Lecture Notes

Data Mining and Exploration

Michael Gutmann
The University of Edinburgh

Spring Semester 2017

February 27, 2017

Contents

Chapter 1

First steps in exploratory data analysis

This chapter is about the first steps of exploratory data analysis. It is assumed that we have available n data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ each containing d attributes and that the data have been transformed to numbers. Each data point $\boldsymbol{x}_i$ thus is a d dimensional vector. We first explain ways to describe any of the d attributes in isolation and then methods to describe the joint behaviour of two attributes at a time. This is followed by some elements of preprocessing for further analysis of the data.

1.1 Distribution of single variables

We are here concerned with the properties of single attributes, e.g. the 10-th coordinate of the multivariate data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. By extracting the coordinate of interest from each data point $\boldsymbol{x}_i$, we obtain the univariate data $x_1, \ldots, x_n$ that we will describe using graphs or numerical summaries.

1.1.1 Numerical summaries

Summaries that characterise the typical value (location), the variability (scale), as well as symmetry and tail-behaviour (shape) of the data are presented.

Location

The perhaps simplest summary is the average value m of the data

$$m = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{1.1}$$

It is a measure of the typical or central value of the data. We also say that it measures the "location" of the data. The average value is further called the sample mean.

Assuming that the data were drawn from a random variable x with probability density function $p(.)$, the average value m of the data is an estimate of the mean

or expected value of x,

$$\mathbb{E}(x) = \int \xi p(\xi) d\xi. \tag{1.2}$$

There are other ways to estimate $\mathbb{E}(x)$ or to measure the location of the data.

The sample median is defined to be the centre of the data: equal proportions of the x_i lie above and below the median. It can be computed by ordering the x_i from small to large and then taking the middle value. Let $x_{(i)}$ denote the ordered data, i.e.

$$x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n)}, \tag{1.3}$$

the median is then given by

$$\text{median}(x_1, \ldots, x_n) = \text{median}(x_i) = \begin{cases} x_{((n+1)/2)} & \text{if } n \text{ is odd} \\ \frac{1}{2}(x_{(n/2)} + x_{(n/2+1)}) & \text{if } n \text{ is even} \end{cases} \tag{1.4}$$

In contrast to the sample mean, the sample median is robust to outliers. Assume that a data point is recorded as $x_i + \delta$ rather than as x_i because of a malfunction of the measurement device. The average then changes from m to $m + \delta/n$, which can be arbitrarily large, while the sample median changes at most to a neighbouring data point. Let, for example, the ordered data be

$$\begin{pmatrix} 1 & 1 & 3 & 3 & 5 & 7 & 9 & 11 & 18 & 20 \end{pmatrix} \tag{1.5}$$

The sample average is 7.8 while the median is $(5 + 7)/2 = 6$. If the data point 20 gets wrongly recorded as 200, the mean changes to 25.8 while the median stays the same. If 7 changes to 7000, the sample mean is 707.1 while the median changes only from 6 to 7.

Another measure of the location of the data that is more robust than the average is the trimmed average. It is the average of the data when leaving out the smallest and largest $k < n/2$ values,

$$\frac{1}{n - 2k} \sum_{i=k+1}^{n-k} x_{(i)}. \tag{1.6}$$

If $k = 0$, the trimmed average is the usual sample average. As k approaches $n/2$, the trimmed average approaches the median.

The sample average (mean) has the advantage that it can be easily computed in parallel for all dimensions of the multivariate data $\boldsymbol{x}_i$. Let $m_1, \ldots, m_d$ be the average values of the different dimensions of $\boldsymbol{x}_i$ computed as in (1.1). We then have

$$\boldsymbol{m} = \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_d \end{pmatrix} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i. \tag{1.7}$$

The sample mean $\boldsymbol{m}$ can be written as a matrix-vector multiplication. While computationally not an efficient way to compute $\boldsymbol{m}$, it is helpful in analytical work. Let $\boldsymbol{X}$ be the $d \times n$ data matrix with the $\boldsymbol{x}_i$ in its *columns*,

$$\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n). \tag{1.8}$$

We can then write $\boldsymbol{m}$ as

$$\boldsymbol{m} = \boldsymbol{X}\frac{1}{n}\mathbf{1}_n, \tag{1.9}$$

where $\mathbf{1}_n$ is a column vector containing n ones, i.e. $\mathbf{1}^\top = (1, \cdots, 1)^\top$ (see e.g. Section A.3).

Scale

A basic way to measure the scale of the data is to determine how much, on average, they deviate from the average value,

$$v = \frac{1}{n}\sum_{i=1}^{n}(x_i - m)^2 = \frac{1}{n}\sum_{i=1}^{n}x_i^2 - m^2. \tag{1.10}$$

This measure is known as the sample variance. It is an estimator of the variance of the random variable x,

$$\mathbb{V}(x) = \int (\xi - \mathbb{E}(x))^2 p(\xi) d\xi = \mathbb{E}(x^2) - \mathbb{E}(x)^2. \tag{1.11}$$

The variance is the 2nd central moment of x – the mean is the first moment while $E(x^2)$ is the second moment. Other estimators of the variance divide by $n-1$ rather than n. Note that both the variance and the sample variance have different units than the data. Hence, it is often better to indicate their square root. The square root of the (sample) variance is called the (sample) standard deviation.

Because of the squaring, the sample variance is more affected by outliers than the sample mean. The median can be used to obtain a more robust measure of the scale of the data: instead of measuring the average squared deviation from the average, we measure the median absolute deviation from the median (MAD),

$$\text{MAD} = \text{median}\left(|x_i - \text{median}(x_i)|\right) \tag{1.12}$$

This measure has the same units as the x_i themselves.

The range $x_{(n)} - x_{(1)}$, i.e. the difference between the largest and the smallest value, is another measure of the scale the data, but it is not robust. A more robust quantity is the difference between the upper and lower end of what makes the central 50% of the data. This is called the interquartile range (IQR). Half of the data are between $x_{(\lceil n/4 \rceil)}$ and $x_{(\lceil 3n/4 \rceil)}$ so that

$$\text{IQR} = x_{(\lceil 3n/4 \rceil)} - x_{(\lceil n/4 \rceil)}, \tag{1.13}$$

where $\lceil 3n/4 \rceil$ means $3n/4$ rounded up to the next higher integer. The number $x_{(\lceil n/4 \rceil)}$ is called the first quartile and often denoted by Q_1, and $x_{(\lceil 3n/4 \rceil)} = Q_3$ is the third quartile. The second quartile Q_2 is the median. For example,

$$(1 \quad 1 \quad \underbrace{\underset{Q1}{3} \quad 3 \quad 5 \quad \underset{Q_2}{} \quad 7 \quad 9 \quad \underset{Q_3}{11}}_{\text{IQR}} \quad 18 \quad 20), \tag{1.14}$$

so that IQR = 8, while the sample standard deviation is $\sqrt{v} = 6.76$ and MAD = 4.

The median and the quartiles are examples of sample quantiles. The α-th sample quantile q_α is roughly the data point with a proportion α of the ordered data $x_{(i)}$ to its left, i.e. $q_\alpha \approx x_{(\lceil n\alpha \rceil)}$. For example, the minimum and the maximum are the 0 and 1 quantiles q_0 and q_1; the median the 0.5 quantile $q_{0.5}$ and the quartiles Q_1 and Q_3 are equal to $q_{0.25}$ and $q_{0.75}$ respectively. Like for the median, quantiles are often computed by interpolation if αn is not an integer.

