i.e. how many features need (or should) be retained in a model.

For large and high-dimensional models **feature selection can also be viewed as a form of regularisation and also dimension reduction**. Specifically, there

may be many variables/ features that do no contribute to the class prediction. Despite having in principle no effect on the outcome the presence of these "null variables" can nonetheless deterioriate (sometimes dramatically!) the overall predictive accuracy of a trained predictor, because they add noise and increase the model dimension. Therefore, variables that do not contribute to prediction should be filtered out in order to be able to construct good prediction models and classifiers.

4.7.1 Choosing a threshold by multiple testing using false discovery rates

The most simple way to determine a cutoff threshold is to use a standard technique for multiple testing.

For each predictor variable $x_1, \ldots, x_d$ we have a corresponding test statistic measuring the influence of this variable on the response, for example the the t-scores and related statistics discussed in the previous section. In addition to providing an overall ranking the set of all these statistics can be used to determine a suitable cutoff by trying to separate two populations of predictor variables:

- "Null" variables that do not contribute to prediction
- "Alternative" variables that are linked to prediction

As discussed in the "Statistical Methods" module last term (Part 2 - Section 8) this can be done as follows:

- The distribution of the observed test statistics z_i is assumed to follow a two-component mixture where $F_0(z)$ and $F_A(z)$ are the distributions corresponding to the null and the alternative, $f_0(z)$ and $f_a(z)$ the densities, and π_0 and $\pi_A = 1 - \pi_0$ are the weights:

$$f(z) = \pi_0 f_0(z) + (1 - \pi_0) f_a(z)$$

- The null model is typically from a parametric family (e.g. normal around zero and with a free variance parameter) whereas the alternative is often modelled nonparametrically.
- After fitting the mixture model, often assuming some additional constraints to make the mixture identifiable, one can compute false discovery rates (FDR) as follows:

  Local FDR:

$$\widehat{\text{fdr}}(z_i) = \widehat{\Pr}(\text{null}|z_i) = \frac{\hat{\pi}_0 \hat{f}_0(z_i)}{\hat{f}(z_i)}$$

  Tail-area-based FDR (=q-value):

$$\widehat{\text{Fdr}}(z_i) = \widehat{\Pr}(\text{null}|Z > z_i) = \frac{\hat{\pi}_0 \hat{F}_0(z_i)}{\hat{F}(z_i)}$$

Note these are essentially p-values adjusted for multiple testing (by a variant of the Benjamini-Hochberg method).

By thresholding false discovery rates it is possible to identify those variables that clearly belong to each of the two groups but also those features that cannot easily be discriminated to fall into either group:

- "alternative" variables have low local FDR, e.g, $\widehat{\text{fdr}}(z_i) \leq 0.2$
- "null" variables have high local FDR, e.g. $\widehat{\text{fdr}}(z_i) \geq 0.8$
- features that cannot easily classified as null or alternative, e.g. $0.2 < \widehat{\text{fdr}}(z_i) < 0.8$

For feature selection in prediction settings we generally aim to remove only those variable that clearly belong to the null group, leaving all others in the model.

4.7.2 Variable selection using cross-validation

A conceptually simple but computationally more expensive approach to variable selection is to estimate the predicion error of the same type of predictor but with different sets of predictors using cross-validation, and then choosing a predictor that achieves good prediction accuracy while using only a small number of featurs.

This is a method that works very well in practise as is demonstrated in a number of problems in the worksheets.

Chapter 5

Multivariate dependencies

5.1 Measuring the linear association between two sets of random variables

5.1.1 Aim

The linear association between two scalar random variables x and y is measured by the correlation $\mathrm{Cor}(x, y) = \rho$.

In this chapter we now would like to explore how to generalise correlation to the case of two random vectors. Specifically, we would like to measure the total linear association between two random vectors (or equivalently two sets of random variables) $\boldsymbol{x} = (x_1, \ldots, x_p)^T$ and $\boldsymbol{y} = (y_1, \ldots, y_q)^T$ in a scalar quantity.

We assume a joint correlation matrix

$$\boldsymbol{P} = \begin{pmatrix} \boldsymbol{P}_x & \boldsymbol{P}_{xy} \\ \boldsymbol{P}_{yx} & \boldsymbol{P}_y \end{pmatrix}$$

with cross-correlation matrix $\boldsymbol{P}_{xy} = \boldsymbol{P}_{yx}^T$ and the within-group group correlations $\boldsymbol{P}_x$ and $\boldsymbol{P}_y$. If the cross-correlations vanish, $\boldsymbol{P}_{xy} = 0$, then the two random vectors are uncorrelated, and the joint correlation matrix becomes a diagonal block matrix

$$\boldsymbol{P}_{\text{indep}} = \begin{pmatrix} \boldsymbol{P}_x & 0 \\ 0 & \boldsymbol{P}_y \end{pmatrix}.$$

To characterise the total association between $\boldsymbol{x}$ and $\boldsymbol{y}$ we are looking for a **scalar quantity** measuring the divergence of a distribution assuming the general joint correlation matrix $\boldsymbol{P}$ from a distribution assuming the joint correlation matrix $\boldsymbol{P}_{\text{indep}}$ for uncorrelated $\boldsymbol{x}$ and $\boldsymbol{y}$.

5.1.2 Known special cases

Ideally, in case of an univariate y this measure should reduce to the *squared multiple correlation* or *coefficient of determination*

$$\mathrm{MCor}(\boldsymbol{x}, y)^2 = \boldsymbol{P}_{y\boldsymbol{x}}\boldsymbol{P}_{\boldsymbol{x}}^{-1}\boldsymbol{P}_{\boldsymbol{x}y}$$

This is well-known in linear regression to describe the strength of total linear association between the predictors $\boldsymbol{x}$ and the response y.

To derive the squared multiple correlation we may proceed as follows. First we whiten the random vector $\boldsymbol{x}$ resulting in $\boldsymbol{z}_{\boldsymbol{x}} = \boldsymbol{W}_{\boldsymbol{x}}\boldsymbol{x} = \boldsymbol{Q}_{\boldsymbol{x}}\boldsymbol{P}_{\boldsymbol{x}}^{-1/2}\boldsymbol{V}_{\boldsymbol{x}}^{-1/2}\boldsymbol{x}$ where $\boldsymbol{Q}_{\boldsymbol{x}}$ is an orthogonal matrix. The correlations between each component in $\boldsymbol{z}_{\boldsymbol{x}}$ and the response y are then

$$\boldsymbol{\omega} = \mathrm{Cor}(\boldsymbol{z}_{\boldsymbol{x}}, y) = \boldsymbol{Q}_{\boldsymbol{x}}\boldsymbol{P}_{\boldsymbol{x}}^{-1/2}\boldsymbol{P}_{\boldsymbol{x}y}$$

As $\mathrm{Var}(\boldsymbol{z}_{\boldsymbol{x}}) = \boldsymbol{I}$ and thus the components in $\boldsymbol{z}_{\boldsymbol{x}}$ are uncorrelated we can simply add up the squared individual correlations to get as total association measure

$$\boldsymbol{\omega}^T\boldsymbol{\omega} = \sum_i \omega_i^2 = \boldsymbol{P}_{y\boldsymbol{x}}\boldsymbol{P}_{\boldsymbol{x}}^{-1}\boldsymbol{P}_{\boldsymbol{x}y}$$

Note that the particular choice of the orthogonal matrix $\boldsymbol{Q}_{\boldsymbol{x}}$ for whitening $\boldsymbol{x}$ is not relevant for the squared multiple correlation.

Note that if the marginal correlations vanish ($\boldsymbol{P}_{\boldsymbol{x}y} = 0$) then $\mathrm{MCor}(\boldsymbol{x}, y)^2 = 0$. If the correlation between the predictors vanishes ($\boldsymbol{P}_{\boldsymbol{x}} = \boldsymbol{I}$) then $\mathrm{MCor}(\boldsymbol{x}, y)^2 = \sum_i \rho_{yx_i}^2$, i.e. it is the sum of the squared marginal correlations.

If there is only a single predictor x then $\boldsymbol{P}_{xy} = \rho$ and $\boldsymbol{P}_x = 1$ and the squared multiple correlation reduces to the squared Pearson correlation

$$\mathrm{Cor}(x, y)^2 = \rho^2 .$$

5.2 Canonical Correlation Analysis (CCA) aka CCA whitening

Canonical correlation analysis was invented by Harald Hotelling in 1936.[1] CCA aims to characterise the linear dependence between to random vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ by a set of **canonical correlations** λ_i.

CCA works by simultaneously whitening the two random vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ where the whitening matrices are chosen in such a way that the cross-correlation matrix between the resulting whitened variables becomes diagonal, and the elements on the diagonal correspond to the **canonical correlations**.

[1] Hotelling, H. 1936. Relations between two sets of variates. Biometrika **28**:321–377. https://doi.org/10.1093/biomet/28.3-4.321

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} \qquad \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix} \qquad \begin{aligned} \text{Var}(\boldsymbol{x}) &= \boldsymbol{\Sigma_x} = \boldsymbol{V_x^{1/2} P_x V_x^{1/2}} \\ \text{Var}(\boldsymbol{y}) &= \boldsymbol{\Sigma_y} = \boldsymbol{V_y^{1/2} P_y V_y^{1/2}} \end{aligned}$$

Dimension p Dimension q

$$\begin{aligned} \text{Whitening of } \boldsymbol{x}\text{:} \quad & \boldsymbol{z_x = W_x x = Q_x P_x^{-1/2} V_x^{-1/2} x} \\ \text{Whitening of } \boldsymbol{y}\text{:} \quad & \boldsymbol{z_y = W_y y = Q_y P_y^{-1/2} V_y^{-1/2} y} \end{aligned}$$

(note we use the correlation-based form of $\boldsymbol{W}$)

Cross-correlation between $\boldsymbol{z_y}$ and $\boldsymbol{z_y}$:

$$\text{Cor}(\boldsymbol{z_x}, \boldsymbol{z_y}) = \boldsymbol{Q_x K Q_y^T}$$

with $\boldsymbol{K = P_x^{-1/2} P_{xy} P_y^{-1/2}}$.

Idea: we can choose suitable orthogonal matrices $\boldsymbol{Q_x}$ and $\boldsymbol{Q_y}$ by putting a structural constraint on the cross-correlation matrix.

CCA: we aim for a *diagonal* $\text{Cor}(\boldsymbol{z_x}, \boldsymbol{z_y})$ so that each component in $\boldsymbol{z_x}$ only influences one (the corresponding) component in $\boldsymbol{z_y}$.

Motivation: pairs of "modules" represented by components of $\boldsymbol{z_x}$ and $\boldsymbol{z_y}$ influencing each other (and not anyone other module).

$$\boldsymbol{z_x} = \begin{pmatrix} z_1^x \\ z_2^x \\ \vdots \\ z_p^x \end{pmatrix} \quad \boldsymbol{z_y} = \begin{pmatrix} z_1^y \\ z_2^y \\ \vdots \\ z_q^y \end{pmatrix}$$

$$\text{Cor}(\boldsymbol{z_x}, \boldsymbol{z_y}) = \begin{pmatrix} \lambda_1 & \dots & 0 \\ \vdots & \vdots & \\ 0 & \dots & \lambda_m \end{pmatrix}$$

where λ_i are the *canonical correlations* and $m = \min(p, q)$.

5.2.1 How to make cross-correlation matrix $\text{Cor}(\boldsymbol{z_x}, \boldsymbol{z_y})$ diagonal?

- Use Singular Value Decomposition (SVD) of matrix $\boldsymbol{K}$:

$$\boldsymbol{K} = (\boldsymbol{Q_x^{\text{CCA}}})^T \boldsymbol{\Lambda} \boldsymbol{Q_y^{\text{CCA}}}$$

where $\mathbf{\Lambda}$ is the diagonal matrix containing the singular values of $\boldsymbol{K}$

- This yields orthogonal matrices $\boldsymbol{Q}_x^{\text{CCA}}$ and $\boldsymbol{Q}_y^{\text{CCA}}$ and thus the desired whitening matrices $\boldsymbol{W}_x^{\text{CCA}}$ and $\boldsymbol{W}_y^{\text{CCA}}$
- As a result $\text{Cor}(\boldsymbol{z}_x, \boldsymbol{z}_y) = \mathbf{\Lambda}$ i.e. singular values λ_i of $\boldsymbol{K}$ are the desired canonical correlations!

$\longrightarrow \boldsymbol{Q}_x^{\text{CCA}}$ and $\boldsymbol{Q}_y^{\text{CCA}}$ are determined by the diagonality constraint (and note these are different to the other previously discussed whitening methods).

Note that the signs of corresponding in columns in $\boldsymbol{Q}_x^{\text{CCA}}$ and $\boldsymbol{Q}_y^{\text{CCA}}$ are not identified. Traditionally, in an SVD the signs are chosen such that the singular values are positive. However, if we impose positive-diagonality on $\boldsymbol{Q}_x^{\text{CCA}}$ and $\boldsymbol{Q}_y^{\text{CCA}}$, and thus positive-diagonality on the cross-correlations $\mathbf{\Psi}_x$ and $\mathbf{\Psi}_y$, then the canonical correlations may take on both positive and negative values.

5.2.2 Related methods

- O2PLS: similar to CCA but using orthogonal projections rather than whitening.
- Vector correlation: aggregates the squared canonical correlations into a single overall measure (see below).

5.3 Vector correlation and RV coefficient

5.3.1 Vector alienation coefficient

In his 1936 paper introducing canonical correlation analysis Hotelling also proposed the *vector alienation coefficient* defined as

$$\begin{aligned} a(\boldsymbol{x}, \boldsymbol{y}) &= \frac{\det(\boldsymbol{P})}{\det(\boldsymbol{P}_{\text{indep}})} \\ &= \frac{\det(\boldsymbol{P})}{\det(\boldsymbol{P}_x)\,\det(\boldsymbol{P}_y)} \end{aligned}$$

With $\boldsymbol{K} = \boldsymbol{P}_x^{-1/2} \boldsymbol{P}_{xy} \boldsymbol{P}_y^{-1/2}$ the vector alienation coefficient can be written (using the Weinstein-Aronszajn determinant identity and the formula for the determinant of block-structured matrices, see Appendix) as

$$\begin{aligned} a(\boldsymbol{x}, \boldsymbol{y}) &= \det\left(\boldsymbol{I}_p - \boldsymbol{K}\boldsymbol{K}^T\right) \\ &= \det\left(\boldsymbol{I}_q - \boldsymbol{K}^T\boldsymbol{K}\right) \\ &= \prod_{i=1}^{m} (1 - \lambda_i^2) \end{aligned}$$

where the λ_i are the singular values of $\boldsymbol{K}$, i.e. the canonical correlations for the pair $\boldsymbol{x}$ and $\boldsymbol{y}$. Therefore, the vector alienation coefficient is computed as a summary statistic of the canonical correlations.

If $\boldsymbol{P}_{xy} = 0$ und thus $\boldsymbol{x}$ and $\boldsymbol{y}$ are uncorrelated then $\boldsymbol{P} = \boldsymbol{P}_{\text{indep}}$ and thus by construction the vector alienation coefficient $a(\boldsymbol{x}, \boldsymbol{y}) = 1$. Hence, the vector alienation coefficient is itself not a generalisation of the squared multiple correlation to the case of two random vectors as such a quantity should vanish in this case.

5.3.2 Rozeboom vector correlation

Instead, Rozeboom (1965)[2] proposed to use as squared *vector correlation* the complement of the vector alienation coefficient

$$\begin{aligned}\text{VCor}(\boldsymbol{x}, \boldsymbol{y})^2 = \rho^2_{xy} &= 1 - a(\boldsymbol{x}, \boldsymbol{y}) \\ &= 1 - \det\left(\boldsymbol{I}_p - \boldsymbol{K}\boldsymbol{K}^T\right) \\ &= 1 - \det\left(\boldsymbol{I}_q - \boldsymbol{K}^T\boldsymbol{K}\right) \\ &= 1 - \prod_{i=1}^{m}(1 - \lambda_i^2)\end{aligned}$$

If $\boldsymbol{P}_{xy} = 0$ then $\boldsymbol{K} = \boldsymbol{0}$ and hence $\text{VCor}(\boldsymbol{x}, \boldsymbol{y})^2 = 0$.

Moreover, if either $p = 1$ or $q = 1$ the squared vector correlation reduces to the corresponding squared multiple correlation, which in turn for both $p = 1$ and $q = 1$ becomes the squared Pearson correlation.

You can find the derivation in Example Sheet 10.

Thus, Rozeboom's vector correlation indeed generalises both Pearson correlation and the multiple correlation coefficient.

5.3.3 RV coefficient

Another common approach to measure association between two random vectors is the RV coefficient introduced by Robert and Escoufier in 1976 as

$$RV(\boldsymbol{x}, \boldsymbol{y}) = \frac{\text{Tr}(\boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yx})}{\sqrt{\text{Tr}(\boldsymbol{\Sigma}_x^2)\text{Tr}(\boldsymbol{\Sigma}_y^2)}}$$

The main advantage of the RV coefficient is that it is easier to compute than the Rozeboom vector correlation as it uses the matrix trace rather than the matrix determinant.

[2] Rozeboom, W. W. 1965. Linear correlations between sets of variables. Psychometrika **30**:57–71. https://doi.org/10.1007/BF02289747

For $q = p = 1$ the RV coefficient reduces to the squared correlation. However, the RV coefficient does *not* reduce to the multiple correlation coefficient for $q = 1$ and $p > 1$, and therefore the RV coefficient cannot be considered a coherent generalisation of Pearson and multiple correlation to the case when $\boldsymbol{x}$ and $\boldsymbol{y}$ are random vectors.

See also Worksheet 10.

5.4 Limits of linear models and correlation

5.4.1 Correlation measures only linear dependence

Linear models and measures of linear association (correlation) are very effective tools. However, it is important to recognise their limits especially when modelling complex nonlinear relationships.

A very simple demonstration of this is given by the following example. Assume x is a normally distributed random variable with $x \sim N(0, 1)$. From x we construct a second random variable $y = x^2$ — thus y fully depends on x with no added extra noise. What is the correlation between x and y?

Let's answer this question by running a small computer simulation:

```
x=rnorm(10000)
y = x^2
cor(x,y)
```

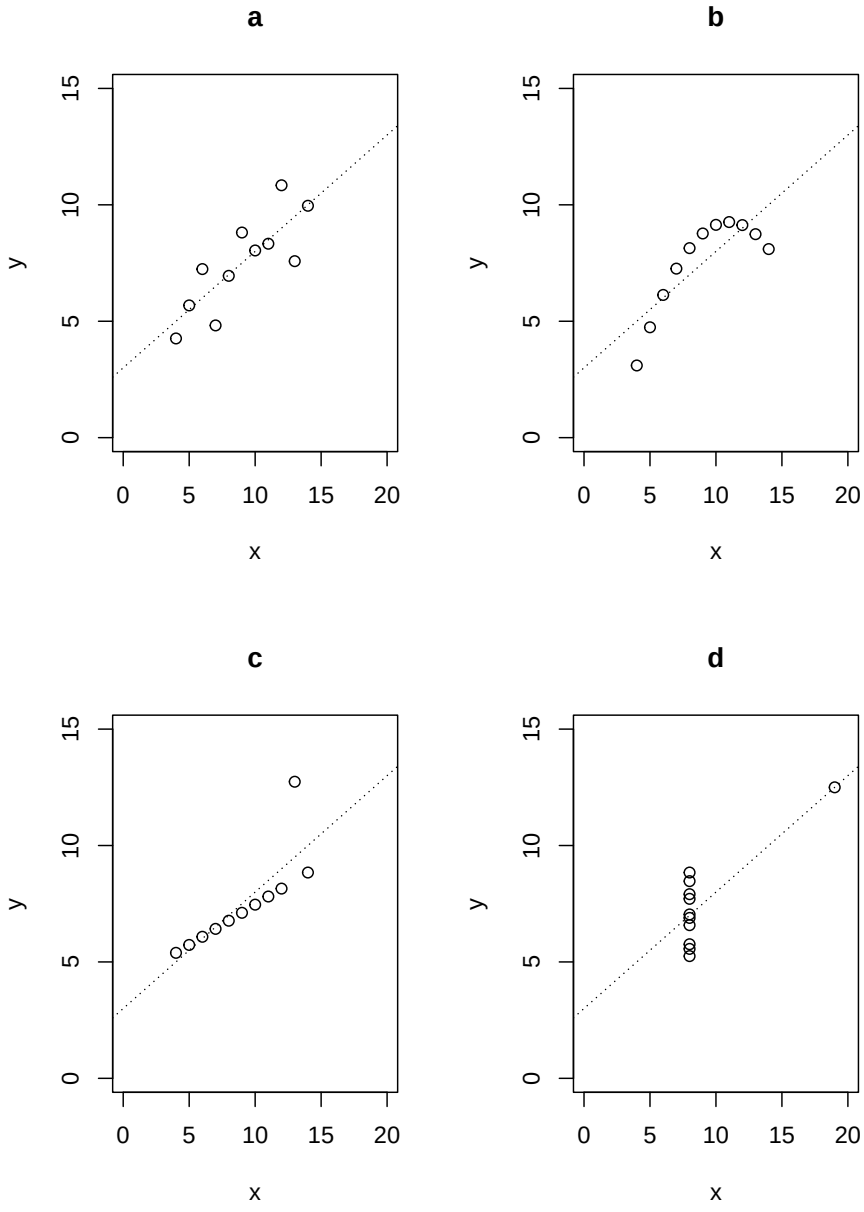
```
## [1] 0.01637254
```

Thus, correlation is (almost) zero even though x and y are dependent variables. This is because correlation only measures linear association, and the relationship between x and y is nonlinear.

5.4.2 Anscombe data sets

Using correlation, and more generally linear models, blindly can easily hide the underlying complexity of the analysed data. This is demonstrated by the classic "Anscombe quartet" of data sets presented in his 1973 paper[3] -

[3] Anscombe, F. J. 1973. Graphs in statistical analysis. The American Statistician 27:17-21, http://doi.org/10.1080/00031305.1973.10478966



As evident from the scatter plots the relationship between the two variables x and y is very different in the four cases! However, intriguingly all four data sets share exactly the same linear characteristics and summary statistics:

- Means $m_x = 9$ and $m_y = 7.5$

- Variances $s_x^2 = 11$ and $s_y^2 = 4.13$
- Correlation $r = 0.8162$
- Linear model fit with intercept $a = 3.0$ and slope $b = 0.5$

Thus, in actual data analysis it is always a **good idea to inspect the data visually** to get a first impression whether using a linear model makes sense.

In the above only data set "a" follows a linear model. Data set "b" represents a quadratic relationship. Data set "c" is linear but with an outlier that disturbs the linear relationship. Finally data set "d" also contains an outlier but also represent a case where y is (apart from the outlier) is not dependent on x.

In the Worksheet 10 a more recent version of the Anscombe quartet will be analysed in the form of the "datasauRus" dozen - 13 highly nonlinear datasets that all share the same linear characteristics.

5.5 Mutual information as generalisation of correlation

5.5.1 Overview

A more general way than the vector correlation to measure multivariate association is mutual information (MI) which not only covers linear but also non-linear associations.

As we will see below the Rozeboom vector correlation arises naturally when computing the MI for the multivariate normal distribution, hence MI also recovers well-known measures of linear association (including multiple correlation and simple correlation), thus truly generalising correlation as measure of association.

5.5.2 Definition of mutual information

Recall the definition of Kullback-Leibler divergence, or relative entropy, between two distributions:

$$D_{\mathrm{KL}}(F, G) := \mathrm{E}_F \log\left(\frac{f(\boldsymbol{x})}{g(\boldsymbol{x})}\right)$$

Here F plays the role of the reference distribution and G is an approximating distribution, with f and g being the corresponding density functions (see MATH20802 Statistical Methods).

The *Mutual Information* (MI) between two random variables $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as the KL divergence between the corresponding joint distribution and the product distribution:

$$\mathrm{MI}(\boldsymbol{x}, \boldsymbol{y}) = D_{\mathrm{KL}}(F_{\boldsymbol{x},\boldsymbol{y}}, F_{\boldsymbol{x}} F_{\boldsymbol{y}}) = \mathrm{E}_{F_{\boldsymbol{x},\boldsymbol{y}}} \log\left(\frac{f(\boldsymbol{x}, \boldsymbol{y})}{f(\boldsymbol{x})\, f(\boldsymbol{y})}\right).$$

Thus, MI measures how well the joint distribution can be approximated by the product distribution (which would be the appropriate joint distribution if $\boldsymbol{x}$ and $\boldsymbol{y}$ are independent). Since MI is an application of KL divergence is shares all its properties. In particular, $\mathrm{MI}(\boldsymbol{x}, \boldsymbol{y}) = 0$ implies that the joint distribution and product distributions are the same. Hence the two random variables $\boldsymbol{x}$ and $\boldsymbol{y}$ are independent if the mutual information vanishes.

5.5.3 Mutual information between two normal scalar variables

The KL divergence between two multivariate normal distributions F_{ref} and F is

$$D_{\mathrm{KL}}(F_{\mathrm{ref}}, F) = \frac{1}{2}\left\{(\boldsymbol{\mu} - \boldsymbol{\mu}_{\mathrm{ref}})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} - \boldsymbol{\mu}_{\mathrm{ref}}) + \mathrm{Tr}\left(\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_{\mathrm{ref}}\right) - \log\det\left(\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_{\mathrm{ref}}\right) - d\right\}$$

This allows compute the mutual information $\mathrm{MI}_{\mathrm{norm}}(x, y)$ between two univariate random variables x and y that are correlated and assumed to be jointly bivariate normal. Let $\boldsymbol{z} = (x, y)^T$. The joint bivariate normal distribution is characterised by the mean $\mathrm{E}(\boldsymbol{z}) = \boldsymbol{\mu} = (\mu_x, \mu_y)^T$ and the covariance matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_x^2 & \rho\,\sigma_x\sigma_y \\ \rho\,\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$

where $\mathrm{Cor}(x, y) = \rho$. If x and y are independent then $\rho = 0$ and

$$\boldsymbol{\Sigma}_{\mathrm{indep}} = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}.$$

The product

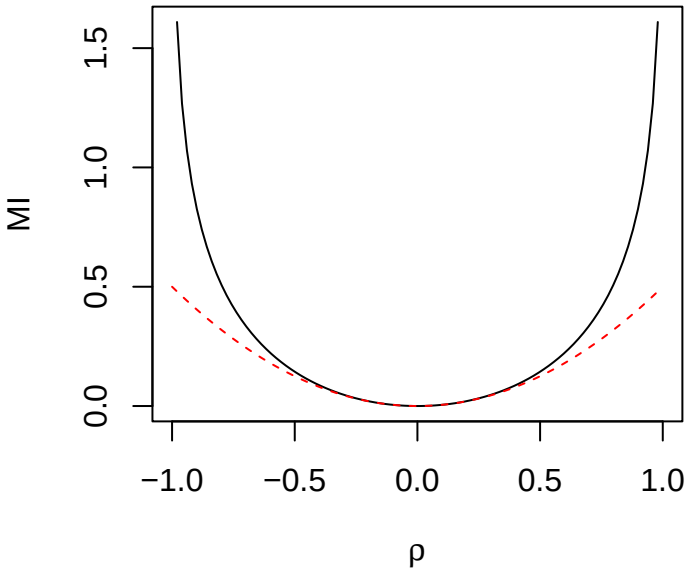
$$\boldsymbol{A} = \boldsymbol{\Sigma}_{\mathrm{indep}}^{-1}\boldsymbol{\Sigma} = \begin{pmatrix} 1 & \rho\frac{\sigma_y}{\sigma_x} \\ \rho\frac{\sigma_x}{\sigma_y} & 1 \end{pmatrix}$$

has trace $\mathrm{Tr}(\boldsymbol{A}) = 2$ and determinant $\det(\boldsymbol{A}) = 1 - \rho^2$.

With this the mutual information between x and y can be computed as

$$\begin{aligned}
\mathrm{MI}_{\mathrm{norm}}(x, y) &= D_{\mathrm{KL}}(N(\boldsymbol{\mu}, \boldsymbol{\Sigma}), N(\boldsymbol{\mu}, \boldsymbol{\Sigma}_{\mathrm{indep}})) \\
&= \frac{1}{2}\left\{\mathrm{Tr}\left(\boldsymbol{\Sigma}_{\mathrm{indep}}^{-1}\boldsymbol{\Sigma}\right) - \log\det\left(\boldsymbol{\Sigma}_{\mathrm{indep}}^{-1}\boldsymbol{\Sigma}\right) - 2\right\} \\
&= \frac{1}{2}\left\{\mathrm{Tr}(\boldsymbol{A}) - \log\det(\boldsymbol{A}) - 2\right\} \\
&= -\frac{1}{2}\log(1 - \rho^2) \\
&\approx \frac{\rho^2}{2}
\end{aligned}$$

Thus $\mathrm{MI}_{\mathrm{norm}}(x, y)$ is is a one-to-one function of the squared correlation ρ^2 between x and y:



For small values of correlation $2\,\mathrm{MI}_{\mathrm{norm}}(x, y) \approx \rho^2$.

5.5.4 Mutual information between two normally distributed random vectors

The mutual information $\mathrm{MI}_{\mathrm{norm}}(\boldsymbol{x}, \boldsymbol{y})$ between two multivariate normal random vector $\boldsymbol{x}$ and $\boldsymbol{y}$ can be computed in a similar fashion as in the bivariate case.

Let $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})^T$ with dimension $d = p + q$. The joint multivariate normal distribution is characterised by the mean $\mathrm{E}(\boldsymbol{z}) = \boldsymbol{\mu} = (\boldsymbol{\mu}_x^T, \boldsymbol{\mu}_y^T)^T$ and the covariance matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_x & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{xy}^T & \boldsymbol{\Sigma}_y \end{pmatrix}.$$

If $\boldsymbol{x}$ and $\boldsymbol{y}$ are independent then $\boldsymbol{\Sigma}_{xy} = 0$ and

$$\boldsymbol{\Sigma}_{\mathrm{indep}} = \begin{pmatrix} \boldsymbol{\Sigma}_x & 0 \\ 0 & \boldsymbol{\Sigma}_y \end{pmatrix}.$$

The product

$$\begin{aligned}\boldsymbol{A} = \boldsymbol{\Sigma}_{\text{indep}}^{-1}\boldsymbol{\Sigma} &= \begin{pmatrix} \boldsymbol{I}_p & \boldsymbol{\Sigma}_x^{-1}\boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_y^{-1}\boldsymbol{\Sigma}_{yx} & \boldsymbol{I}_q \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{I}_p & \boldsymbol{V}_x^{-1/2}\boldsymbol{P}_x^{-1}\boldsymbol{P}_{xy}\boldsymbol{V}_y^{1/2} \\ \boldsymbol{V}_y^{-1/2}\boldsymbol{P}_y^{-1}\boldsymbol{P}_{yx}\boldsymbol{V}_x^{1/2} & \boldsymbol{I}_q \end{pmatrix}\end{aligned}$$

has trace $\text{Tr}(\boldsymbol{A}) = d$ and determinant

$$\begin{aligned}\det(\boldsymbol{A}) &= \det(\boldsymbol{I}_p - \boldsymbol{K}\boldsymbol{K}^T) \\ &= \det(\boldsymbol{I}_q - \boldsymbol{K}^T\boldsymbol{K})\end{aligned}$$

with $\boldsymbol{K} = \boldsymbol{P}_x^{-1/2}\boldsymbol{P}_{xy}\boldsymbol{P}_y^{-1/2}$. With $\lambda_1, \ldots, \lambda_m$ the singular values of $\boldsymbol{K}$ (i.e. the canonical correlations between $\boldsymbol{x}$ and $\boldsymbol{y}$) we get

$$\det(\boldsymbol{A}) = \prod_{i=1}^{m}(1 - \lambda_i^2)$$

The mutual information between $\boldsymbol{x}$ and $\boldsymbol{y}$ is then

$$\begin{aligned}\text{MI}_{\text{norm}}(\boldsymbol{x}, \boldsymbol{y}) &= D_{\text{KL}}(N(\boldsymbol{\mu}, \boldsymbol{\Sigma}), N(\boldsymbol{\mu}, \boldsymbol{\Sigma}_{\text{indep}})) \\ &= \frac{1}{2}\left\{\text{Tr}\left(\boldsymbol{\Sigma}_{\text{indep}}^{-1}\boldsymbol{\Sigma}\right) - \log\det\left(\boldsymbol{\Sigma}_{\text{indep}}^{-1}\boldsymbol{\Sigma}\right) - d\right\} \\ &= \frac{1}{2}\left\{\text{Tr}(\boldsymbol{A}) - \log\det(\boldsymbol{A}) - d\right\} \\ &= -\frac{1}{2}\sum_{i=1}^{m}\log(1 - \lambda_i^2)\end{aligned}$$

Note that $\text{MI}_{\text{norm}}(\boldsymbol{x}, \boldsymbol{y})$ is simply the sum of the MIs resulting from the individual canonical correlations λ_i with the same functional form as in the bivariate normal case.