Shape

Skewness is a quantity that measures the symmetry of the data. For a random variable x, skewness is defined as its third standardised moment,

$$\text{skew}(x) = \mathbb{E}\left[\left(\frac{x - \mathbb{E}(x)}{\sqrt{\mathbb{V}(x)}}\right)^3\right] = \mathbb{E}\left[\left(\frac{x-\mu}{\sigma}\right)^3\right], \tag{1.15}$$

with mean $\mu = \mathbb{E}(x)$ and standard deviation $\sigma = \sqrt{\mathbb{V}(x)}$. The subtraction of the mean and the division by the standard deviation normalises x to have zero mean and unit variance, which "standardises" x by removing the location and scale information before taking the third power. Removal of the scale information eases the comparison of skewness between different random variables.

A random variable that is symmetric around its mean has skewness zero. As

$$\text{skew}(x) = \int \left(\frac{\xi - \mu}{\sigma}\right)^3 p(\xi) d\xi \tag{1.16}$$

$$= \underbrace{\int_{\xi \le \mu} \left(\frac{\xi - \mu}{\sigma}\right)^3 p(\xi) d\xi}_{\le 0} + \underbrace{\int_{\xi > \mu} \left(\frac{\xi - \mu}{\sigma}\right)^3 p(\xi) d\xi}_{\ge 0} \tag{1.17}$$

skewness is positive if x tends to take on values much larger than the mean (heavy upper tails). Conversely, skewness is negative in case of heavy lower tails. For a data set $x_1, \ldots, x_n$, skewness can be measured by replacing the expectations with sample averages, as we have done in case of the mean and variance. The measure is then called sample skewness.

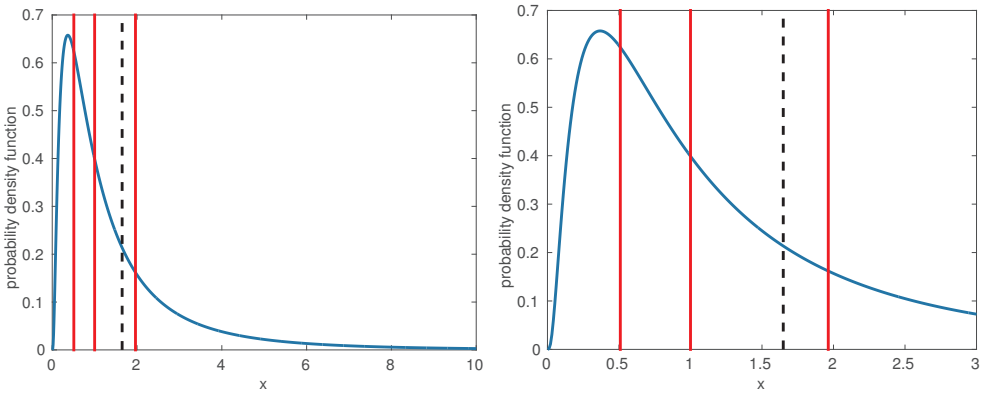
Due to the third power, sample skewness is sensitive to outliers. A more robust measure can be obtained by means of the quartiles,

$$\text{Galton's measure of skewness} = \frac{(Q_3 - Q_2) - (Q_2 - Q_1)}{Q_3 - Q_1}. \tag{1.18}$$

The denominator is the interquartile range and normalises the skewness measure like the standard deviation in (1.15). By definition of the quartiles both $Q_3 - Q_2$ and $Q_2 - Q_1$ are positive. The first term measures the range of the third quarter while the second term measures the range of the second quarter. Galton's skewness thus computes the difference between the ranges of the two quarters in a normalised way. It is positive if the range of the third quarter is larger than the range of the first quarter, and conversely. Figure 1.1 shows an example.

The fourth standardised moment is called the kurtosis,

$$\text{kurt}(x) = \mathbb{E}\left[\left(\frac{x-\mu}{\sigma}\right)^4\right]. \tag{1.19}$$



(a) Long (heavy) upper tails (b) Zoom

Figure 1.1: Example of positive skewness. The distribution has skewness equal to 6.18 according to (1.15), its interquartile range is 1.45, and $Q_3 - Q_2 = 0.96$ while $Q_2 - Q_1 = 0.49$, so that Galton's measure of skewness is positive. The black dashed line indicates the mean, while the three red lines show from left to right Q_1, Q_2, and Q_3.

Due to the fourth power, kurtosis is insensitive to the symmetry of the distribution of x. It measures how often x takes on values that are considerably larger or smaller than its standard deviation; it is said to measure how heavy the tails of the distribution of x are. Figure 1.2 shows the function $u \mapsto u^4$. It is relatively flat for $-1 < u < 1$ so that kurtosis basically ignores the behaviour of x within one standard deviation around its mean. The function then grows rapidly and values larger than two standard deviations away from the mean contribute strongly to the value of kurtosis.

A Gaussian random variable with probability density function

$$p(\xi) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\xi-\mu)^2}{2\sigma^2}\right) \tag{1.20}$$

has kurtosis equal to three. This is often taken as a reference value, and people report the excess kurtosis,

$$\text{excess kurtosis}(x) = \text{kurt}(x) - 3. \tag{1.21}$$

Positive excess kurtosis indicates that the random variable has heavier tails than a Gaussian random variable. This means that it produces large values more often than a Gaussian. Negative excess kurtosis, on the other hand, indicates a random variable with fewer and less extreme values than a Gaussian. Figure 1.3 shows examples. Both (excess) kurtosis and skewness can thus be taken as measures of non-Gaussianity of the data.

Kurtosis can be computed by replacing the expectation with a sample average, or more robustly, by means of quantiles,

$$\text{robust kurtosis}(x) = \frac{(q_{7/8} - q_{5/8}) + (q_{3/8} - q_{1/8})}{q_{3/4} - q_{1/4}}. \tag{1.22}$$

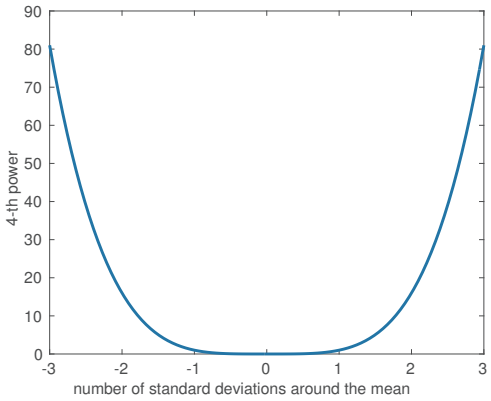
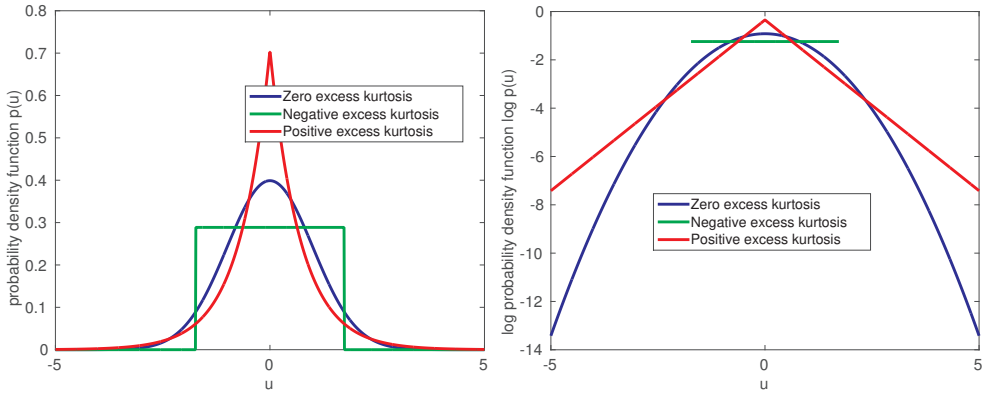


Figure 1.2: The figure shows the function $u \mapsto u^4$ that occurs in the definition of the kurtosis. It is relatively flat on $(-1, 1)$ and grows quickly for values outside the interval.



(a) probability density functions (b) log probability density functions

Figure 1.3: Probability density functions (pdfs) with zero, negative, and positive excess kurtosis. The curves shown are correspondingly the Gaussian (blue), uniform (green), and Laplace (red) probability density functions.

The denominator is the interquartile range, and the numerator measures the length of the upper and lower tails. Further robust measures of kurtosis and skewness are discussed by Kim and White (2004).

1.1.2 Graphs

Graphs often allow us to see interesting patterns in the data more easily than numerical summaries.

Histogram

The histogram is among the most used graphs. To construct the histogram, we count how many times certain values occur among the x_i. If the x_i can only take on finitely many values $v_1, \ldots v_k$, the histogram plots the frequency of occurrence

$f_1, \ldots, f_k$ of each value, i.e.

$$f_j = \frac{n_j}{n}, \qquad n_j = \sum_{i=1}^{n} 1_{\{v_j\}}(x_i), \qquad 1_{\{v_j\}}(\xi) = \begin{cases} 1 & \text{if } \xi = v_j \\ 0 & \text{otherwise} \end{cases} \tag{1.23}$$

The sum $\sum_{i=1}^{n} 1_{\{v_j\}}(x_i)$ equals the number of times the value v_j occurs in the data set. Here, the values v_j do not need to be numerical.

If the x_i are real numbers, we can quantise them so that they take on only k different values, and then proceed as above. The usual way to do this is via binning: We form k bins $B_1, \ldots, B_k$ and count the number of data points n_j that fall into each bin B_j,

$$n_j = \sum_{i=1}^{n} 1_{B_j}(x_i), \qquad 1_{B_j}(\xi) = \begin{cases} 1 & \text{if } \xi \in B_j \\ 0 & \text{otherwise} \end{cases} \tag{1.24}$$

We can then compute the frequencies $f_j = n_j/n$ as before. The bins are often chosen to have equal width h. For equal bin sizes the bins B_j are

$$B_1 = [L, L+h), \quad B_2 = [L+h, L+2h), \quad \cdots, \quad B_k = [L+(k-1)h, L+kh]. \tag{1.25}$$

They are centred at the locations $L + h/2 + jh$. The starting value L, the bin-width h and the number of bins k are parameters that the user needs to choose. One may choose L to correspond to the smallest value of all x_i and given k, the bin-width h such that the bins cover the whole range of the data.

Figure 1.4(a) shows an example. We can see that different starting values L may lead to differently looking histograms.

The frequencies can further be converted to probability density estimates by dividing them by the length of the bins. The estimated probability density function $\hat{p}(x)$ is piecewise constant.[1] For equal bin sizes h, we have

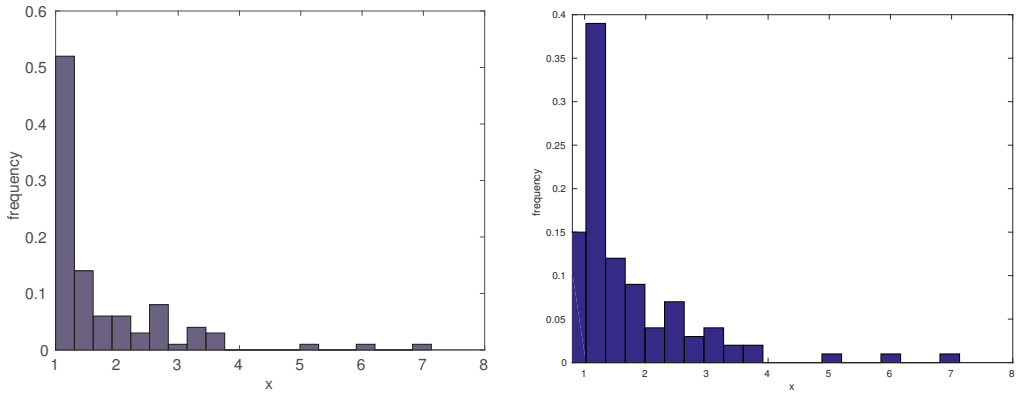
$$\hat{p}(x) = \frac{f_j}{h} = \frac{n_j}{nh} \qquad \text{if } x \in B_j. \tag{1.26}$$

This is really just a re-normalisation of the frequencies, but the interpretation is a bit different. We can think that we first discretise x to equal the bin centre of B_j, then count the number of data points n_j in its neighbourhood of size h, and finally normalise the counts by nh to convert them to a density.

Kernel density estimate

A critique of density estimates like (1.26) is that they depend on the starting point L, and that they are not smooth even if the data were drawn from a distribution with a smooth probability density function. Kernel density estimation addresses these two shortcomings of histograms. It still requires the specification of h, which is called the bandwidth in kernel density estimation.

[1] We have here overloaded the symbol x to refer to both the random variable and the value it may take. This is done very often, and also in this lecture without any further warning.



(a) $L = \min_i(x_i)$ (b) $L = \min_i(x_i) - 0.3$

Figure 1.4: Describing the scalar data by histograms. The edges of the bars correspond to the edges of the bins used for the histogram. (a) and (b) show histograms with different starting vales L.

The dependency on the starting point L can be removed by not discretising x in (1.26). We just directly count the number of points in the neighbourhood of x, which can be done by computing

$$n(x) = \sum_{i=1}^{n} \sqcap_h(x - x_i), \qquad \sqcap_h(\xi) = \begin{cases} 1 & \text{if } \xi \in [-\frac{h}{2}, \frac{h}{2}] \\ 0 & \text{otherwise.} \end{cases} \tag{1.27}$$

The function $\sqcap_h(.)$ is called the rectangular or the boxcar function. Normalising the counts as before yields

$$\hat{p}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h} \sqcap_h (x - x_i). \tag{1.28}$$

This quantity is a kernel density estimate, and the function $1/h \sqcap_h (.)$ is called the boxcar kernel.

When we use the boxcar kernel, we stop counting data points as soon as they are further than a $h/2$ distance away from x. In a sense, the data points are assigned zero weight if outside the neighbourhood and unit weight when inside. Instead of this binary weight assignment, it is often more reasonable to assign to each data point a weight that decreases smoothly with the distance from the query point x. This can be done by replacing the boxcar kernel with another kernel $K_h(.)$ so that the kernel density estimate is

$$\hat{p}(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i). \tag{1.29}$$

The kernel $K_h(.)$ must integrate to one for $\hat{p}(x)$ to integrate to one. A popular choice for $K_h(.)$ is the Gaussian kernel

$$K_h(\xi) = \frac{1}{\sqrt{2\pi h^2}} \exp\left(-\frac{\xi^2}{2h^2}\right). \tag{1.30}$$

Other kernels are plotted in Figure 1.5. Figure 1.6 shows the densities estimated by rescaling the histogram, with the boxcar kernel, and with the Gaussian kernel.