By comparison with the squared Rozeboom vector correlation coefficient ρ_{xy}^2 we recognize that

$$\text{MI}_{\text{norm}}(\boldsymbol{x}, \boldsymbol{y}) = -\frac{1}{2}\log(1 - \rho_{xy}^2) \approx \frac{1}{2}\rho_{xy}^2$$

Thus, in the multivariate case $\text{MI}_{\text{norm}}(\boldsymbol{xy})$ has again exactly the same functional relationship with the vector correlation $\rho_{x,y}^2$ as the $\text{MI}_{\text{norm}}(x, y)$ for two univariate variables with squared Pearson correlation ρ^2.

Thus, Rozeboom's vector correlation is directly linked to mutual information for jointly multivariate normally distributed variables.

5.5.5 Using MI for variable selection

A very general way to write down a model predicting $\boldsymbol{y}$ by $\boldsymbol{x}$ is as follows:

- $F_{\boldsymbol{y}|\boldsymbol{x}}$ is a conditional distribution of $\boldsymbol{y}$ given predictors $\boldsymbol{x}$ and
- $F_{\boldsymbol{y}}$ is the marginal distribution of $\boldsymbol{y}$ without predictors.

Typically $F_{\boldsymbol{y}|\boldsymbol{x}}$ is a complex model and $F_{\boldsymbol{y}}$ a simple model (no predictors). Note that the predictive model can assume any form (incl. nonlinear).

Intriguingly the expected KL divergence between the conditional and the marginal distribution

$$\mathrm{E}_{F_{\boldsymbol{x}}} D_{\mathrm{KL}}(F_{\boldsymbol{y}|\boldsymbol{x}}, F_{\boldsymbol{y}}) = \mathrm{MI}(\boldsymbol{x}, \boldsymbol{y})$$

is equal to mutual information between $\boldsymbol{x}$ and $\boldsymbol{y}$! Thus $\mathrm{MI}(\boldsymbol{x}, \boldsymbol{y})$ measures the impact of conditioning. If the MI is small (i.e. close to zero) then $\boldsymbol{x}$ is not useful in predicting $\boldsymbol{y}$.

The above identity can be verified as follows. The KL divergence between $F_{\boldsymbol{y}|\boldsymbol{x}}$ and $F_{\boldsymbol{y}}$ is given by

$$D_{\mathrm{KL}}(F_{\boldsymbol{y}|\boldsymbol{x}}, F_{\boldsymbol{y}}) = \mathrm{E}_{F_{\boldsymbol{y}|\boldsymbol{x}}} \log\left(\frac{f(\boldsymbol{y}|\boldsymbol{x})}{f(\boldsymbol{y})}\right),$$

which is a random variable since it depends on $\boldsymbol{x}$. Taking the expectation with regard to $F_{\boldsymbol{x}}$ (the distribution of $\boldsymbol{x}$) we get

$$\begin{aligned}\mathrm{E}_{F_{\boldsymbol{x}}} D_{\mathrm{KL}}(F_{\boldsymbol{y}|\boldsymbol{x}}, F_{\boldsymbol{y}}) &= \mathrm{E}_{F_{\boldsymbol{x}}} \mathrm{E}_{F_{\boldsymbol{y}|\boldsymbol{x}}} \log\left(\frac{f(\boldsymbol{y}|\boldsymbol{x}) f(\boldsymbol{x})}{f(\boldsymbol{y}) f(\boldsymbol{x})}\right) \\ &= \mathrm{E}_{F_{\boldsymbol{x},\boldsymbol{y}}} \log\left(\frac{f(\boldsymbol{x}, \boldsymbol{y})}{f(\boldsymbol{y}) f(\boldsymbol{x})}\right) = \mathrm{MI}(\boldsymbol{x}, \boldsymbol{y}).\end{aligned}$$

Because of this link of MI with conditioning the MI between response and predictor variables is often used for variable and feature selection in general models.

5.5.6 Other measures of general dependence

In principle, MI can be computed for any distribution and model and thus applies to both normal and non-normal models, and to both linear and nonlinear relationships.

Besides mutual information there are others measures of general dependence between multivariate random variables.

Two important measures to capture nonlinear association that have been proposed in recent literature are

i) distance correlation and
ii) the maximal information coefficient (MIC and MIC_e).

5.6 Graphical models

5.6.1 Purpose

Graphical models combine features from

- graph theory
- probability
- statistical inference

The literature on graphical models is huge, we focus here only on two commonly used models:

- DAGs (directed acyclic graphs), all edges are directed, no directed loops (i.e. no cycles, hence "acyclic")
- GGM (Gaussian graphical models), all edges are undirected

Graphical models provide probabilistic models for trees and for networks, with random variables represented by nodes in the graphs, and branches representing conditional dependencies. In this regard they generalise both the tree-based clustering approaches as well as the probabilistic non-hierarchical methods (GMMs).

However, the class of graphical models goes much beyond simple unsupervised learning models. It also includes regression, classification, time series models etc. See e.g. the reference book by Murphy (2012).

5.6.2 Basic notions from graph theory

- Mathematically, a graph $G = (V, E)$ consists of a a set of vertices or nodes $V = \{v_1, v_2, \ldots\}$ and a set of branches or edges $E = \{e_1, e_2, \ldots\}$.
- Edges can be undirected or directed.
- Graphs containing only directed edges are directed graphs, and likewise graphs containing only undirected edges are called undirected graphs. Graphs containing both directed and undirected edges are called partially directed graphs.
- A path is a sequence of of vertices such that from each of its vertices there is an edge to the next vertex in the sequence.
- A graph is connected when there is a path between every pair of vertices.
- A cycle is a path in a graph that connects a node with itself.
- A connected graph with no cycles is a called a tree.
- The degree of a node is the number of edges it connects with. If edges are all directed the degree of a node is the sum of the in-degree and out-degree, which counts the incoming and outgoing edges, respectively.
- External nodes are nodes with degree 1. In a tree-structured graph these are also called leaves.

Some notions are only relevant for graphs with directed edges:

- In a directed graph the parent node(s) of vertex v is the set of nodes $\text{pa}(v)$ directly connected to v via edges directed from the parent node(s) towards v.
- Conversely, v is called a child node of $\text{pa}(v)$. Note that a parent node can have several child nodes, so v may not be the only child of $\text{pa}(v)$.
- In a directed tree graph, each node has only a single parent, except for one particular node that has no parent at all (this node is called the root node).
- A DAG, or directed acyclic graph, is a directed graph with no directed cycles. A (directed) tree is a special version of a DAG.

5.6.3 Probabilistic graphical models

A graphical model uses a graph to describe the relationship between random variables $x_1, \ldots, x_d$. The variables are assumed to have a joint distribution with density/mass function $p(x_1, x_2, \ldots, x_d)$. Each random variable is placed in a node of the graph.

The structure of the graph and the type of the edges connecting (or not connecting) any pair of nodes/variables is used to describe the conditional dependencies, and to simplify the joint distribution.

Thus, a graphical model is in essence a visualisation of the joint distribution using structural information from the graph helping to understand the mutual relationship among the variables.

5.6.4 Directed graphical models

In a **directed graphical model** the graph structure is assumed to be a DAG (or a directed tree, which is also a DAG).

Then the joint probability distribution can be factorised into a *product of conditional probabilities* as follows:

$$p(x_1, x_2, \ldots, x_d) = \prod_i p(x_i | \text{pa}(x_i))$$

Thus, the overall joint probability distribution is specified by local conditional distributions and the graph structure, with the directions of the edges providing the information about parent-child node relationships.

Probabilistic DAGs are also known as "Bayesian networks".

Idea: by trying out all possible trees/graphs and fitting them to the data using maximum likelihood (or Bayesian inference) we hope to be able identify the graph structure of the data-generating process.

Challenges

1) in the tree/network the internal nodes are usually not known, and thus have to be treated as *latent* variables.

Answer: To impute the states at these nodes we may use the EM algorithm as in GMMs (which in fact can be viewed as graphical models, too!).

2) If we treat the internal nodes as unknowns we need to marginalise over the internal nodes, i.e. we need to sum / integrate over all possible set of states of the internal nodes!

Answer: This can be handled very effectively using the **Viterbi algorithm** which is essentially an application of the generalised distributive law. In particular for tree graphs this means that the summations occurs locally at each node and propagates recursively across the tree.

3) In order to infer the tree or network structure the space of all trees or networks need to be explored. This is not possible in an exhaustive fashion unless the number of variables in the tree is very small.

Answer: Solution: use heuristic approaches for tree and network search!

4) Furthermore, there exist so-called "equivalence classes" of graphical models, i.e. sets of graphical models that share the same joint probability distribution. Thus, all graphical models within the same equivalence class cannot be distinguished from observational data, even with infinite sample size!

Answer: this is a fundamental mathematical problem of identifiability so there is now way around this issue. However, on the positive side, this also implies that the search through all graphical models can be restricted to finding the so-called "essential graph" (e.g. https://projecteuclid.org/euclid.aos/1031833662)

Conclusion: using directed graphical models for structure discovery is very time consuming and computationally demanding for anything but small toy data sets.

This also explains why heuristic and non-model based approaches (such as hierarchical clustering) are so popular even though full statistical modelling is in principle possible.

5.6.5 Undirected graphical models

Another class of graphical models are models that contain only undirected edges. These **undirected graphical models** are used to represent the pairwise conditional (in)dependencies among the variables in the graph, and the resulting model is therefore also called **conditional independence graph**.

Suppose x_i and x_j are two random variables/nodes from $\{x_1, \ldots, x_d\}$, and the set $\{x_k\}$ represents all other variables/nodes with $k \neq i$ and $k \neq j$. Then the variables x_i and x_j are conditionally independent given all the other variables $\{x_k\}$

$$x_i \perp\!\!\!\perp x_j | \{x_k\}$$

if the joint probability density for all variables $\{x_1, \ldots, x_d\}$ factorises as

$$p(x_1, x_2, \ldots, x_d) = p(x_i|\{x_k\})\, p(x_j|\{x_k\})\, p(\{x_k\})\,.$$

or equivalently

$$\frac{p(x_i, x_j, \ldots, x_d)}{p(\{x_k\})} = p(x_i, x_j|\{x_k\}) = p(x_i|\{x_k\})\, p(x_j|\{x_k\})\,.$$

In the corresponding conditional independence graph note there is no edge between x_i and x_j, as in such a graph *missing edges correspond to conditional independence* between the respective non-connected nodes.

5.6.5.1 Gaussian graphical model

Assuming that $x_1, \ldots, x_d$ are jointly normally distributed, i.e. $\boldsymbol{x} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, it turns out that it is straightforward to identify the pairwise conditional independencies. From $\boldsymbol{\Sigma}$ we first obtain the precision matrix

$$\boldsymbol{\Omega} = (\omega_{ij}) = \boldsymbol{\Sigma}^{-1}\,.$$

Crucially, it can be shown that $\omega_{ij} = 0$ implies $x_i \perp\!\!\!\perp x_j \,|\, \{x_k\}$. Hence, from the precision matrix $\boldsymbol{\Omega}$ we can directly read off all the pairwise conditional independencies among the variables $x_1, x_2, \ldots, x_d$.

Often, the covariance matrix $\boldsymbol{\Sigma}$ is dense (few zeros) but the corresponding precision matrix $\boldsymbol{\Omega}$ is sparse (many zeros).

The conditional independence graph computed for normally distributed variables is called a **Gaussian graphical model**, or short **GGM**. A further alternative name commonly used is **covariance selection model**.

5.6.5.2 Related quantity: partial correlation

From the precision matrix $\boldsymbol{\Omega}$ we can also compute the matrix of pairwise full conditional *partial correlations*:

$$\rho_{ij|\text{rest}} = -\frac{\omega_{ij}}{\sqrt{\omega_{ii}\omega_{jj}}}$$

which is essentially the standardised precision matrix (similar to correlation but with an extra minus sign!)

The partial correlations lie in the range between -1 and +1, $\rho_{ij|\text{rest}} \in [-1, 1]$, just like standard correlations.

If $\boldsymbol{x}$ is multivariate normal then $\rho_{ij|\text{rest}} = 0$ indicates conditional independence between x_i and x_j.

Regression interpretation: partial correlation is the correlation that remains between the two variables if the effect of the other variables is "regressed away". In other words, the partial correlation is exactly equivalent to the correlation between the residuals that remain after regressing x_i on the variables $\{x_k\}$ and x_j on $\{x_k\}$.

5.6.6 Null distribution of the empirical correlation coefficient

Suppose we have two uncorrelated random variables x and y with $\rho = \text{Cor}(x, y) = 0$. After observing data $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ we compute the the empirical covariance matrix $\hat{\mathbf{\Sigma}}_{xy}$ and from it the empirical correlation coefficient $r = \widehat{\text{Cor}}(x, y)$.

The distribution of the empirical correlation assuming $\rho = 0$ is useful as null-model for testing whether the underlying correlation is in fact zero having observed empirical correlation r. If x and y are normally distributed with $\rho = 0$ the distribution of the empirical correlation r has mean $\text{E}(r) = 0$ and variance $\text{Var}(r) = \frac{1}{\kappa}$. Here κ is the degree of freedom of the null distribution which for standard correlation is $\kappa = n - 1$. Furthermore, the squared empirical correlation is distributed according to a Beta distribution

$$r^2 \sim \text{Beta}\left(\frac{1}{2}, \frac{\kappa - 1}{2}\right)$$

For partial correlation the null distribution of r^2 has the same form but with a different degree of freedom. Specifically, κ is reduced by the number of variables being conditioned on. If for d dimensions we condition on $d - 2$ variables the resulting degree of freedom is $\kappa = n - 1 - (d - 2) = n - d + 1$. For $d = 2$ we get back the degree of freedom for standard empirical correlation.

5.6.7 Algorithm for learning GGMs

From the above we can devise a simple algorithm to learn Gaussian graphical model (GGM) from data:

1. Estimate covariance $\hat{\mathbf{\Sigma}}$ (in such a way that it is invertible!)
2. Compute corresponding partial correlations
3. If $\hat{\rho}_{ij|\text{rest}} \approx 0$ then there is (approx.) conditional independence between x_i and x_j.

The test for conditional independence is done by statistical testing for vanishing partial correlation. Specifically, we compute the p-value assuming that the true underlying partial correlation is zero and then decide whether to reject the null assumption of zero partial correlation.

If there are many edges tested simultaneously we may need to adjust (i.e reduce) the test threshold, for example applying Bonferroni or FDR methods.

5.6.8 Example: exam score data

This is a data set from Mardia et al. (1979) and features $d = 5$ variables measured on $n = 88$ subjects.

Correlations (rounded to 2 digits):

```
##            mechanics vectors algebra analysis statistics
## mechanics       1.00    0.55    0.55     0.41       0.39
## vectors         0.55    1.00    0.61     0.49       0.44
## algebra         0.55    0.61    1.00     0.71       0.66
## analysis        0.41    0.49    0.71     1.00       0.61
## statistics      0.39    0.44    0.66     0.61       1.00
```

Partial correlations (rounded to 2 digits):

```
##            mechanics vectors algebra analysis statistics
## mechanics       1.00    0.33    0.23     0.00       0.02
## vectors         0.33    1.00    0.28     0.08       0.02
## algebra         0.23    0.28    1.00     0.43       0.36
## analysis        0.00    0.08    0.43     1.00       0.25
## statistics      0.02    0.02    0.36     0.25       1.00
```

Note that there are no zero correlations but there are **four partial correlations close to 0**, indicating **conditional independence** between:

- analysis and mechanics,
- statistics and mechanics,
- analysis and vectors, and
- statistics and vectors.