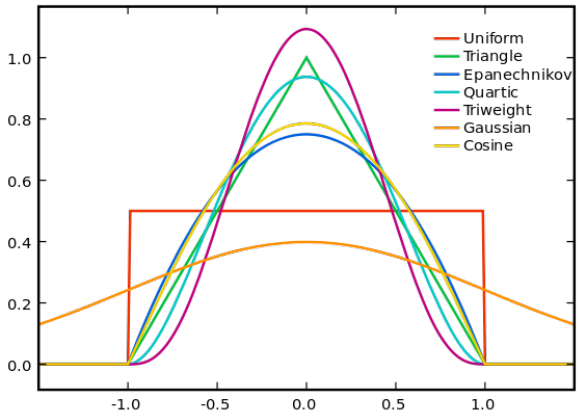
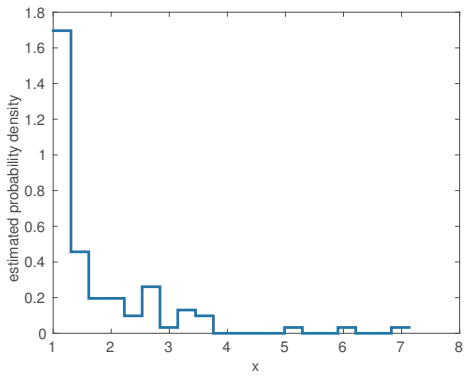
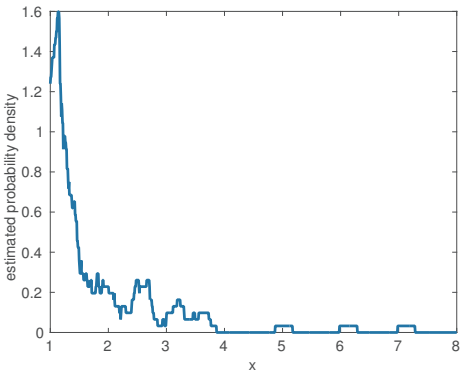


Figure 1.5: Kernels used in kernel density estimation. Figure from https://en.wikipedia.org/wiki/Kernel_(statistics)

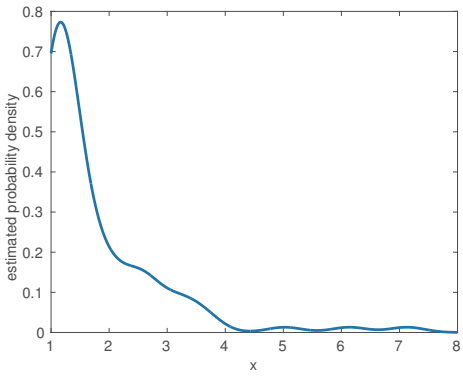
.



(a) Scaled histogram



(b) Boxcar kernel



(c) Gaussian kernel

Figure 1.6: Estimating the probability density function from (a) the histogram or (b-c) via kernel density estimation.

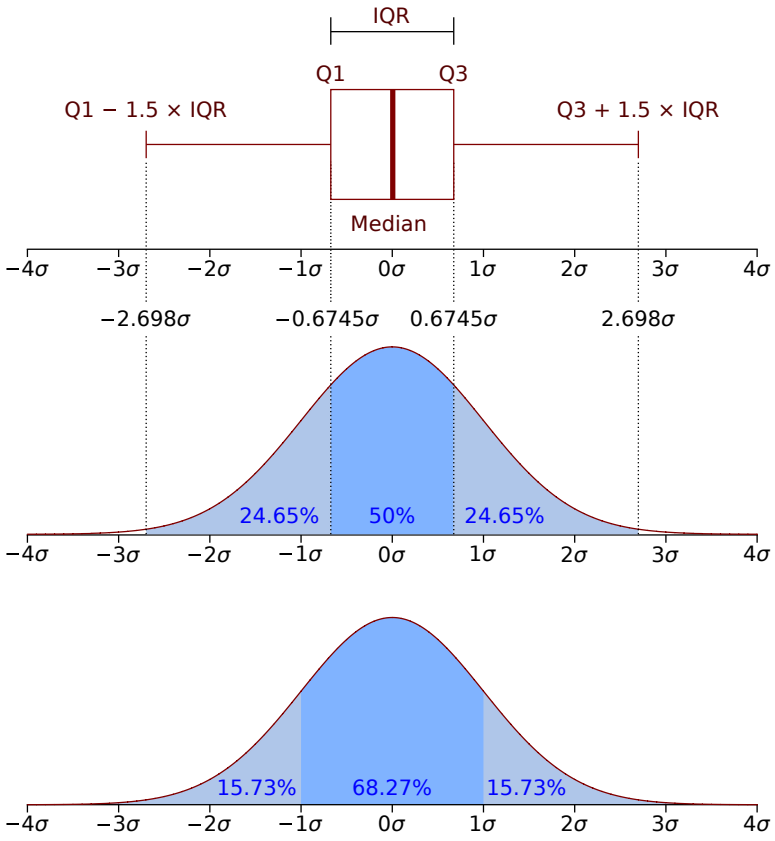


Figure 1.7: Boxplot and comparison to the Gaussian probability density function. Figure from https://en.wikipedia.org/wiki/Box_plot

Boxplot

The boxplot visualises several key quantile-based summaries in a single graph. Figure 1.7 shows the boxplot (top) and maps the summaries to the corresponding areas of the Gaussian distribution. Violin plots are a modification that combine quantile information with a kernel density estimate (Hintze and Nelson, 1998).

1.2 Joint distribution of two variables

The previous section described the behaviour of single variables in isolation. It was concerned with summarising and estimating the probability distribution of a single attribute of the multivariate data. This section is about numerical summaries and graphs for describing the joint probability distribution of two variables.

1.2.1 Numerical summaries

We present measures that quantify the tendency of two variables to vary together.

Linear relationship: covariance and correlation

Let x and y be two random variables with mean μ_x and μ_y, respectively. The covariance $\text{cov}(x, y)$ measures the strength of association between them. It is defined as

$$\text{cov}(x, y) = \mathbb{E}\left[(x - \mu_x)(y - \mu_y)\right] = \mathbb{E}\left(xy\right) - \mu_x\mu_y. \tag{1.31}$$

From the definition, we have $\text{cov}(x, x) = \mathbb{V}(x)$ and $\text{cov}(ax + b, y) = a\,\text{cov}(x, y)$. The value of the covariance thus depends on the scale of x and y.

A measure of association that does not depend on their scale is the correlation coefficient ρ,

$$\rho(x, y) = \frac{\text{cov}(x, y)}{\sqrt{\mathbb{V}(x)\,\mathbb{V}(y)}}. \tag{1.32}$$

Denoting the standard deviation of x and y by σ_x and σ_y, the correlation coefficient can be written as

$$\rho(x, y) = \mathbb{E}\left[\left(\frac{x - \mu_x}{\sigma_x}\right)\left(\frac{y - \mu_y}{\sigma_y}\right)\right], \tag{1.33}$$

which means that x and y are standardised to zero mean and unit variance before the expectation is taken. If ρ is positive, $x > \mu_x$ and $y > \mu_y$ tend to co-occur, and x and y are said to be positively correlated. If $\rho < 0$, x and y are negatively correlated. If $\rho = 0$, the random variables are uncorrelated, and $\mathbb{E}(xy) = \mathbb{E}(x)\,\mathbb{E}(y)$.

Assume $y = ax + b$, where $a \neq 0$ and b are some constants. Then $\mu_y = a\mu_x + b$ and the variance of y is

$$\mathbb{V}(y) = \mathbb{E}((y - \mu_y)^2) = \mathbb{E}((ax + b - a\mu_x - b)^2) = \mathbb{E}(a^2(x - \mu_x)^2) = a^2\,\mathbb{V}(x). \tag{1.34}$$

The covariance is

$$\text{cov}(x, y) = \mathbb{E}\left[(x - \mu_x)(y - \mu_y)\right] = \mathbb{E}\left[(x - \mu_x)(ax + b - a\mu_x - b)\right] = a\,\mathbb{V}(x) \tag{1.35}$$

so that

$$\rho(x, y) = \frac{a\,\mathbb{V}(x)}{\sqrt{\mathbb{V}(x)a^2\,\mathbb{V}(x)}} = \frac{a}{\sqrt{a^2}} = \begin{cases} 1 & \text{if } a > 0 \\ -1 & \text{if } a < 0 \end{cases} \tag{1.36}$$

That is, if y is linearly related to x, the correlation coefficient is one for positive slopes a, and minus one for negative slopes. The magnitude of a does not affect the value of ρ.

What does affect the value of ρ? Let $y = x + n$ where x and n are uncorrelated. We then have $\mathbb{V}(y) = \mathbb{V}(x) + \mathbb{V}(n)$, which we denote by $\sigma_x^2 + \sigma_n^2$, and $\text{cov}(x, y) = \mathbb{V}(x) = \sigma_x^2$, so that

$$\rho(x, y) = \frac{\sigma_x^2}{\sqrt{\sigma_x^2(\sigma_x^2 + \sigma_n^2)}} = \frac{1}{\sqrt{1 + \frac{\sigma_n^2}{\sigma_x^2}}}. \tag{1.37}$$

Hence, $|\rho|$ becomes small if σ_n^2 dominates σ_x^2, that is, if the so-called signal-to-noise ratio σ_x^2/σ_n^2 is small. Importantly, $\rho \approx 0$ only means that x and y are not linearly related. It does not mean that x and y are not related, see Figure 1.8.

The covariance and correlation coefficient can be estimated from n tuples (x_i, y_i), that is from data $(x_1, y_1), \ldots, (x_n, y_n)$, by replacing the expectation with the sample average as we have done before.

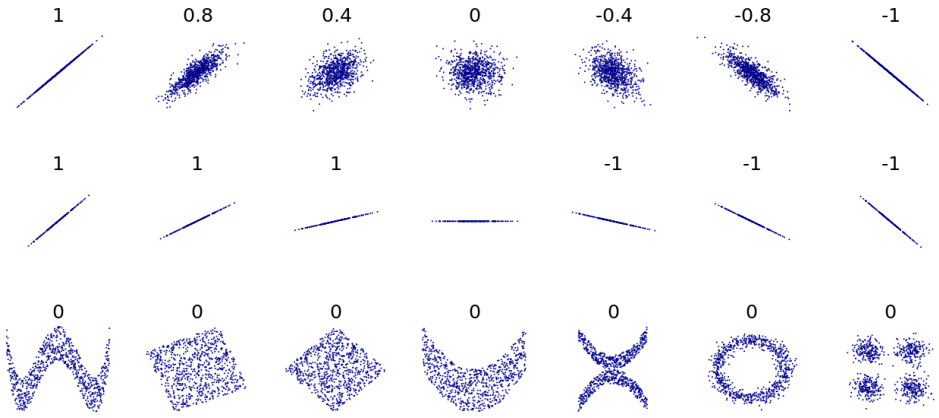


Figure 1.8: Correlation coefficients for different data sets. Figure from https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient.

Covariance matrix

For a random vector $\boldsymbol{x} = (x_1, \ldots, x_d)^\top$, the covariance matrix is the $d \times d$ matrix $\boldsymbol{C} = \mathbb{V}(\boldsymbol{x})$ that contains all covariances of the variables,

$$\boldsymbol{C} = \begin{pmatrix} \operatorname{cov}(x_1, x_1) & \operatorname{cov}(x_1, x_2) & \cdots & \operatorname{cov}(x_1, x_d) \\ \operatorname{cov}(x_2, x_1) & \operatorname{cov}(x_2, x_2) & \cdots & \operatorname{cov}(x_2, x_d) \\ \vdots & \vdots & & \vdots \\ \operatorname{cov}(x_d, x_1) & \operatorname{cov}(x_d, x_2) & \cdots & \operatorname{cov}(x_d, x_d). \end{pmatrix} \tag{1.38}$$

Since $\operatorname{cov}(x_i, x_j) = \operatorname{cov}(x_j, x_i)$, $\boldsymbol{C}$ is symmetric, and since $\operatorname{cov}(x_i, x_i) = \mathbb{V}(x_i)$, the elements on the diagonal are the variances.

Let $\boldsymbol{\mu}$ be the d-dimensional vector with the means of the random variables x_i. We can then write $\boldsymbol{C}$ as

$$\boldsymbol{C} = \mathbb{E}\left(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top \right). \tag{1.39}$$

This follows immediately from the properties of the outer product $\boldsymbol{a}\boldsymbol{b}^\top$ between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, see e.g. Section A.2. Indeed, $(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top$ is a $d \times d$ matrix where the (i, j)-th element is $(x_i - \mathbb{E}(x_i))(x_j - \mathbb{E}(x_j))$. Its expectation is hence $\operatorname{cov}(x_i, x_j)$. By linearity of expectation

$$\boldsymbol{w}^\top \boldsymbol{C} \boldsymbol{w} = \mathbb{E}\left(\underbrace{\boldsymbol{w}^\top (\boldsymbol{x} - \boldsymbol{\mu})}_{\text{scalar}} \underbrace{(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{w}}_{\text{scalar}} \right) = \mathbb{E}\left((\boldsymbol{w}^\top (\boldsymbol{x} - \boldsymbol{\mu}))^2 \right) \geq 0, \tag{1.40}$$

which means that $\boldsymbol{C}$ is a positive semi-definite matrix. It thus has an eigenvalue decomposition $\boldsymbol{C} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$, where $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_d)$ is a diagonal matrix containing the eigenvalues $\lambda_i \geq 0$, and $\boldsymbol{U}$ is an orthogonal matrix with the eigenvectors as columns (see e.g. Appendix A for a linear algebra refresher). The total variance of the d random variables x_i is the sum of all eigenvalues. With

the definition of the trace a matrix, see e.g. (A.6), we have

$$\sum_{i=1}^{d} \mathbb{V}(x_i) = \operatorname{trace}(\boldsymbol{C}) \tag{1.41}$$

$$= \operatorname{trace}(\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top) \tag{1.42}$$

$$\overset{(A.8)}{=} \operatorname{trace}(\boldsymbol{\Lambda}\boldsymbol{U}^\top\boldsymbol{U}) \tag{1.43}$$

$$= \operatorname{trace}(\boldsymbol{\Lambda}) \tag{1.44}$$

$$= \sum_{i=1}^{d} \lambda_i, \tag{1.45}$$

where we have used that, for an orthogonal matrix $\boldsymbol{U}$, $\boldsymbol{U}^\top\boldsymbol{U}$ is the identity matrix.