The can be verified by computing the normal p-values for the partial correlations (with $\kappa = 84$ as degree of freedom):

```
##            mechanics vectors algebra analysis statistics
## mechanics         NA   0.002   0.034    0.988      0.823
## vectors           NA      NA   0.009    0.477      0.854
## algebra           NA      NA      NA    0.000      0.001
## analysis          NA      NA      NA       NA      0.020
## statistics        NA      NA      NA       NA         NA
```

There are six edges with small p-value (smaller than say 0.05) and these correspond to the edges for which the null assumption of zero partial correlation can be rejected so that out of ten possible edges four are not statistically significant. Therefore the conditional independence graph looks as follows:

Chapter 6

Nonlinear and nonparametric models

In the last part of the module we discuss methods that go beyond the linear parametric methods prevalent in classical multivariate statistics.

Relevant textbooks:

The lectures for much of this part of the module follow selected chapters from the following text books:

- James et al. (2021) *An introduction to statistical learning with applications in R (2nd edition)*. Springer.
- Rogers and Girolami (2017) *A first course in machine learning (2nd edition)*. CRC Press.

Please study the relevant section and chapters as indicated below in each subsection!

6.1 Random forests

Another widely used approach for prediction in nonlinear settings is the method of random forests.

Relevant reading:

Please read: James et al. (2021) **Chapter 8 "Tree-Based Methods"**

Specifically:

- Section 8.1 The Basics of Decision Trees
- Section 8.2.1 Bagging
- Section 8.2.2 Random Forests

6.1.1 Stochastic vs. algorithmic models

Two cultures in statistical modelling: stochastic vs. algorithmic models

Classic discussion paper by Leo Breiman (2001): Statistical modeling: the two cultures. Statistical Science **16**:199–231. https://doi.org/10.1214/ss/1009213726

This paper has recently be revisited in the following discussion paper by Efron (2020) and discussants: Prediction, estimation, and attribution. JASA **115**:636–677. https://doi.org/10.1080/01621459.2020.1762613

6.1.2 Random forests

Proposed by Leo Breimann in 2001 as application of "bagging" (Breiman 1996) to decision trees.

Basic idea:

- A single decision tree is unreliable and unstable (weak predictor/classifier).
- Use boostrap to generate multiple decision trees (="forest")
- Average over predictions from all tree (="bagging", bootstrap aggregation)

The averaging procedure has the effect of variance stabilisation. Intringuingly, averaging across all decision trees dramatically improves the overall prediction accuracy!

The Random Forests approach is an example of an **ensemble method** (since it is based on using an "ensemble" of trees).

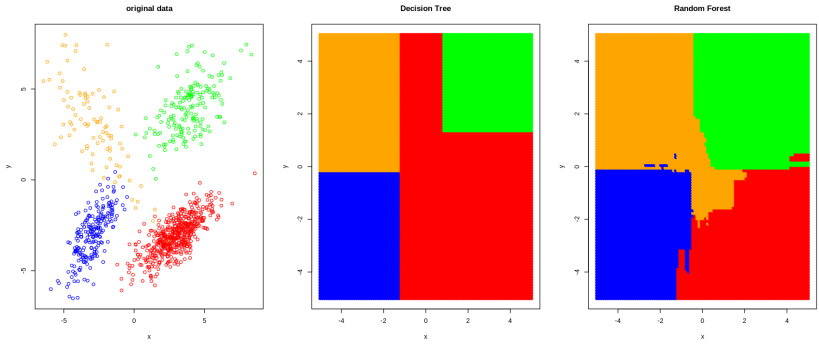
Variations: boosting, XGBoost (https://xgboost.ai/)

Random forests will be applied in Worksheet 11.

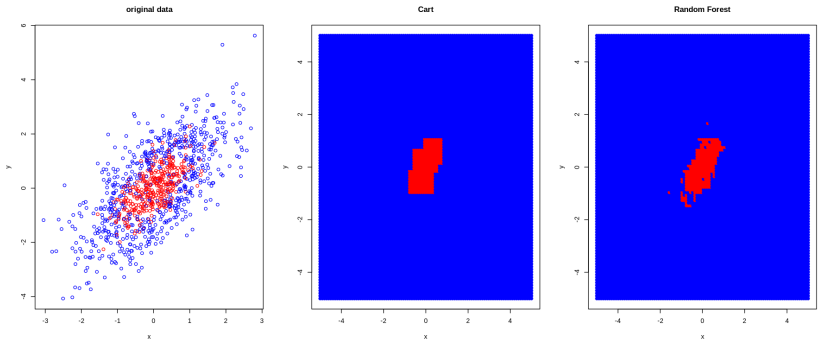
They are computationally expensive but typically perform very well!

6.1.3 Comparison of decision boundaries: decision tree vs. random forest

Non-nested case:



Nested case:



Compare also with the decision boundaries for LDA and QDA (Chapter 4).

6.2 Gaussian processes

Gaussian processes offer another nonparametric approach to model nonlinear dependencies. They provide a probabilistic model for the unknown nonlinear function.

Relevant reading:

Please read: Rogers and Girolami (2017) **Chapter 8: Gaussian processes.**

6.2.1 Main concepts

- Gaussian processes (GPs) belong the the family of **Bayesian nonparametric models**
- Idea:
  - start with prior over a function (!),
  - then condition on observed data to get posterior distribution (again over a function)
- GPs use an infinitely dimensional multivariate normal distribution as prior

6.2.2 Conditional multivariate normal distribution

GPs make use of the fact that marginal and conditional distributions of a multivariate normal distribution are also multivariate normal.

Multivariate normal distribution:

$$\boldsymbol{z} \sim N_d(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Assume:

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix}$$

with

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}$$

and

$$\boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_1 & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{12}^T & \boldsymbol{\Sigma}_2 \end{pmatrix}$$

with corresponding dimensions d_1 and d_2 and $d_1 + d_2 = d$.

Marginal distributions:

Any subset of $\boldsymbol{z}$ is also multivariate normally distributed. Specifically,

$$\boldsymbol{z}_1 \sim N_{d_1}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$$

and

$$\boldsymbol{z}_2 \sim N_{d_2}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$$

Conditional multivariate normal:

The conditional distribution is also multivariate normal:

$$z_1|z_2 = z_{1|2} \sim N_{d_1}(\boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2})$$

with

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_2^{-1}(z_2 - \boldsymbol{\mu}_2)$$

and

$$\boldsymbol{\Sigma}_{1|2} = \boldsymbol{\Sigma}_1 - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_{12}^T$$

$z_{1|2}$ and $\boldsymbol{\mu}_{1|2}$ have dimension $d_1 \times 1$ and $\boldsymbol{\Sigma}_{1|2}$ has dimension $d_1 \times d_1$, i.e. the same dimension as the unconditioned variables.

You may recall the above formula in the context of linear regression, with $y = z_1$ and $\boldsymbol{x} = z_2$ so that the conditional mean becomes

$$\begin{aligned} \mathrm{E}(y|\boldsymbol{x}) &= \boldsymbol{\mu}_y + \boldsymbol{\Sigma}_{y\boldsymbol{x}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}}) \\ &= \beta_0 + \boldsymbol{\beta}^T\boldsymbol{x} \end{aligned}$$

with $\boldsymbol{\beta} = \boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{x}y}$ and $\beta_0 = \boldsymbol{\mu}_y - \boldsymbol{\beta}^T\boldsymbol{\mu}_{\boldsymbol{x}}$, and the corresponding conditional variance is

$$\mathrm{Var}(y|\boldsymbol{x}) = \sigma_y^2 - \boldsymbol{\Sigma}_{y\boldsymbol{x}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{\Sigma}_{\boldsymbol{x}y} \, .$$

6.2.3 Covariance functions and kernels

The GP prior is an infinitely dimensional multivariate normal distribution with mean zero and the **covariance specified by a function** $k(x, x')$:

A widely used covariance function is

$$k(x, x') = \mathrm{Cov}(x, x') = \sigma^2 e^{-\frac{(x-x')^2}{2l^2}}$$

This is known as the **squared-exponential kernel** or **Radial-basis function (RBF) kernel**.

Note that this kernel implies

- $k(x, x) = \mathrm{Var}(x) = \sigma^2$ and
- $\mathrm{Cor}(x, x') = e^{-\frac{(x-x')^2}{2l^2}}$.

The parameter l in the RBF kernel is the length scale parameter and describes the "wigglyness" or "stiffness" of the resulting function. Small values of l correspond to more complex, more wiggly functions, and to low spatial correlation, as the correlation decreases quicker with distance, and large values correspond to more rigid, stiffer functions, with longer range spatial correlation (note that in a time series context this would be called autocorrelation).

There are many other kernel functions, including linear, polynomial or periodic kernels.

6.2.4 GP model

Nonlinear regression in the GP approach is conceptually very simple:

- start with multivariate prior
- then condition on the observed data
- the resulting conditional multivariate normal can used to predict the function values at any unobserved values
- the conditional variance can be used to compute credible intervals for predictions.

GP regression also provides a direct link with classical Bayesian linear regression (using a linear kernel).

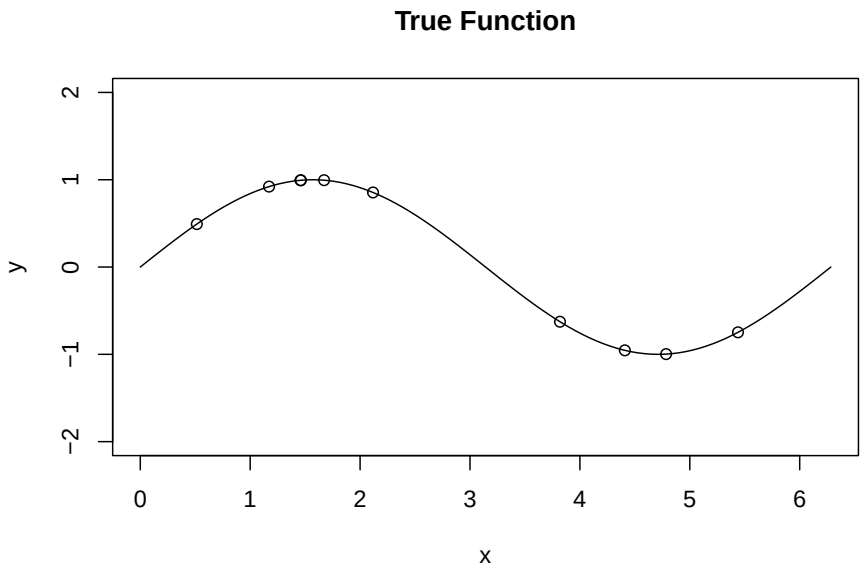
Drawbacks: computationally expensive, typically $O(n^3)$ because of the matrix inversion.

6.2.5 Gaussian process example

We now show how to apply Gaussian processes in R justing using standard matrix calculations.

Our aim is to estimate the following nonlinear function from a number of observations. Note that initially we assume that there is no additional noise (so the observations lie directly on the curve):

```
truefunc = function(x) sin(x)
XLIM = c(0, 2*pi)
YLIM = c(-2, 2)

n2 = 10
x2 = runif(n2, min=XLIM[1], max=XLIM[2])
y2 = truefunc(x2)  # no noise

curve( truefunc(x), xlim=XLIM, ylim=YLIM, xlab="x", ylab="y",
      main="True Function")
points(x2, y2)
```



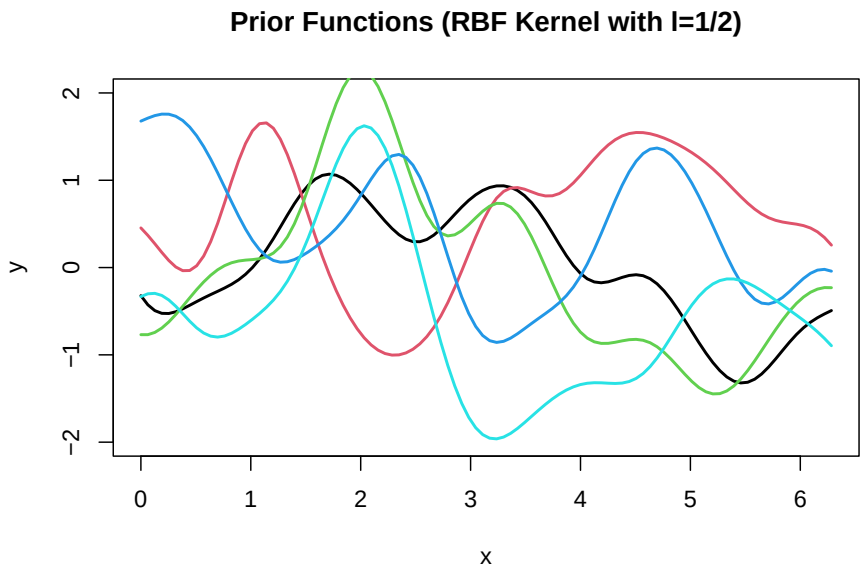
Next we use the RFB kernel as the prior covariance and also assume that the prior has mean zero:

```
# RBF kernel
rbfkernel = function(xa, xb, s2=1, l=1/2) s2*exp(-1/2*(xa-xb)^2/l^2)
kfun.mat = function(xavec, xbvec, FUN=rbfkernel)
  outer(X=as.vector(xavec), Y=as.vector(xbvec), FUN=FUN)

# prior mean
mu.vec = function(x) rep(0, length(x))
```

We can now visualise the functions sampled from the multivariate normal prior:

```
# grid of x-values
n1 = 100
x1 = seq(XLIM[1], XLIM[2], length.out=n1)

# unconditioned covariance and mean (unobserved samples x1)
K1 = kfun.mat(x1, x1)
m1 = mu.vec(x1)

## sample functions from GP prior
B = 5
library("MASS") # for mvrnorm
y1r = t(mvrnorm(B, mu = m1, Sigma=K1))
```

```
plot(x1, y1r[,1], type="l", lwd=2, ylab="y", xlab="x", ylim=YLIM,
  main="Prior Functions (RBF Kernel with l=1/2)")
for(i in 2:B)
  lines(x1, y1r[,i], col=i, lwd=2)
```



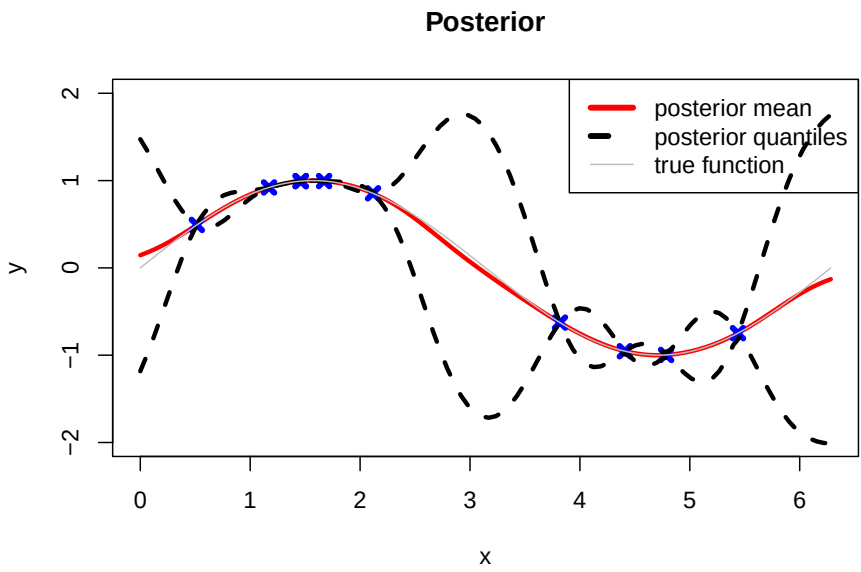
Now we compute the posterior mean and variance by conditioning on the observations:

```
# unconditioned covariance and mean (observed samples x2)
K2 = kfun.mat(x2, x2)
m2 = mu.vec(x2)
iK2 = solve(K2) # inverse

# cross-covariance
K12 = kfun.mat(x1, x2)

# Conditioning: x1 conditioned on x2

# conditional mean
m1.2 = m1 + K12 %*% iK2 %*% (y2 - m2)

# conditional variance
K1.2 = K1 - K12 %*% iK2 %*% t(K12)
```

Now we can plot the posterior mean and upper and lower bounds of a 95%

credible interval:

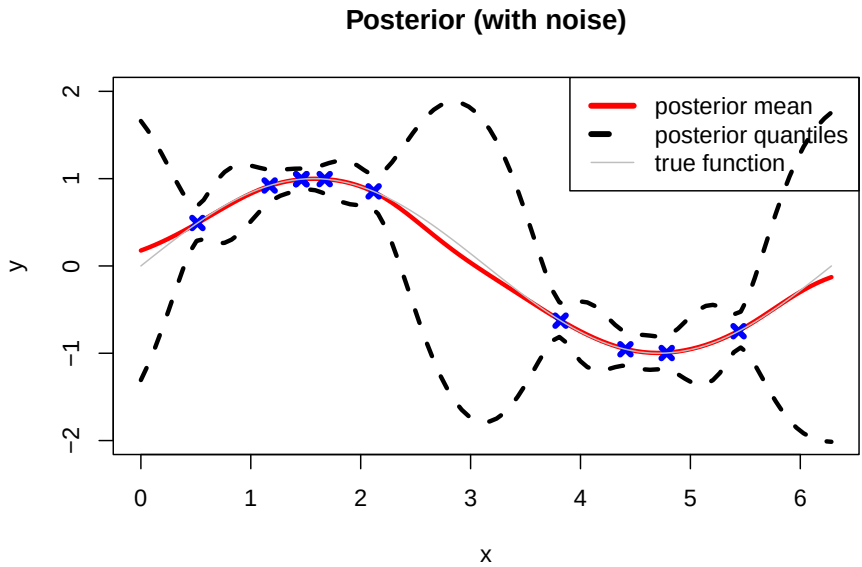
```
# upper and lower CI
upper.bound = m1.2 + 1.96*sqrt(diag(K1.2))
lower.bound = m1.2 - 1.96*sqrt(diag(K1.2))

plot(x1, m1.2, type="l", xlim=XLIM, ylim=YLIM, col="red", lwd=3,
  ylab="y", xlab = "x", main = "Posterior")

points(x2,y2,pch=4,lwd=4,col="blue")
lines(x1,upper.bound,lty=2,lwd=3)
lines(x1,lower.bound,lty=2,lwd=3)
curve(truefunc(x), xlim=XLIM, add=TRUE, col="gray")

legend(x="topright",
  legend=c("posterior mean", "posterior quantiles", "true function"),
  lty=c(1, 2, 1),lwd=c(4, 4, 1), col=c("red","black", "gray"), cex=1.0)
```



Finally, we can take into acount noise at the measured data points by adding an error term:

```
# add some noise
sdeps = 0.1
K2 = K2 + sdeps^2*diag(1,length(x2))

# update
```

```
iK2 = solve(K2) # inverse
m1.2 = m1 + K12 %*% iK2 %*% (y2 - m2)
K1.2 = K1 - K12 %*% iK2 %*% t(K12)
upper.bound = m1.2 + 1.96*sqrt(diag(K1.2))
lower.bound = m1.2 - 1.96*sqrt(diag(K1.2))

plot(x1, m1.2, type="l", xlim=XLIM, ylim=YLIM, col="red", lwd=3,
  ylab="y", xlab = "x", main = "Posterior (with noise)")

points(x2,y2,pch=4,lwd=4,col="blue")
lines(x1,upper.bound,lty=2,lwd=3)
lines(x1,lower.bound,lty=2,lwd=3)
curve(truefunc(x), xlim=XLIM, add=TRUE, col="gray")

legend(x="topright",
  legend=c("posterior mean", "posterior quantiles", "true function"),
  lty=c(1, 2, 1),lwd=c(4, 4, 1), col=c("red","black", "gray"), cex=1.0)
```



Note that in the vicinity of data points the CIs are small and the further away from data the more uncertain the estimate of the underlying function becomes.

6.3 Neural networks

Another highly important class of models for nonlinear prediction (and nonlinear function approximation) are neural networks.

Relevant reading:

Please read: Hastie, Tibshirani, and Friedman (2009) **Chapter 11 "Neural networks"** and James et al. (2021) **Chapter 10 "Deep Learning"**

6.3.1 History

Neural networks are actually relatively old models, going back to the 1950s.

Three phases of neural networks (NN)

- 1950/60: replicating functions of neurons in the brain (e.g. perceptron)
- 1980/90: neural networks as universal function approximators
- 2010—today: deep learning

The first phase was biologically inspired, the second phase focused on mathematical properties, and the current phase is pushed forward by advances in computer science and numerical optimisation:

- backpropagation algorithm
- efficient automatic symbolic differentiation
- stochastic gradient descent algorithms (e.g. Adam)
- use of GPUs and TPUs (e.g. for linear algebra)
- availability and development of deep learning packages:
  - Aesara (PyMC), formely Theano (University of Montreal)
  - PyTorch (PyTorch Foundation, formerly Meta/Facebook)
  - Flax / JAX (Google Research)
  - TensorFlow (Google Research)
  - MXNet (Amazon)
  - PaddlePaddle (Baidu)

and high-level wrappers:

- PyTorch-Lightning (for PyTorch)
- Keras (for Tensorflow, MXNet, Theano)

6.3.2 Neural networks

Neural networks are essentially stacked systems of linear regressions, with nonlinear mappings between each layer, mapping the input to output via one or more layers of internal hidden nodes corresponding to internal latent variables:

- Each internal node is a nonlinear function of all or some of nodes in the previous layer
- Typically, the output of a node is computed using a **non-linear activation function**, such as the sigmoid function or a piecewise linear function (ReLU), from a linear combination of the input variables of that node.

A simple architecture is a feedforward network with a single hidden layer. More complex models are multilayer perceptrons and convolutional neural networks.

It can be shown that even simple network architectures can (with sufficient number of nodes) approximate any arbitrary non-linear function. This is called the "universal function approximation" property.

"Deep" neural networks have many layers, and the optimisation of their parameters requires advanced techniques (see above), with the objective function typically an empirical risk based on, e.g., squared error loss or cross-entropy loss. Neural networks are very highly parameterised models and therefore require lots of data for training, and typically also some form of regularisation (e.g. dropout).

In the limit of an infinite width a single layer fully connected neural network becomes equivalent to a Gaussian process. This was first shown by Radford Neal in 1996. More recently, this equivalence has also been demonstrated for other types of neural networks (with the kernel function of the GP being determined by the neural network architecture). This is formalised in the "neural tangent kernel" (NTK) framework.

Some of the statistical aspects of neural networks are not well understood. For example, there is the paradox that neural networks typically overfit the training data but still generalise well - this clearly violates the traditional understanding of bias-variance tradeoff for classical modelling in statistics and machine learning. Some researchers argue that this contradiction can be resolved by better understanding the effective dimension of complex models. There is a lot of current research to explain this phenomenon of "multiple descent", i.e. the decrease of prediction error for models with very many parameters. A further topic is robustness of the predictions, which is also caused by overfitting. It is well known that neural networks can sometimes be "fooled" by so-called adversarial examples, e.g., the classification of a sample may change if a small amount of noise is added to the test data.

6.3.3 Learning more about deep learning

A good place to to learn more about deep learning and actual application in computer code using various deep learning software frameworks is the online book "Dive into deep learning" by Zhang et al. (2022) available online at https://d2l.ai/

Appendix A

Brief refresher on matrices

In multivariate statistics we will frequently make use of matrix calculations and matrix notation. This helps to make multivariate equations simpler and enables a better understanding of the underlying concepts.

Throughout the module we mostly work with **real matrices**, i.e. we assume all matrix elements are real numbers. However, one important matrix decomposition — the eigenvalue decomposition — can yield complex-valued matrices even when applied to real matrices. Thus occasionally we will need to deal also with complex numbers.

For further details on matrix theory please consult the lecture notes of related modules (e.g. linear algebra).

A.1 Matrix basics

A.1.1 Matrix notation

In matrix notation we distinguish between scalars, vectors, and matrices:

Scalar: x, X, lower or upper case, plain type.

Vector: $\boldsymbol{x}$, lower case, bold type. In handwriting an arrow $\vec{x}$ indicates a vector.

In component notation we write $\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix}$. By default, a vector is a column vector, i.e. the elements are arranged in a column and index of the components x_i refers to the row.

The **transpose** of a vector (indicated by the superscript T) turns it into a row vector. To save space we can write the column vector $\boldsymbol{x}$ as $\boldsymbol{x} = (x_1, \ldots, x_d)^T$ so

that $\boldsymbol{x}^T$ is a row vector.

Matrix: $\boldsymbol{X}$, upper case, bold type. In handwriting an underscore $\underline{X}$ indicates a matrix.

In component notation we write $\boldsymbol{X} = (x_{ij})$. By convention, the first index (here i) of the scalar elements x_{ij} denotes the row and the second index (here j) the column of the matrix. For n the number of rows and d the number of columns we can view the matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d)$ either as being composed of d column vectors $\boldsymbol{x}_j = \begin{pmatrix} x_{1j} \\ \vdots \\ x_{nj} \end{pmatrix}$ or $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{z}_1^T \\ \vdots \\ \boldsymbol{z}_n^T \end{pmatrix}$ being composed of n row vectors $\boldsymbol{z}_i^T = (x_{i1}, \ldots, x_{id})$.

A (column) vector of dimension d is a matrix of size $d \times 1$. A row vector of dimension d is a matrix of size $1 \times d$. A scalar is of dimension 1 and is a matrix of size 1×1.

A.1.2 Random matrix

A **random matrix** (vector) is a matrix (vector) whose elements are random variables.

Note that the standard notation used in univariate statistics to distinguish random variables and their realisations (i.e. upper versus lower case) does not work in multivariate statistics. Therefore, you need to determine from the context whether a quantity represents a random variable, or whether it is a constant.

A.1.3 Special matrices

$\boldsymbol{I}_d$ is the identity matrix. It is a square matrix of size $d \times d$ with the diagonal filled with 1 and off-diagonals filled with 0.

$$\boldsymbol{I}_d = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & 0 & & 1 \end{pmatrix}$$

1 is a matrix that contains only ones. Most often it is used in the form of a column vector with d rows:

$$\mathbf{1}_d = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

Similarly, $\mathbf{0}$ is a matrix that contains only zeros. Most often it is used in the form of a column vector with d rows:

$$\mathbf{0}_d = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

A diagonal matrix is a matrix where all off-diagonal elements are zero. By $\text{Diag}(\boldsymbol{A})$ we access the diagonal elements of a matrix as vector and by $\text{Diag}(a_1, \ldots, a_d)$ we specify a diagonal matrix by listing the diagonal elements.

A triangular matrix is a square matrix whose elements either below or above the diagonal are all zero (upper vs. lower triangular matrix).

A.2 Simple matrix operations

A.2.1 Matrix addition and multiplication

Matrices behave much like common numbers. For example, we can add matrices $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$ and multiply matrices $\boldsymbol{C} = \boldsymbol{AB}$.

For **matrix addition** $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$ we add the corresponding elements $c_{ij} = a_{ij} + b_{ij}$. For matrix addition $\boldsymbol{A}$ and $\boldsymbol{B}$ must have the same dimensions, i.e. the same number of rows and columns.

The **dot product**, or **scalar product**, of two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is a scalar given by $\boldsymbol{a} \cdot \boldsymbol{b} = \langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^T \boldsymbol{b} = \boldsymbol{b}^T \boldsymbol{a} = \sum_{i=1}^{d} a_i b_i$.

Matrix multiplication $\boldsymbol{C} = \boldsymbol{AB}$ is obtained by setting $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$ where m is the number of columns of $\boldsymbol{A}$ and the number of rows in $\boldsymbol{B}$. Thus, $\boldsymbol{C}$ contains all possible dot products of the row vectors in $\boldsymbol{A}$ with the column vectors in $\boldsymbol{B}$. For matrix multiplication the number of columns in $\boldsymbol{A}$ must match the number of rows in $\boldsymbol{B}$. Note that matrix multiplication in general (for $m > 1$) does not commute, i.e. $\boldsymbol{AB} \neq \boldsymbol{BA}$.

A.2.2 Matrix transpose

The matrix transpose $\boldsymbol{A}^T$ indicate by the superscript T interchanges rows and columns of a matrix. The transpose is a linear operator $(\boldsymbol{A} + \boldsymbol{B})^T = \boldsymbol{A}^T + \boldsymbol{B}^T$ and applied to a matrix product it reverses the ordering, i.e. $(\boldsymbol{AB})^T = \boldsymbol{B}^T \boldsymbol{A}^T$.

If $\boldsymbol{A} = \boldsymbol{A}^T$ then $\boldsymbol{A}$ is symmetric (and square).

By construction given a rectangular $\boldsymbol{A}$ the matrices $\boldsymbol{A}^T \boldsymbol{A}$ and $\boldsymbol{A}\boldsymbol{A}^T$ are symmetric with non-negative diagonal.

A.3 Matrix summaries

A.3.1 Row, column and grand sum

Assume a matrix $\boldsymbol{A}$ of size $n \times m$.

The sum over the m entries of row i is $\sum_{j=1}^{m} a_{ij}$. In matrix notation the n row sums are given by $\boldsymbol{A}\,\mathbf{1}_m$.

The sum over the n entries of column j is $\sum_{i=1}^{n} a_{ij}$. In matrix notation the m column sums are $\boldsymbol{A}^T\mathbf{1}_n$.

The grand sum of all matrix entries of $\boldsymbol{A}$ is obtained by

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij} = \mathbf{1}_n^T\,\boldsymbol{A}\,\mathbf{1}_m$$

A.3.2 Matrix trace

The trace of the matrix is the sum of the diagonal elements $\mathrm{Tr}(\boldsymbol{A}) = \sum a_{ii}$.

The trace is invariant against transposition, i.e.

$$\mathrm{Tr}(\boldsymbol{A}) = \mathrm{Tr}(\boldsymbol{A}^T)$$

A useful identity for the matrix trace of the product of two matrices is

$$\mathrm{Tr}(\boldsymbol{AB}) = \mathrm{Tr}(\boldsymbol{BA})$$

Intriguingly, the trace of a matrix equals the sum of the eigenvalues of the matrix (see further below).

A.3.3 Row, column and grand sum of squares

The sum over the m squared entries of row i is $\sum_{j=1}^{m} a_{ij}^2$. In matrix notation the n row sums of squares are given by $\mathrm{Diag}(\boldsymbol{AA}^T)$.

The sum over the n squared entries of column j is $\sum_{i=1}^{n} a_{ij}^2$. In matrix notation the m column sums of squares are $\mathrm{Diag}(\boldsymbol{A}^T\boldsymbol{A})$.

The grand sum of all squared elements of $\boldsymbol{A}$ is obtained by

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}^2 = \mathrm{Tr}(\boldsymbol{A}^T\boldsymbol{A}) = \mathrm{Tr}(\boldsymbol{AA}^T)$$

This is also known as the squared Frobenius norm of $\boldsymbol{A}$ (see below).

A.3.4 Sum of squared diagonal entries

The sum of the squared entries on the diagonal is in matrix notation

$$\text{Diag}(\boldsymbol{A})^T \text{Diag}(\boldsymbol{A}) = \sum_{i=1}^{\min(n,m)} a_{ii}^2$$

A.3.5 Frobenius inner product

The **Frobenius inner product** between two rectangular matrices of the same dimension is the scalar

$$\begin{aligned}\langle \boldsymbol{A}, \boldsymbol{B} \rangle &= \text{Tr}(\boldsymbol{A}\boldsymbol{B}^T) = \text{Tr}(\boldsymbol{B}\boldsymbol{A}^T) \\ &= \text{Tr}(\boldsymbol{A}^T\boldsymbol{B}) = \text{Tr}(\boldsymbol{B}^T\boldsymbol{A}) \\ &= \sum_{i,j} a_{ij} b_{ij} \,.\end{aligned}$$

This generalises the dot product between two vectors. Note that the dot product can therefore also be written as the trace of a matrix

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \text{Tr}(\boldsymbol{a}\boldsymbol{b}^T) = \text{Tr}(\boldsymbol{b}\boldsymbol{a}^T) \,.$$

A.3.6 Euclidean norm

The **squared Euclidean norm** or the **squared length** of the vector $\boldsymbol{a}$ is the dot product $||\boldsymbol{a}||_2^2 = \boldsymbol{a} \cdot \boldsymbol{a} = \langle \boldsymbol{a}, \boldsymbol{a} \rangle = \boldsymbol{a}^T\boldsymbol{a} = \boldsymbol{a}\boldsymbol{a}^T = \sum_{i=1}^d a_i^2$.