If $\boldsymbol{x}$ has covariance matrix $\boldsymbol{C}$, the linearly transformed random variable $\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}$ has covariance matrix $\boldsymbol{A}\boldsymbol{C}\boldsymbol{A}^\top$. This is due to the linearity of expectation,

$$\mathbb{V}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}) = \mathbb{E}\left[(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}-\mathbb{E}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}))(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}-\mathbb{E}(\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}))^\top\right] \tag{1.46}$$

$$= \mathbb{E}\left[(\boldsymbol{A}\boldsymbol{x}-\mathbb{E}(\boldsymbol{A}\boldsymbol{x}))(\boldsymbol{A}\boldsymbol{x}-\mathbb{E}(\boldsymbol{A}\boldsymbol{x}))^\top\right] \tag{1.47}$$

$$= \mathbb{E}\left[(\boldsymbol{A}(\boldsymbol{x}-\mathbb{E}(\boldsymbol{x})))(\boldsymbol{A}(\boldsymbol{x}-\mathbb{E}(\boldsymbol{x})))^\top\right] \tag{1.48}$$

$$= \mathbb{E}\left[\boldsymbol{A}(\boldsymbol{x}-\mathbb{E}(\boldsymbol{x}))(\boldsymbol{x}-\mathbb{E}(\boldsymbol{x}))^\top\boldsymbol{A}^\top\right] \tag{1.49}$$

$$= \boldsymbol{A}\,\mathbb{E}\left[(\boldsymbol{x}-\mathbb{E}(\boldsymbol{x}))(\boldsymbol{x}-\mathbb{E}(\boldsymbol{x}))^\top\right]\boldsymbol{A}^\top \tag{1.50}$$

$$= \boldsymbol{A}\boldsymbol{C}\boldsymbol{A}^\top. \tag{1.51}$$

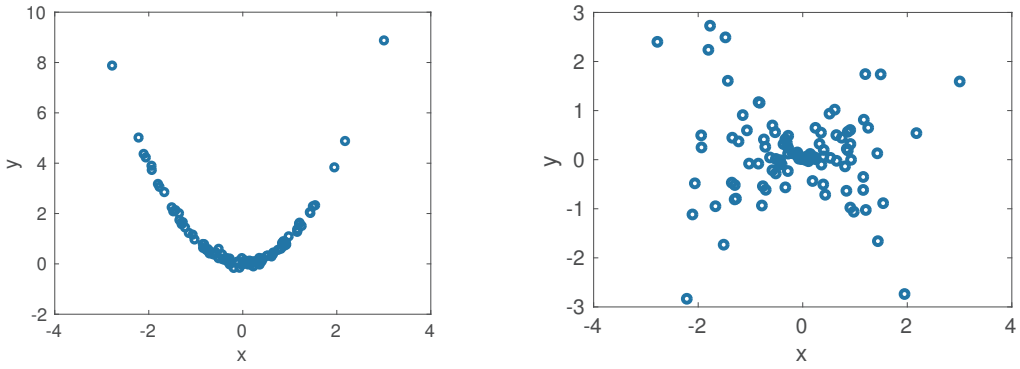
The correlation matrix $\boldsymbol{K}$ is the $d \times d$ matrix with all correlation coefficients. The correlation coefficient is essentially the covariance between two random variables that are standardised to unit variance (and zero mean), which can be achieved by the linear transformation $\boldsymbol{D}^{-1/2}(\boldsymbol{x}-\boldsymbol{\mu})$ where $\boldsymbol{D}$ contains the diagonal elements of $\boldsymbol{C}$, i.e. $\boldsymbol{D} = \operatorname{diag}(\mathbb{V}(x_1), \ldots, \mathbb{V}(x_d))$. We thus have

$$\boldsymbol{K} = \boldsymbol{D}^{-1/2}\boldsymbol{C}\boldsymbol{D}^{-1/2}. \tag{1.52}$$

By construction, the diagonal elements of $\boldsymbol{K}$ are all one. Both the correlation matrix and the covariance matrix can be computed from data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ by replacing the expectations with sample averages. In Section 1.3.2, we present a compact matrix expression.

If the dimension d is not too large, we can just display the numerical values of the covariance or correlation matrix. But for large d, this is not helpful. For large dimensions, options are

- visualising $\boldsymbol{C}$ or $\boldsymbol{K}$ as an $d \times d$ image with suitably chosen colour codes,
- summarising the distribution of the variances and correlation coefficients by any of the techniques from the previous section, e.g. histograms or kernel density estimates,
- doing principal component analysis for a more in-depth analysis (see Chapter 2).



(a) nonlinear functional relation

(b) variance dependency

Figure 1.9: Measuring nonlinear relationships between two variables. The linear correlation coefficient is small in both (a) and (b) The correlation of the absolute values, however, captures the relation between x and y. (a) $\rho(|x|, |y|) = 0.93$, (b) $\rho(|x|, |y|) = 0.68$.

Nonlinear relationships

A simple way to measure possible nonlinear relationships between two random variables x and y is to compute their covariance or correlation after transforming them nonlinearly, i.e. to compute

$$\rho(g(x), g(y)) = \frac{\operatorname{cov}(g(x), g(y))}{\sqrt{\mathbb{V}(g(x))\,\mathbb{V}(g(y))}} \tag{1.53}$$

for some nonlinear function g.

Figure 1.9 shows two examples. In Figure 1.9(a) there is a clear functional relation between x and y but the (linear) correlation coefficient is -0.15, wrongly indicating a negative correlation between x and y. Computing the correlation between the absolute values $|x|$ and $|y|$, however, yields a correlation coefficient of 0.93. In Figure 1.9(b), the variance of y depends on the magnitude of x. The linear correlation is practically zero while the absolute values have a correlation coefficient of 0.68.

Different nonlinearities g can be used to measure different properties of the data. The absolute value, for example, can be used to measure variance dependencies. And to measure whether x and y are monotonically related, one can correlate the ranks of the data points instead of their actual value. The latter quantity is known as Spearman's rank correlation coefficient.

1.2.2 Graphs

The graphs in Figure 1.9 are known as scatter plots. Each bubble corresponds to a data point. The scatter plot is one of the most used techniques to visualise the distribution of two variables. Colouring the bubbles and changing their size enables the visualisation of further dimensions or class labels.

Other graphs that are being used are two-dimensional extensions of the histogram and the kernel density estimator. For random vectors, we can show these

kind of graphs for all (unique) pairwise combinations. But as the dimension grows, the plots become impractical. In Chapter 3, methods for dimensionality reduction are presented that can be used to visualise high-dimensional data in the plane.

1.3 Simple preprocessing

Preprocessing refers to various operations that need to be performed in order to prepare the data for further analysis. We here discuss simple methods for outlier detection and data standardisation.

1.3.1 Simple outlier detection

An outlier is a data point that seems unusual compared to others. This is a vague definition, which reflects the various possible causes for outliers.

An outlier can be due to an error in the data gathering stage, for example because a measurement device did not work properly ("bad data"). A data point may, however, also appear unusual because it does not conform to the current assumptions that are made about the data. In the former case, we may omit the corrupted data points from the analysis, while in the latter case, the data points contain valuable information that should not be discarded.

Some bad data points can be spotted by the methods above for describing univariate or bivariate data. If there is a strong difference between the sample mean and sample median, for example, the cause may be a bad data point. Quantiles, histograms, and scatter plots further enable one to spot bad data points. Tukey's test is a classical method that is based on quartiles and considers data points outside the range

$$[Q_1 - k \text{ IQR}, Q_3 + k \text{ IQR}] \tag{1.54}$$

as outliers. Typically $k = 1.5$, so that points that fall beyond the whiskers of the boxplot in Figure 1.7 are considered outliers.