The **squared Frobenius norm** is a generalisation of the squared Euclidean vector norm to a rectangular matrix and is the sum of the squares of all its elements. Using the trace it can be written as

$$\begin{aligned}||\boldsymbol{A}||_F^2 &= \langle \boldsymbol{A}, \boldsymbol{A} \rangle \\ &= \text{Tr}(\boldsymbol{A}^T\boldsymbol{A}) = \text{Tr}(\boldsymbol{A}\boldsymbol{A}^T) \\ &= \sum_{i,j} a_{ij}^2 \,.\end{aligned}$$

A useful identity for the **squared Frobenius norm of the difference of two matrices** is

$$\begin{aligned}||\boldsymbol{A} - \boldsymbol{B}||_F^2 &= ||\boldsymbol{A}||_F^2 + ||\boldsymbol{B}||_F^2 - 2\langle \boldsymbol{A}, \boldsymbol{B} \rangle \\ &= \text{Tr}(\boldsymbol{A}^T\boldsymbol{A}) + \text{Tr}(\boldsymbol{B}^T\boldsymbol{B}) - 2\text{Tr}(\boldsymbol{A}^T\boldsymbol{B}) \\ &= \sum_{i,j} (a_{ij} - b_{ij})^2 \,.\end{aligned}$$

The Frobenius norm of a matrix $||\boldsymbol{A}||_F$ is not to be confused with the induced 2-norm of a matrix $||\boldsymbol{A}||_2$. The latter equals the maximum absolute eigenvalue of the matrix, with $||\boldsymbol{A}||_2 \leq ||\boldsymbol{A}||_F$.

A.3.7 Determinant of a matrix

If $\boldsymbol{A}$ is a square matrix the determinant $\det(\boldsymbol{A})$ is a scalar measuring the volume spanned by the column vectors in $\boldsymbol{A}$ with the sign determined by the orientation of the vectors.

If $\det(\boldsymbol{A}) \neq 0$ the matrix $\boldsymbol{A}$ is non-singular or non-degenerate. Conversely, if $\det(\boldsymbol{A}) = 0$ the matrix $\boldsymbol{A}$ is singular or degenerate.

Intriguingly, the determinant of $\boldsymbol{A}$ is the product of the eigenvalues of $\boldsymbol{A}$ (see further below).

One way to compute the determinant of a matrix $\boldsymbol{A}$ is the Laplace cofactor expansion approach that proceeds recursively based on the determinants of the submatrices $\boldsymbol{A}_{-i,-j}$ obtained by deleting row i and column j from $\boldsymbol{A}$. Specifically, at each level we compute the

1) cofactor expansion either
   a) along the i-th row — pick any row i:

$$\det(\boldsymbol{A}) = \sum_{j=1}^{d} a_{ij}(-1)^{i+j} \det(\boldsymbol{A}_{-i,-j}) \text{ , or}$$

   b) along the j-th column — pick any j:

$$\det(\boldsymbol{A}) = \sum_{i=1}^{d} a_{ij}(-1)^{i+j} \det(\boldsymbol{A}_{-i,-j})$$

.

2) Then repeat until the submatrix is a scalar a and $\det(a) = a$.

The recursive nature of this algorithm leads to a complexity of order $O(d!)$ so it is not practical except for very small d. Therefore, in practice other more efficient algorithms for computing determinants are used but these still have algorithmic complexity in the order of $O(d^3)$ so for large dimensions obtaining determinants is very expensive.

However, some specially structured matrices do allow for very fast calculation.

The determinant of a **triangular matrix** (and thus also of a **diagonal matrix**)

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ a_{d1} & a_{d2} & \cdots & a_{dd} \end{pmatrix}$$

is the product of its diagonal elements, i.e. $\det(\boldsymbol{A}) = \prod_{i=1}^{d} a_{ii}$.

For a two-dimensional matrix $\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ the determinant is $\det(A) = a_{11}a_{22} - a_{12}a_{21}$.

For a block-structured square matrix

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix},$$

where the matrices on the diagonal $\boldsymbol{A}_{11}$ and $\boldsymbol{A}_{22}$ are themselves square but $\boldsymbol{A}_{21}$ and $\boldsymbol{A}_{21}$ can have any shape, the determinant is

$$\det(\boldsymbol{A}) = \det(\boldsymbol{A}_{22}) \det(\boldsymbol{C}_1) = \det(\boldsymbol{A}_{11}) \det(\boldsymbol{C}_2)$$

with the (Schur complement of $\boldsymbol{A}_{22}$)

$$\boldsymbol{C}_1 = \boldsymbol{A}_{11} - \boldsymbol{A}_{12}\boldsymbol{A}_{22}^{-1}\boldsymbol{A}_{21}$$

and (Schur complement of $\boldsymbol{A}_{11}$)

$$\boldsymbol{C}_2 = \boldsymbol{A}_{22} - \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12}$$

Note that $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are square matrices.

For a block-diagonal matrix $\boldsymbol{A}$ with $\boldsymbol{A}_{12} = 0$ and $\boldsymbol{A}_{21} = 0$ the determinant is $\det(\boldsymbol{A}) = \det(\boldsymbol{A}_{11}) \det(\boldsymbol{A}_{22})$, i.e. the product of the determinants of the submatrices along the diagonal.

Determinants have a multiplicative property,

$$\det(\boldsymbol{AB}) = \det(\boldsymbol{BA}) = \det(\boldsymbol{A}) \det(\boldsymbol{B}).$$

In the above $\boldsymbol{A}$ and $\boldsymbol{B}$ are both square and of the same dimension.

For rectangular $\boldsymbol{A}$ $(n \times m)$ and rectangular $\boldsymbol{B}$ $(m \times n)$ with $m \geq n$ this generalises to the Cauchy-Binet formula

$$\det(\boldsymbol{AB}) = \sum_{w} \det(\boldsymbol{A}_{,w}) \det(\boldsymbol{B}_{w,})$$

where the summation is over all $\binom{m}{n}$ index subsets w of size n taken from $\{1, \ldots, m\}$ keeping the ordering and $\boldsymbol{A}_{,w}$ and $\boldsymbol{B}_{w,}$ are the corresponding square $n \times n$ submatrices. If $m < n$ then $\det(\boldsymbol{AB}) = 0$.

For scalar a $\det(a\boldsymbol{B}) = a^d \det(\boldsymbol{B})$ where d is the dimension of $\boldsymbol{B}$.

Another important identity is

$$\det(\boldsymbol{I}_n + \boldsymbol{AB}) = \det(\boldsymbol{I}_m + \boldsymbol{BA})$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are rectangular matrices. This is called the Weinstein-Aronszajn determinant identity (also credited to Sylvester).

A.4 Matrix inverse

A.4.1 Inversion of square matrix

If $\boldsymbol{A}$ is a square matrix then the inverse matrix $\boldsymbol{A}^{-1}$ is a matrix such that

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}\,.$$

Only non-singular matrices with $\det(\boldsymbol{A}) \neq 0$ are invertible.

As $\det(\boldsymbol{A}^{-1}\boldsymbol{A}) = \det(\boldsymbol{I}) = 1$ the determinant of the inverse matrix equals the inverse determinant,

$$\det(\boldsymbol{A}^{-1}) = \det(\boldsymbol{A})^{-1}\,.$$

The transpose of the inverse is the inverse of the transpose as

$$\begin{aligned}(\boldsymbol{A}^{-1})^T &= (\boldsymbol{A}^{-1})^T\,\boldsymbol{A}^T(\boldsymbol{A}^T)^{-1}\\ &= (\boldsymbol{A}\boldsymbol{A}^{-1})^T\,(\boldsymbol{A}^T)^{-1} = (\boldsymbol{A}^T)^{-1}\,.\end{aligned}$$

The inverse of a matrix product $(\boldsymbol{A}\boldsymbol{B})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$ is the product of the indivdual matrix inverses in reverse order.

There are many different algorithms to compute the inverse of a matrix (which is essentially a problem of solving a system of equations). The computational complexity of matrix inversion is of the order $O(d^3)$ where d is the dimension of $\boldsymbol{A}$. Therefore matrix inversion is very costly in higher dimensions.

Example A.1. Inversion of a 2×2 matrix:

The inverse of the matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is $A^{-1} = \frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$

A.4.2 Inversion of structured matrices

However, for specially structured matrices inversion can be done effectively:

- The inverse of a **diagonal matrix** is another diagonal matrix obtained by inverting the diagonal elements.
- More generally, the inverse of a **block-diagonal matrix** is obtained by individually inverting the blocks along the diagonal.

The **Woodbury matrix identity** simplifies the inversion of matrices that can be written as $\boldsymbol{A} + \boldsymbol{U}\boldsymbol{B}\boldsymbol{V}$ where $\boldsymbol{A}$ and $\boldsymbol{B}$ are both square and $\boldsymbol{U}$ and $\boldsymbol{V}$ are suitable rectangular matrices:

$$(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{B}\boldsymbol{V})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{B}^{-1} + \boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1}$$

Typically, the inverse $\boldsymbol{A}^{-1}$ is either already known or can be easily obtained and the dimension of $\boldsymbol{B}$ is much lower than that of $\boldsymbol{A}$.

The class of matrices that can be most easily inverted are **orthogonal matrices** whose inverse is obtained simply by transposing the matrix.

A.5 Orthogonal matrices

A.5.1 Properties

An orthogonal matrix $\boldsymbol{Q}$ is a square matrix with the property that $\boldsymbol{Q}^T = \boldsymbol{Q}^{-1}$, i.e. the transpose is also the inverse. This implies that $\boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$.

Both the column and the row vectors in $\boldsymbol{Q}$ all have length 1. This implies that each element q_{ij} of $\boldsymbol{Q}$ can only take a value in the interval $[-1, 1]$.

The identity matrix $\boldsymbol{I}$ is the simplest example of an orthogonal matrix.

The squared Euclidean and Frobenius norm is preserved when a vector $\boldsymbol{a}$ or matrix $\boldsymbol{A}$ is multiplied with an orthogonal matrix $\boldsymbol{Q}$:

$$||\boldsymbol{Q}\boldsymbol{a}||_2^2 = (\boldsymbol{Q}\boldsymbol{a})^T\boldsymbol{Q}\boldsymbol{a} = \boldsymbol{a}^T\boldsymbol{a} = ||\boldsymbol{a}||_2^2$$

and

$$||\boldsymbol{Q}\boldsymbol{A}||_F^2 = \mathrm{Tr}\left((\boldsymbol{Q}\boldsymbol{A})^T\boldsymbol{Q}\boldsymbol{A}\right) = \mathrm{Tr}\left(\boldsymbol{A}^T\boldsymbol{A}\right) = ||\boldsymbol{A}||_F^2$$

Multiplication of $\boldsymbol{Q}$ with a vector results in a new vector of the same length but with a change in direction (unless $\boldsymbol{Q} = \boldsymbol{I}$). An orthogonal matrix $\boldsymbol{Q}$ can thus be interpreted geometrically as an operator performing rotation, reflection and/or permutation.

The product $\boldsymbol{Q}_3 = \boldsymbol{Q}_1\boldsymbol{Q}_2$ of two orthogonal matrices $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ yields another orthogonal matrix as $\boldsymbol{Q}_3\boldsymbol{Q}_3^T = \boldsymbol{Q}_1\boldsymbol{Q}_2(\boldsymbol{Q}_1\boldsymbol{Q}_2)^T = \boldsymbol{Q}_1\boldsymbol{Q}_2\boldsymbol{Q}_2^T\boldsymbol{Q}_1^T = \boldsymbol{I}$.

The determinant $\det(\boldsymbol{Q})$ of an orthogonal matrix is either +1 or -1, because $\boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{I}$ and thus $\det(\boldsymbol{Q})\det(\boldsymbol{Q}^T) = \det(\boldsymbol{Q})^2 = \det(\boldsymbol{I}) = 1$.

The set of all orthogonal matrices of dimension d together with multiplication form a group called the orthogonal group $O(d)$. The subset of orthogonal matrices with $\det(\boldsymbol{Q}) = 1$ are called rotation matrices and form with multiplication the special orthogonal group $SO(d)$. Orthogonal matrices with $\det(\boldsymbol{Q}) = -1$ are rotation-reflection matrices.

A.5.2 Semi-orthogonal matrices

A rectangular $d \times k$ matrix $\boldsymbol{Q}$ is semi-orthogonal if for $k < d$ the k column vectors are orthonormal and hence $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_k$, or if for $k > d$ the d row vectors are orthonormal with $\boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{I}_d$.

The set of all (semi)-orthogonal matrices $\boldsymbol{Q}$ with $k \leq d$ column vectors is known as the Stiefel manifold $\mathrm{St}(d, k)$.

A.5.3 Generating orthogonal matrices

In two dimensions ($d = 2$) all orthogonal matrices $\boldsymbol{R}$ representing rotations with $\det(\boldsymbol{R}) = 1$ are given by

$$\boldsymbol{R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

and those representing rotation-reflections $\boldsymbol{G}$ with $\det(\boldsymbol{G}) = -1$ by

$$\boldsymbol{G}(\theta) = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}.$$

Every orthogonal matrix of dimension $d = 2$ can be represented as the product of at most two rotation-reflection matrices because

$$\boldsymbol{R}(\theta) = \boldsymbol{G}(\theta)\,\boldsymbol{G}(0) = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

Thus, the matrix $\boldsymbol{G}$ is a generator of two-dimensional orthogonal matrices. Note that $\boldsymbol{G}(\theta)$ is symmetric, orthogonal and has determinant -1.

More generally, and applicable in arbitrary dimension, the role of generator is taken by the Householder reflection matrix

$$\boldsymbol{Q}_{HH}(\boldsymbol{v}) = \boldsymbol{I} - 2\boldsymbol{v}\boldsymbol{v}^T$$

where $\boldsymbol{v}$ is a vector of unit length (with $\boldsymbol{v}^T\boldsymbol{v} = 1$) orthogonal to the reflection hyperplane. Note that $\boldsymbol{Q}_{HH}(\boldsymbol{v}) = \boldsymbol{Q}_{HH}(-\boldsymbol{v})$. By construction the matrix $\boldsymbol{Q}_{HH}(\boldsymbol{v})$ is symmetric, orthogonal and has determinant -1.

It can be shown that any d-dimensional orthogonal matrix $\boldsymbol{Q}$ can be represented as the product of at most d Householder reflection matrices. The two-dimensional generator $\boldsymbol{G}(\theta)$ is recovered as the Householder matrix $\boldsymbol{Q}_{HH}(\boldsymbol{v})$ with $\boldsymbol{v} = \begin{pmatrix} -\sin\frac{\theta}{2} \\ \cos\frac{\theta}{2} \end{pmatrix}$ or $\boldsymbol{v} = \begin{pmatrix} \sin\frac{\theta}{2} \\ -\cos\frac{\theta}{2} \end{pmatrix}$.

A.5.4 Permutation matrix

A special type of an orthogonal matrix is a permutation matrix $\boldsymbol{P}$ created by permuting rows and/or columns of the identity matrix $\boldsymbol{I}$. Thus, each row and column of $\boldsymbol{P}$ contains exactly one entry of 1, but not necessarily on the diagonal.

If a permutation matrix $\boldsymbol{P}$ is multiplied with a matrix $\boldsymbol{A}$ it acts as an operator permuting the columns ($\boldsymbol{AP}$) or the rows ($\boldsymbol{PA}$). For a set of d elements there exist $d!$ permutations. Thus, for dimension d there are $d!$ possible permutation matrices (including the identity matrix).

The determinant of a permutation matrix is either +1 or -1. The product of two permutation matrices yields another permutation matrix.

Symmetric permutation matrices correspond to self-inverse permutations (i.e. the permutation matrix is its own inverse), and are also called permutation involutions. They can have determinant +1 and -1.

A transposition is a permutation where only two elements are exchanged. Thus, in a transposition matrix $\boldsymbol{T}$ exactly two rows and/or columns are exchanged compared to identity matrix $\boldsymbol{I}$. Transpositions are self-inverse, and transposition matrices are symmetric. There are $\frac{d(d-1)}{2}$ different transposition matrices. The determinant of a transposition matrix is $\det(\boldsymbol{T}) = -1$.