1.3.2 Data standardisation

Data standardisation refers classically to normalising the data to have zero (sample) mean and unit (sample) variance. It may, however, also refer to other kinds of transformations to make all variables comparable, for example, transforming the variables to be in $[0, 1]$. Common further transformations that are being used are removing the average value of each single data vector, re-scaling the vector to unit norm, or computing the logarithm of its values. The transformations are often problem dependent.

Denote the "raw" data vector by $\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n$ and the corresponding data matrix by $\tilde{\boldsymbol{X}}$,

$$\tilde{\boldsymbol{X}} = (\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n). \tag{1.55}$$

We next write the removal of the mean (centring) and scaling to unit variance as matrix operations.

Centring

Let us denote the average of the $\tilde{\boldsymbol{x}}_i$ by $\boldsymbol{m}$ and the centred data points by $\boldsymbol{x}_i$, so that

$$\boldsymbol{x}_i = \tilde{\boldsymbol{x}}_i - \boldsymbol{m}. \tag{1.56}$$

Let us further collect all centred data points into the matrix $\boldsymbol{X}$,

$$\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n). \tag{1.57}$$

We will now see that the centring operation can be written compactly as a matrix multiplication,

$$\boldsymbol{X} = \tilde{\boldsymbol{X}}\boldsymbol{H}_n, \qquad \boldsymbol{H}_n = \boldsymbol{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top, \tag{1.58}$$

where $\boldsymbol{I}_n$ is the $n \times n$ identity matrix and $\mathbf{1}_n^\top = (1, 1, \ldots, 1)^\top$ is a vector of ones. The matrix $\boldsymbol{H}_n$ is called the centring matrix.

This can be seen as follows: From (1.9), we know that we can express the sample mean $\boldsymbol{m} = (m_1, \cdots, m_d)^\top$ as

$$\boldsymbol{m} = \frac{1}{n}\sum_{i=1}^{n}\tilde{\boldsymbol{x}}_i = \tilde{\boldsymbol{X}}\frac{1}{n}\mathbf{1}_n, \tag{1.59}$$

Note that m_i is the average over all elements in *row* i of the matrix $\tilde{\boldsymbol{X}}$, i.e.

$$m_i = \frac{1}{n}\sum_{j=1}^{n}(\tilde{\boldsymbol{X}})_{ij}, \tag{1.60}$$

where $(\tilde{\boldsymbol{X}})_{ij}$ denotes the (ij)-th element of $\tilde{\boldsymbol{X}}$. Since $\boldsymbol{X}$ is obtained by subtracting vector $\boldsymbol{m}$ from each column of $\tilde{\boldsymbol{X}}$, we have

$$\boldsymbol{X} = (\tilde{\boldsymbol{x}}_1 - \boldsymbol{m}, \ldots, \tilde{\boldsymbol{x}}_n - \boldsymbol{m}) = (\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n) - (\boldsymbol{m}, \ldots, \boldsymbol{m}) = \tilde{\boldsymbol{X}} - (\boldsymbol{m}, \ldots, \boldsymbol{m}) \tag{1.61}$$

The matrix $(\boldsymbol{m}, \ldots, \boldsymbol{m})$ has n copies of $\boldsymbol{m}$ as its columns. We can write $(\boldsymbol{m}, \ldots, \boldsymbol{m})$ as the outer product $\boldsymbol{m}\mathbf{1}_n^\top$ between $\boldsymbol{m}$ and $\mathbf{1}_n$,

$$\boldsymbol{m}\mathbf{1}_n^\top = \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_d \end{pmatrix}\begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix} = \begin{pmatrix} m_1 & m_1 & \cdots & m_1 \\ m_2 & m_2 & \cdots & m_2 \\ \vdots & \vdots & & \vdots \\ m_d & m_d & \cdots & m_d \end{pmatrix} = (\boldsymbol{m}, \ldots, \boldsymbol{m}). \tag{1.62}$$

With $\boldsymbol{m} = \tilde{\boldsymbol{X}}\frac{1}{n}\mathbf{1}_n$, we thus obtain

$$(\boldsymbol{m}, \ldots, \boldsymbol{m}) = \tilde{\boldsymbol{X}}\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top \tag{1.63}$$

and hence

$$\boldsymbol{X} = \tilde{\boldsymbol{X}} - (\boldsymbol{m}, \ldots, \boldsymbol{m}) = \tilde{\boldsymbol{X}} - \tilde{\boldsymbol{X}}\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top = \tilde{\boldsymbol{X}}(\boldsymbol{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top) = \tilde{\boldsymbol{X}}\boldsymbol{H}_n, \tag{1.64}$$

as claimed in (1.58).

Multiplying $\tilde{\boldsymbol{X}}$ with $\boldsymbol{H}_n$ from the *right* subtracts the average value of each *row* of $\tilde{\boldsymbol{X}}$ from each element in said row, i.e.

$$(\tilde{\boldsymbol{X}}\boldsymbol{H}_n)_{ij} = (\tilde{\boldsymbol{X}})_{ij} - \frac{1}{n}\sum_{j=1}^{n}(\tilde{\boldsymbol{X}})_{ij}. \tag{1.65}$$

As a side note, multiplying $\boldsymbol{H}_n$ from the left with a column vector $\boldsymbol{a} = (a_1, \ldots, a_n^\top)$ would subtract the average of all a_i from each element of $\boldsymbol{a}$,

$$\boldsymbol{H}_n\boldsymbol{a} = \begin{pmatrix} a_1 - \bar{a} \\ a_2 - \bar{a} \\ \vdots \\ a_n - \bar{a} \end{pmatrix} \qquad \bar{a} = \frac{1}{n}\sum_{i=1}^{n} a_i, \tag{1.66}$$

and hence, multiplying a matrix with $\boldsymbol{H}_n$ from the *left* would subtract the average of each *column* from each column of the matrix. In brief, $\boldsymbol{H}_n$ is a projection matrix that projects vectors on the space orthogonal to $\mathbf{1}_n$ (see Section A.6). It satisfies $\boldsymbol{H}_n\boldsymbol{H}_n = \boldsymbol{H}_n$.

Scaling to unit variance

The sample covariance matrix $\hat{\boldsymbol{C}}$ for the (raw) data is

$$\hat{\boldsymbol{C}} = \frac{1}{n}\sum_{i=1}^{n}(\tilde{\boldsymbol{x}}_i - \boldsymbol{m})(\tilde{\boldsymbol{x}}_i - \boldsymbol{m})^\top \tag{1.67}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i\boldsymbol{x}_i^\top \tag{1.68}$$

$$= \frac{1}{n}\boldsymbol{X}\boldsymbol{X}^\top \tag{1.69}$$

$$= \frac{1}{n}\tilde{\boldsymbol{X}}\boldsymbol{H}_n\tilde{\boldsymbol{X}}^\top. \tag{1.70}$$

Its diagonal elements are the sample variances. Let us collect them into the diagonal matrix $\boldsymbol{D}$. We can then scale the centred data $\boldsymbol{x}_i$ to unit variance by pre-multiplying them with $\boldsymbol{D}^{-1/2}$. Note that each dimension of the data points $\boldsymbol{D}^{-1/2}\boldsymbol{x}_i$ has now unit variance, but they may still be correlated. In fact, the (sample) covariance matrix of the rescaled data $\boldsymbol{D}^{-1/2}\boldsymbol{x}_i$ is an estimate of the correlation matrix $\boldsymbol{K}$ in (1.52). Finally, we will later often use $\boldsymbol{C}$ to denote not only the covariance matrix but also the sample covariance matrix. The context will make clear which matrix is meant.