Note that the transposition matrix is an instance of a Householder matrix $\boldsymbol{Q}_{HH}(v)$ with vector v filled with zeros except for two elements that have value $\frac{\sqrt{2}}{2}$ and $-\frac{\sqrt{2}}{2}$.

Any permutation of d elements can be generated by a series of at most $d-1$ transpositions. Correspondingly, any permutation matrix $\boldsymbol{P}$ can be constructed by multiplication of the identity matrix with at most $d-1$ transposition matrices. If the number of transpositions is even then $\det(\boldsymbol{P}) = 1$ otherwise for an uneven number $\det(\boldsymbol{P}) = -1$. This is called the sign or signature of the permutation.

The set of all permutations form the symmetric group S_d, the subset of even permutations (with positive sign and $\det(\boldsymbol{P}) = 1$) the alternating group A_d.

A.6 Eigenvalues and eigenvectors

A.6.1 Definition

Assume a square matrix $\boldsymbol{A}$ of size $d \times d$. A vector $\boldsymbol{u} \neq 0$ is called an eigenvector of the matrix $\boldsymbol{A}$ and λ the corresponding eigenvalue if

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{u}\lambda\,.$$

This is called **eigenvalue equation** or **eigenequation**.

A.6.2 Finding eigenvalues and vectors

To find the eigenvalues and eigenvectors the eigenequation is rewritten as

$$(\boldsymbol{A} - \boldsymbol{I}\lambda)\,\boldsymbol{u} = \boldsymbol{0}\,.$$

For this equation to hold for an eigenvector $\boldsymbol{u} \neq 0$ with eigenvalue λ implies that the matrix $\boldsymbol{A} - \boldsymbol{I}\lambda$ is singular. Correspondingly, its determinant must vanish

$$\det(\boldsymbol{A} - \boldsymbol{I}\lambda) = 0\,.$$

This is called the *characteristic equation* of the matrix $\boldsymbol{A}$, and its solution yields the d eigenvalues $\lambda_1, \ldots, \lambda_d$. Note the eigenvalues need not be distinct and they may be complex even if the matrix $\boldsymbol{A}$ is real.

If there are complex eigenvalues, for a real matrix those eigenvalues come in conjugate pairs. Hence, for a complex $\lambda_1 = re^{i\phi}$ there will also be a corresponding complex eigenvalue $\lambda_2 = re^{-i\phi}$.

Given the eigenvalues we then solve the eigenequation for the corresponding non-zero eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_d$. Note that eigenvectors of real matrices can have complex components. Also the eigenvector is only defined by the eigenequation up to a scalar. By convention eigenvectors are therefore typically standardised to unit length but this still leaves a sign ambiguity for real eigenvectors and implies that complex eigenvectors are defined only up to a factor with modulus 1.

A.6.3 Eigenequation in matrix notation

With the matrix

$$\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_d)$$

containing the standardised eigenvectors in the columns and the diagonal matrix

$$\boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \lambda_d \end{pmatrix}$$

containing the eigenvalues (typically sorted in order of magnitude) the eigenvalue equation can be written as

$$\boldsymbol{A}\boldsymbol{U} = \boldsymbol{U}\boldsymbol{\Lambda}\,.$$

A.6.4 Permutation of eigenvalues

If eigenvalues are not in order, we may apply a permutation matrix $\boldsymbol{P}$ to arrange them in order. With $\boldsymbol{\Lambda}^{\text{sort}} = \boldsymbol{P}^T\boldsymbol{\Lambda}\boldsymbol{P}$ as the sorted eigenvalues and $\boldsymbol{U}^{\text{sort}} = \boldsymbol{U}\boldsymbol{P}$ as the corresponding eigenvectors the eigenequation becomes

$$\boldsymbol{A}\boldsymbol{U}^{\text{sort}} = \boldsymbol{A}\boldsymbol{U}\boldsymbol{P} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{P} = \boldsymbol{U}\boldsymbol{P}\boldsymbol{P}^T\boldsymbol{\Lambda}\boldsymbol{P} = \boldsymbol{U}^{\text{sort}}\boldsymbol{\Lambda}^{\text{sort}}\,.$$

A.6.5 Similar matrices

Two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are called **similar** if they share the same eigenvalues.

From $\boldsymbol{A}$ with eigenvalues $\boldsymbol{\Lambda}$ and eigenvectors $\boldsymbol{U}$ we can construct a similar $\boldsymbol{B}$ via the **similarity transformation** $\boldsymbol{B} = \boldsymbol{M}\boldsymbol{A}\boldsymbol{M}^{-1}$ where $\boldsymbol{M}$ is an invertible matrix.

Then $\boldsymbol{\Lambda}$ are the eigenvalues of $\boldsymbol{B}$ and $\boldsymbol{V} = \boldsymbol{M}\boldsymbol{U}$ its eigenvectors as

$$\boldsymbol{B}\boldsymbol{V} = \boldsymbol{M}\boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{M}\boldsymbol{U} = \boldsymbol{M}\boldsymbol{A}\boldsymbol{U} = \boldsymbol{M}\boldsymbol{U}\boldsymbol{\Lambda} = \boldsymbol{V}\boldsymbol{\Lambda}\,.$$

A.6.6 Defective matrix

In most cases the eigenvectors $\boldsymbol{u}_i$ will be linearly independent so that they form a basis to span a d dimensional space.

However, if this is not the case and the matrix $\boldsymbol{A}$ does not have a complete basis of eigenvectors, then the matrix is called defective. In this case the matrix $\boldsymbol{U}$ containing the eigenvectors is singular and $\det(\boldsymbol{U}) = 0$.

An example of a defective matrix is $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ which has determinant 1 so that it can be inverted and its column vectors do form a complete basis but has only one distinct eigenvector $(1, 0)^T$ so that the eigenvector basis is incomplete.

A.6.7 Eigenvalues of a diagonal or triangular matrix

In the special case that $\boldsymbol{A}$ is diagonal or a triangular matrix the eigenvalues are easily determined. This follows from the simple form of their determinants as the product of the diagonal elements. Hence for these matrices the characteristic equation becomes $\prod_i^d (a_{ii} - \lambda) = 0$ and has solution $\lambda_i = a_{ii}$, i.e. the eigenvalues are equal to the diagonal elements.

A.6.8 Eigenvalues and vectors of a symmetric matrix

If $\boldsymbol{A}$ is symmetric, i.e. $\boldsymbol{A} = \boldsymbol{A}^T$, then its eigenvalues and eigenvectors have special properties:

i) all eigenvalues of $\boldsymbol{A}$ are real,
ii) the eigenvectors are orthogonal, i.e $\boldsymbol{u}_i^T \boldsymbol{u}_j = 0$ for $i \neq j$, and real. Thus, the matrix $\boldsymbol{U}$ containing the standardised orthonormal eigenvectors is orthogonal.
iii) $\boldsymbol{A}$ is never defective as $\boldsymbol{U}$ forms a complete basis.

Furthermore, for a symmetric matrix $\boldsymbol{A}$ with diagonal elements $p_1 \geq \ldots \geq p_d$ and eigenvalues $\lambda_1 \geq \ldots \geq \lambda_d$ (note both written in decreasing order) the sum of the largest k eigenvalues forms an upper bound of the sum of the largest k diagonal elements:

$$\sum_i^k \lambda_i \geq \sum_i^k p_i$$

This theorem is due to Schur (1923).[1] The equality holds for $k = d$ (as the trace of $\boldsymbol{A}$ equals the sum of its eigenvalues) and for any k if $\boldsymbol{A}$ is diagonal (as in this case of the diagonal elements equal the eigenvalues).

[1] Schur, I. 1923. Über eine Klasse von Mittelbildungen mit Anwendungen auf die Determinantentheorie. Sitzungsber. Berl. Math. Ges. **22**:9–29.

A.6.9 Eigenvalues of orthogonal matrices

The eigenvalues of an orthogonal matrix $\boldsymbol{Q}$ are not necessarily real but they all have modulus 1 and lie on the unit circle . Thus, the eigenvalues of $\boldsymbol{Q}$ all have the form $\lambda = e^{i\phi} = \cos\phi + i\sin\phi$.

In any real matrix complex eigenvalues come in conjugate pairs. Hence if an orthogonal matrix $\boldsymbol{Q}$ has the complex eigenvalue $e^{i\phi}$ it also has an complex eigenvalue $e^{-i\phi} = \cos\phi - i\sin\phi$. The product of these two conjugate eigenvalues is 1. Thus, an orthogonal matrix of uneven dimension has at least one real eigenvalue (+1 or -1).

The eigenvalues of a Hausholder matrix $\boldsymbol{Q}_{HH}(\boldsymbol{v})$ are all real (recall that it is symmetric!). In fact, in dimension d its eigenvalues are -1 (one time) and 1 ($d-1$ times). Since a transposition matrix $\boldsymbol{T}$ is a special Householder matrix they have the same eigenvalues.

A.6.10 Positive definite matrices

If all eigenvalues of a square matrix $\boldsymbol{A}$ are real and $\lambda_i \geq 0$ then $\boldsymbol{A}$ is called *positive semi-definite*. If all eigenvalues are strictly positive $\lambda_i > 0$ then $\boldsymbol{A}$ is called *positive definite*.

Note that a matrix does not need to be symmetric to be positive definite, e.g. $\begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix}$ has positive eigenvalues 5 and 1. It also has a complete set of eigenvectors and is diagonisable.

A symmetric matrix $\boldsymbol{A}$ is positive definite if the quadratic form $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} > 0$ for any non-zero $\boldsymbol{x}$, and it is positive semi-definite if $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \geq 0$. This holds also the other way around: a symmetric positive definite matrix (with positive eigenvalues) has a positive quadratic form, and a symmetric positive semi-definite matrix (with non-negative eigenvalues) a non-negative quadratic form.

A symmetric positive definite matrix always has a positive diagonal (this can be seen by setting $\boldsymbol{x}$ above to a unit vector with 1 at a single position, and 0 at all other elements). However, just requiring a positive diagonal is too weak to ensure positive definiteness of a symmetric matrix, for example $\begin{pmatrix} 1 & 10 \\ 10 & 1 \end{pmatrix}$ has a negative eigenvalue of -9. On the other hand, a symmetric matrix is indeed positive definite if it is strictly diagonally dominant, i.e. if all its diagonal elements are positive and are larger than the absolute value of any of the corresponding row or column elements. However, diagonal dominance is too restrictive as criterion to characterise all symmetric positive definite matrices, since there are many symmetric matrices that are positive definite but not diagonally dominant, such as $\begin{pmatrix} 1 & 2 \\ 2 & 5 \end{pmatrix}$.

Finally, the sum of a symmetric positive semi-definite matrix $\boldsymbol{A}$ and a symmetric

positive definite matrix $\boldsymbol{B}$ is itself symmetric positive definite because the corresponding quadratic form $\boldsymbol{x}^T(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{B}\boldsymbol{x} > 0$ is positive. Similarly, the sum of two symmetric positive (semi)-definite matrices is itself symmetric positive (semi)-definite.

A.7 Matrix decompositions

A.7.1 Diagonalisation and eigenvalue decomposition

If $\boldsymbol{A}$ is a square non-defective matrix then the eigensystem $\boldsymbol{U}$ is invertible and we can rewrite the eigenvalue equation to

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{-1}\,.$$

This is called the **eigendecomposition**, or **spectral decomposition**, of $\boldsymbol{A}$ and equivalently

$$\boldsymbol{\Lambda} = \boldsymbol{U}^{-1}\boldsymbol{A}\boldsymbol{U}$$

is the diagonalisation of $\boldsymbol{A}$.

If $\boldsymbol{A}$ is defective (i.e. $\boldsymbol{U}$ is singular) one can still *approximately* diagonalise $\boldsymbol{A}$ as there always exists a similarity transformation to $\boldsymbol{J} = \boldsymbol{M}\boldsymbol{A}\boldsymbol{M}^{-1}$ where $\boldsymbol{M}$ is a invertible matrix and $\boldsymbol{J}$ has Jordan canonical form, i.e. $\boldsymbol{J}$ is upper triangular with the (potentially complex) eigenvalues on the diagonal and some non-zero entries equal to 1 immediately above the main diagonal.

A.7.2 Orthogonal eigenvalue decomposition

For symmetric $\boldsymbol{A}$ with real eigenvalues and orthogonal matrix $\boldsymbol{U}$ the spectral decomposition becomes

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$$

and

$$\boldsymbol{\Lambda} = \boldsymbol{U}^T\boldsymbol{A}\boldsymbol{U}\,.$$

This special case is known as the orthogonal diagonalisation of $\boldsymbol{A}$.

The orthogonal decomposition for symmetric $\boldsymbol{A}$ is unique apart from the signs of the eigenvectors (columns of $\boldsymbol{U}$). Thus, in a computer application depending on the specific implementation of a numerical algorithm for eigenvalue decomposition the signs may vary.

A.7.3 Singular value decomposition

The **singular value decomposition** (SVD) is a generalisation of the orthogonal eigenvalue decomposition for symmetric matrices.

Any (!) rectangular matrix $\boldsymbol{A}$ of size $n \times d$ can be factored into the product

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$$

where $\boldsymbol{U}$ is a $n \times n$ orthogonal matrix, $\boldsymbol{V}$ is a second $d \times d$ orthogonal matrix and $\boldsymbol{D}$ is a diagonal but rectangular matrix of size $n \times d$ with $m = min(n, d)$ real diagonal elements $d_1, \ldots d_m$. The d_i are called singular values, and appear along the diagonal in $\boldsymbol{D}$ by order of magnitude.

The SVD is unique apart from the signs of the columns vectors in $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{D}$ (you can freely specify the column signs of any two of the three matrices). By convention the signs are chosen such that the singular values in $\boldsymbol{D}$ are all non-negative, which leaves ambiguity in columns signs of $\boldsymbol{U}$ and $\boldsymbol{V}$. Alternatively, one may fix the columns signs of $\boldsymbol{U}$ and $\boldsymbol{V}$, e.g. by requiring a positive diagonal, which then determines the sign of the singular values (thus allowing for negative singular values as well).

If $\boldsymbol{A}$ is symmetric then the SVD and the orthogonal eigenvalue decomposition coincide (apart from different sign conventions for singular values, eigenvalues and eigenvectors).

Since $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{V}\boldsymbol{D}^T\boldsymbol{D}\boldsymbol{V}^T$ and $\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{U}\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{U}^T$ the squared singular values correspond to the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}\boldsymbol{A}^T$. It also follows that $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}\boldsymbol{A}^T$ are both positive semi-definite symmetric matrices, and that $\boldsymbol{V}$ and $\boldsymbol{U}$ contain the respective sets of eigenvectors.

A.7.4 Polar decomposition

Any square matrix $\boldsymbol{A}$ can be factored into the product

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{B}$$

of an orthogonal matrix $\boldsymbol{Q}$ and a symmetric positive semi-definite matrix $\boldsymbol{B}$.

This follows from the SVD of $\boldsymbol{A}$ given as

$$\begin{aligned}\boldsymbol{A} &= \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T \\ &= (\boldsymbol{U}\boldsymbol{V}^T)(\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T) \\ &= \boldsymbol{Q}\boldsymbol{B}\end{aligned}$$

with non-negative $\boldsymbol{D}$. Note that this decomposition is unique as the sign ambiguities in the columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ cancel out in $\boldsymbol{Q}$ and $\boldsymbol{B}$.

A.7.5 Cholesky decomposition

A symmetric positive definite matrix $\boldsymbol{A}$ can be decomposed into a product of a triangular matrix $\boldsymbol{L}$ with its transpose

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T .$$

Here, $\boldsymbol{L}$ is a lower triangular matrix with positive diagonal elements.

This decomposition is unique and is called **Cholesky factorisation**. It is often used to check whether a symmetric matrix is positive definite as it is algorithmically less demanding than eigenvalue decomposition.

Note that some implementations of the Cholesky decomposition (e.g. in R) use upper triangular matrices $\boldsymbol{K}$ with positive diagonal so that $\boldsymbol{A} = \boldsymbol{K}^T\boldsymbol{K}$ and $\boldsymbol{L} = \boldsymbol{K}^T$.

A.8 Matrix summaries based on eigenvalues and singular values

A.8.1 Trace and determinant computed from eigenvalues

The eigendecomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{-1}$ allows to establish a link between the trace and the determinant and the eigenvalues of a matrix.

Specifically,

$$\begin{aligned}\mathrm{Tr}(\boldsymbol{A}) &= \mathrm{Tr}(\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{-1}) = \mathrm{Tr}(\boldsymbol{\Lambda}\boldsymbol{U}^{-1}\boldsymbol{U}) \\ &= \mathrm{Tr}(\boldsymbol{\Lambda}) = \sum_{i=1}^{d}\lambda_i\end{aligned}$$

thus the trace of a square matrix $\boldsymbol{A}$ is equal to the *sum* of its eigenvalues. Likewise,

$$\begin{aligned}\det(\boldsymbol{A}) &= \det(\boldsymbol{U})\det(\boldsymbol{\Lambda})\det(\boldsymbol{U}^{-1}) \\ &= \det(\boldsymbol{\Lambda}) = \prod_{i=1}^{d}\lambda_i\end{aligned}$$

therefore the determinant of $\boldsymbol{A}$ is the *product* of the eigenvalues.

The relationship between the eigenvalues of a square matrix and the trace and the determinant of that matrix is shown above for diagonisable matrices. However, it holds more generally for any square matrix, i.e. also for defective matrices. For the latter the Jordan canonical form $\boldsymbol{J}$ replaces $\boldsymbol{\Lambda}$ (in both cases the eigenvalues are simply the entries on the diagonal).

If any of the eigenvalues are equal to zero then $\det(\boldsymbol{A}) = 0$ and as hence $\boldsymbol{A}$ is singular and not invertible.

The trace and determinant of a real matrix are always real even though the individual eigenvalues may be complex.

A.8.2 Eigenvalues of a squared matrix

From the eigendecomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{-1}$ it is easy to see that the eigenvalues of $\boldsymbol{A}^2$ are simply the squared eigenvalues of $\boldsymbol{A}$ as

$$\boldsymbol{A}^2 = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{-1}\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{-1} = \boldsymbol{U}\boldsymbol{\Lambda}^2\boldsymbol{U}^{-1}$$

As a result we can compute the trace of $\boldsymbol{A}^2$ as the sum of the squared eigenvalues of $\boldsymbol{A}$, i.e. $\text{Tr}(\boldsymbol{A}^2) = \sum_{i=1}^{d} \lambda_i^2$, and the determinant as the product of squared eigenvalues, i.e $\det(\boldsymbol{A}^2) = \prod_{i=1}^{d} \lambda_i^2$.

If $\boldsymbol{A}$ is symmetric then $\text{Tr}(\boldsymbol{A}^2) = \text{Tr}(\boldsymbol{A}\boldsymbol{A}^T) = ||A||_F^2 = \sum_{i=1}^{d}\sum_{j=1}^{d} a_{ij}^2$. This leads to the identity

$$\sum_{i=1}^{d} \lambda_i^2 = \sum_{i=1}^{d}\sum_{j=1}^{d} a_{ij}^2$$

between the sum of the squared eigenvalues and the sum of all squared entries of a symmetric matrix $\boldsymbol{A}$.

A.8.3 Rank and condition number

The rank is the dimension of the space spanned by both the column and row vectors. A rectangular matrix of dimension $n \times d$ will have rank of at most $m = \min(n, d)$, and if the maximum is indeed achieved then it has full rank.

The condition number describes how well- or ill-conditioned a full rank matrix is. For example, for a square matrix a large condition number implies that the matrix is close to being singular and thus ill-conditioned. If the condition number is infinite then the matrix is not full rank.

The rank and condition of a matrix can both be determined from the m singular values $d_1, \ldots, d_m$ of a matrix obtained by SVD:

i) The rank is number of non-zero singular values.
ii) The condition number is the ratio of the largest singular value divided by the smallest singular value (absolute values if signs are allowed).

If a square matrix $\boldsymbol{A}$ is singular then the condition number is infinite, and it will not have full rank. On the other hand, a non-singular square matrix, such as a positive definite matrix, has full rank.

A.9 Functions of symmetric matrices

We focus on *symmetric* square matrices $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ which are always diagonisable with real eigenvalues $\boldsymbol{\Lambda}$ and orthogonal eigenvectors $\boldsymbol{U}$.

A.9.1 Definition of a matrix function

Assume a real-valued function $f(a)$ of a real number a. Then the corresponding matrix function $f(\boldsymbol{A})$ is defined as

$$f(\boldsymbol{A}) = \boldsymbol{U} f(\boldsymbol{\Lambda}) \boldsymbol{U}^T = \boldsymbol{U} \begin{pmatrix} f(\lambda_1) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & f(\lambda_d) \end{pmatrix} \boldsymbol{U}^T$$

where the function $f(a)$ is applied to the eigenvalues of $\boldsymbol{A}$. By construction $f(\boldsymbol{A})$ is real, symmetric and has real eigenvalues $f(\lambda_i)$.

Examples:

Example A.2. Matrix power: $f(a) = a^p$ (with p a real number)

Special cases of matrix power include :

- Matrix inversion: $f(a) = a^{-1}$
  Note that if the matrix $\boldsymbol{A}$ is singular, i.e. contains one or more eigenvalues $\lambda_i = 0$, then $\boldsymbol{A}^{-1}$ is not defined and therefore $\boldsymbol{A}$ is not invertible.

However, a so-called *pseudoinverse* can still be computed, by inverting the non-zero eigenvalues, and keeping the zero eigenvalues as zero.

- Matrix square root: $f(a) = a^{1/2}$
  Since there are multiple solutions to the square root there are also multiple matrix square roots. The principal matrix square root is obtained by using the positive square roots of all the eigenvalues. Thus the **principal matrix square root** of a positive semi-definite matrix is also positive semi-definite and it is unique.

Example A.3. Matrix exponential: $f(a) = \exp(a)$
Note that because $\exp(a) \geq 0$ for all real a the matrix $\exp(\boldsymbol{A})$ is positive semi-definite. Thus, the matrix exponential can be used to generate positive semi-definite matrices.

If $\boldsymbol{A}$ and $\boldsymbol{B}$ commute, i.e. if $\boldsymbol{AB} = \boldsymbol{BA}$, then $\exp(\boldsymbol{A}+\boldsymbol{B}) = \exp(\boldsymbol{A})\exp(\boldsymbol{B})$. However, this is not the case otherwise!

Example A.4. Matrix logarithm: $f(a) = \log(a)$
As the logarithm requires $a > 0$ the matrix $\boldsymbol{A}$ needs to be positive definite for $\log(\boldsymbol{A})$ to be defined.

A.9.2 Identities for the matrix exponential and logarithm

The above give rise to useful identities:

1) For any symmetric matrix $\boldsymbol{A}$ we have

$$\det(\exp(\boldsymbol{A})) = \exp(\mathrm{Tr}(\boldsymbol{A}))$$

because $\prod_i \exp(\lambda_i) = \exp(\sum_i \lambda_i)$ where λ_i are the eigenvalues of $\boldsymbol{A}$.

2) If we take the logarithm on both sides and replace $\exp(\boldsymbol{A}) = \boldsymbol{B}$ we get another identity for a symmetric positive definite matrix $\boldsymbol{B}$:

$$\log\det(\boldsymbol{B}) = \mathrm{Tr}(\log(\boldsymbol{B}))$$

because $\log(\prod_i \lambda_i) = \sum_i \log(\lambda_i)$ where λ_i are the eigenvalues of $\boldsymbol{B}$.

A.10 Matrix calculus

A.10.1 First order vector derivatives

A.10.1.1 Gradient

The **gradient** of a scalar-valued function $h(\boldsymbol{x})$ with vector argument $\boldsymbol{x} = (x_1, \ldots, x_d)^T$ is the vector containing the first order partial derivatives of $h(\boldsymbol{x})$ with regard to each $x_1, \ldots, x_d$:

$$\begin{aligned}\nabla h(\boldsymbol{x}) &= \begin{pmatrix} \frac{\partial h(\boldsymbol{x})}{\partial x_1} \\ \vdots \\ \frac{\partial h(\boldsymbol{x})}{\partial x_d} \end{pmatrix} \\ &= \frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}} \\ &= \text{grad } h(\boldsymbol{x})\end{aligned}$$

The symbol ∇ is called the **nabla operator** (also known as **del operator**).

Note that we write the gradient as a **column vector**. This is called the **denominator layout** convention, see https://en.wikipedia.org/wiki/Matrix_calculus for details. In contrast, many textbooks (and also earlier versions of these lecture notes) assume that gradients are row vectors, following the so-called numerator layout convention.

Example A.5. Examples for the gradient:

- $h(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{x} + b$. Then $\nabla h(\boldsymbol{x}) = \frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}} = \boldsymbol{a}$.
- $h(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}$. Then $\nabla h(\boldsymbol{x}) = \frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}} = 2\boldsymbol{x}$.
- $h(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$. Then $\nabla h(\boldsymbol{x}) = \frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}} = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{x}$.

A.10.1.2 Jacobian matrix

For a vector-valued function

$$\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), \ldots, h_m(\boldsymbol{x}))^T$$

we can also compute the vector derivative

$$\begin{aligned}\frac{\partial \boldsymbol{h}(\boldsymbol{x})}{\partial \boldsymbol{x}} &= \begin{pmatrix} \frac{\partial h_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial h_m(\boldsymbol{x})}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_1(\boldsymbol{x})}{\partial x_d} & \cdots & \frac{\partial h_m(\boldsymbol{x})}{\partial x_d} \end{pmatrix} \\ &= \left(\frac{\partial h_j(\boldsymbol{x})}{\partial x_i} \right) \\ &= (\nabla h_1(\boldsymbol{x}), \ldots, \nabla h_m(\boldsymbol{x}))\end{aligned}$$

which yields a matrix whose columns contain the gradient vectors of the component of $\boldsymbol{h}(\boldsymbol{x})$.

The transpose of this matrix is called the **Jacobian matrix** (of size m rows and d columns):

$$\begin{aligned} J_{\boldsymbol{h}}(\boldsymbol{x}) &= \begin{pmatrix} \nabla h_1(\boldsymbol{x})^T \\ \vdots \\ \nabla h_m(\boldsymbol{x})^T \end{pmatrix} \\ &= \left(\frac{\partial h_i(\boldsymbol{x})}{\partial x_j} \right) \\ &= \left(\frac{\partial \boldsymbol{h}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right)^T \end{aligned}$$

Note that by convention the Jacobian matrix contains the gradients in its rows.

Example A.6. $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{A}^T\boldsymbol{x} + \boldsymbol{b}$. Then $\frac{\partial \boldsymbol{h}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \boldsymbol{A}$ and $J_{\boldsymbol{h}}(\boldsymbol{x}) = \boldsymbol{A}^T$.

If $m = d$ then the Jacobian matrix is a square matrix and this allows to compute the **Jacobian determinant** $\det J_{\boldsymbol{h}}(\boldsymbol{x})$.

Both the Jacobian matrix and the Jacobian determinant are often called simply "the Jacobian".

If $\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x})$ is an invertible function with $\boldsymbol{x} = \boldsymbol{h}^{-1}(\boldsymbol{y})$ then the Jacobian matrix is invertible and the inverted matrix is in fact the Jacobian of the inverse function!

This allows to compute the Jacobian determinant of the backtransformation as as the inverse of the Jacobian determinant the original function:

$$\det J_{\boldsymbol{h}^{-1}}(\boldsymbol{y}) = (\det J_{\boldsymbol{h}}(\boldsymbol{x}))^{-1}$$

or in alternative notation

$$\det J_{\boldsymbol{x}}(\boldsymbol{y}) = \frac{1}{\det J_{\boldsymbol{y}}(\boldsymbol{x})}$$

A.10.2 Second order vector derivatives

The matrix of all second order partial derivates of scalar-valued function with vector-valued argument is called the **Hessian matrix**:

$$\begin{aligned}\nabla\nabla^T h(\boldsymbol{x}) &= \begin{pmatrix} \frac{\partial^2 h(\boldsymbol{x})}{\partial x_1^2} & \frac{\partial^2 h(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 h(\boldsymbol{x})}{\partial x_1 \partial x_d} \\ \frac{\partial^2 h(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 h(\boldsymbol{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 h(\boldsymbol{x})}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 h(\boldsymbol{x})}{\partial x_d \partial x_1} & \frac{\partial^2 h(\boldsymbol{x})}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 h(\boldsymbol{x})}{\partial x_d^2} \end{pmatrix} \\ &= \left(\frac{\partial h(\boldsymbol{x})}{\partial x_i \partial x_j} \right) \\ &= \frac{\partial^2 h(\boldsymbol{x})}{\partial \boldsymbol{x} \partial \boldsymbol{x}^T}\end{aligned}$$

By construction the Hessian matrix is square and symmetric.

Example A.7. $h(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}$. Then $\nabla\nabla^T h(\boldsymbol{x}) = \frac{\partial^2 h(\boldsymbol{x})}{\partial \boldsymbol{x} \partial \boldsymbol{x}^T} = (\boldsymbol{A} + \boldsymbol{A}^T)$.

A.10.3 First order matrix derivatives

The derivative of a scalar-valued function $f(\boldsymbol{X})$ with regard to a matrix argument $\boldsymbol{X}$ can also be defined and results in a matrix. Below you find some matrix calculus rules (written in **denominator layout** convention). See https://en.wikipedia.org/wiki/Matrix_calculus for further examples.

Example A.8. $\frac{\partial \mathrm{Tr}(\boldsymbol{A}^T \boldsymbol{X})}{\partial \boldsymbol{X}} = \boldsymbol{A}$

Example A.9. $\frac{\partial \mathrm{Tr}(\boldsymbol{A}^T \boldsymbol{X} \boldsymbol{B})}{\partial \boldsymbol{X}} = \boldsymbol{A}\boldsymbol{B}^T$

Example A.10. $\frac{\partial \mathrm{Tr}(\boldsymbol{X}^T \boldsymbol{A} \boldsymbol{X})}{\partial \boldsymbol{X}} = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{X}$

Example A.11. $\frac{\partial \log\det(\boldsymbol{X})}{\partial \boldsymbol{X}} = \frac{\partial \mathrm{Tr}(\log \boldsymbol{X})}{\partial \boldsymbol{X}} = (\boldsymbol{X}^{-1})^T$

Appendix B

Further study

In this module we can only touch the surface of the field of multivariate statistics and machine learning. If you would like to study further I recommend the following books below as a starting point.

B.1 Recommended reading

For multivariate statistics and machine learning:

- Marden (2015) *Multivariate Statistics: Old School*
- Rogers and Girolami (2017) *A first course in machine learning (2nd Edition)*. Chapman and Hall / CRC.
- James et al. (2021) *An introduction to statistical learning with applications in R (2nd edition)*. Springer.

B.2 Advanced reading

Additional (advanced) reference books for probabilistic machine learning are:

- Bishop (2006) *Pattern recognition and machine learning*. Springer.
- Hastie, Tibshirani, and Friedman (2009) *The elements of statistical learning: data mining, inference, and prediction*. Springer.
- Murphy (2012) *Machine learning: a probabilistic perspective*. MIT Press.
- Fan et al. (2020) *Statistical Foundations of Data Science*. Chapman and Hall / CRC.
- Murphy (2022) *Probabilistic Machine Learning: An Introduction*. MIT Press.

You can find further suggestions on my list of online textbooks in statistics and machine learning.

Bibliography

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer. https://www.microsoft.com/en-us/research/people/cmbishop/prml-book/.

Fan, J., R. Li, C.-H. Zhang, and H. Zou. 2020. *Statistical Foundations of Data Science*. Chapman; Hall / CRC.

Hastie, T., R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer. https://web.stanford.edu/~hastie/ElemStatLearn/.

James, G., D. Witten, T. Hastie, and R. Tibshirani. 2021. *An Introduction to Statistical Learning with Applications in R*. 2nd ed. Springer. https://www.statlearning.com.

Marden, J. I. 2015. *Multivariate Statistics: Old School*. CreateSpace. http://stat.istics.net/Multivariate.

Murphy, K. P. 2012. *Machine Learning: A Probabilistic Perspective*. MIT Press.

———. 2022. *Probabilistic Machine Learning: An Introduction*. MIT Press. https://probml.github.io/pml-book/book1.html.

Rogers, S., and M. Girolami. 2017. *A First Course in Machine Learning*. 2nd ed. Chapman; Hall / CRC.

Zhang, A., Z. C. Lipton, M. Li, and A. J. Smola. 2022. *Dive into Deep Learning*. https://d2l.ai.