References

[1] J.L. Hintze and R.D. Nelson. "Violin Plots: A Box Plot-Density Trace Synergism". In: *The American Statistician* 52.2 (1998).

[2] T.-H. Kim and H. White. "On more robust estimation of skewness and kurtosis". In: *Finance Research Letters* 1.1 (2004), pp. 56–73.

Chapter 2

Principal component analysis

This chapter presents several equivalent views on principle component analysis (PCA). The two main themes are finding directions in the data space along which the data are maximally variable, and finding lower-dimensional yet accurate representations of the data. We assume that the data have been centred, i.e. that the sample mean has been subtracted from the data points $\boldsymbol{x}_i$, and that the corresponding random vector $\boldsymbol{x}$ has zero mean.

2.1 PCA by sequential variance maximisation

We first explain how to find the first principal component direction and then how to iteratively find the subsequent ones.

2.1.1 First principal component direction

The first principal component direction is the unit vector $\boldsymbol{w}_1$ for which the projected data $\boldsymbol{w}_1^\top \boldsymbol{x}_i$ are maximally variable, where variability is measured by the sample variance. Equivalently, we can work with the random vector $\boldsymbol{x}$ and look for the direction $\boldsymbol{w}_1$ for which the variance of $z_1 = \boldsymbol{w}_1^\top \boldsymbol{x}$ is maximal.

The variance $\mathbb{V}(z_1)$ can be expressed in terms of the covariance matrix $\boldsymbol{C}$ of $\boldsymbol{x}$,

$$\mathbb{V}(z_1) = \mathbb{V}(\boldsymbol{w}_1^\top \boldsymbol{x}) = \boldsymbol{w}_1^\top \boldsymbol{C} \boldsymbol{w}_1, \tag{2.1}$$

which follows from (1.51) with $\boldsymbol{A} = \boldsymbol{w}_1^\top$. The first principal component direction is thus the solution to the following optimisation problem:

$$\begin{aligned} \underset{\boldsymbol{w}_1}{\text{maximise}} \quad & \boldsymbol{w}_1^\top \boldsymbol{C} \boldsymbol{w}_1 \\ \text{subject to} \quad & ||\boldsymbol{w}_1|| = 1 \end{aligned} \tag{2.2}$$

The eigenvalue decomposition of $\boldsymbol{C}$ allows us to find a solution in closed form. Let

$$\boldsymbol{C} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\top, \tag{2.3}$$

where $\boldsymbol{U}$ is an orthogonal matrix and where $\boldsymbol{\Lambda}$ is diagonal with eigenvalues $\lambda_i \geq 0$ (see Section A.8). We further assume that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$. As the columns

$\boldsymbol{u}_i$ of $\boldsymbol{U}$ form an orthogonal basis, we can express $\boldsymbol{w}_1$ as

$$\boldsymbol{w}_1 = \sum_{i=1}^{d} a_i \boldsymbol{u}_i = \boldsymbol{U}\boldsymbol{a}, \tag{2.4}$$

where $\boldsymbol{a} = (a_1, \ldots, a_d)^\top$. The quadratic form $\boldsymbol{w}_1^\top \boldsymbol{C}\boldsymbol{w}_1$ can thus be written as

$$\boldsymbol{w}_1^\top \boldsymbol{C}\boldsymbol{w}_1 = \boldsymbol{a}^\top \underbrace{\boldsymbol{U}^\top \boldsymbol{U}}_{\boldsymbol{I}_d} \boldsymbol{\Lambda} \underbrace{\boldsymbol{U}^\top \boldsymbol{U}}_{\boldsymbol{I}_d} \boldsymbol{a} = \boldsymbol{a}^\top \boldsymbol{\Lambda} \boldsymbol{a} = \sum_{i=1}^{d} a_i^2 \lambda_i, \tag{2.5}$$

and the unit norm constraint on $\boldsymbol{w}_1$ becomes

$$||\boldsymbol{w}_1||^2 = \boldsymbol{w}_1^\top \boldsymbol{w}_1 = \boldsymbol{a}^\top \boldsymbol{U}^\top \boldsymbol{U}\boldsymbol{a} = \boldsymbol{a}^\top \boldsymbol{a} = \sum_{i=1}^{d} a_i^2 \stackrel{!}{=} 1 \tag{2.6}$$

An equivalent formulation of the optimisation problem in (2.2) is thus

$$\begin{aligned} \underset{a_1,\ldots,a_d}{\text{maximise}} \quad & \sum_{i=1}^{d} a_i^2 \lambda_i \\ \text{subject to} \quad & \sum_{i=1}^{d} a_i^2 = 1 \end{aligned} \tag{2.7}$$

As $\lambda_1 \geq \lambda_i, i = 2, \ldots, d$, setting a_1 to one and the remaining a_i to zero is a solution to the optimisation problem. This is the unique solution if λ_1 is the largest eigenvalue. But if, for example, $\lambda_1 = \lambda_2$, the solution is not unique any more: any a_1 and a_2 with $a_1^2 + a_2^2 = 1$ satisfy the constraint and yield the same objective. Assuming from now on that $\lambda_1 > \lambda_i,\ i = 2, \ldots, d$, the unique $\boldsymbol{w}_1$ that solves the optimisation problem in (2.2) is

$$\boldsymbol{w}_1 = \boldsymbol{U} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \boldsymbol{u}_1. \tag{2.8}$$

The corresponding value of the objective $\boldsymbol{w}_1^\top \boldsymbol{C}\boldsymbol{w}_1$ is λ_1.

The first principal component direction $\boldsymbol{w}_1$ is thus given by the eigenvector of the covariance matrix of $\boldsymbol{x}$ that has the largest eigenvalue. The random variable $z_1 = \boldsymbol{w}_1^\top \boldsymbol{x}$ is called the first principal component of $\boldsymbol{x}$.

The variance of z_1 is equal to λ_1 – the largest eigenvalue of $\boldsymbol{C}$ and the maximal value of the objective in (2.2). We say that λ_1 is the variance of $\boldsymbol{x}$ explained by the first principal component (direction). Since $\boldsymbol{x}$ is assumed centred, the expected value of z_1 is zero,

$$\mathbb{E}(z_1) = \mathbb{E}(\boldsymbol{w}_1^\top \boldsymbol{x}) = \boldsymbol{w}_1^\top \underbrace{\mathbb{E}(\boldsymbol{x})}_{0} = 0. \tag{2.9}$$

In practice, we work with the centred data points $\boldsymbol{x}_i$. The projections $\boldsymbol{w}_1^\top \boldsymbol{x}_i, i = 1, \ldots, n$ are often called the first principle components too, but also, more precisely, the first principle component scores. Collecting the centred data points into the $d \times n$ data matrix $\boldsymbol{X}$,

$$\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n), \tag{2.10}$$

the (row) vector $\boldsymbol{z}_1^\top$ with all first principle component scores is given by $\boldsymbol{w}_1^\top \boldsymbol{X}$.

2.1.2 Subsequent principal component directions

Given $\boldsymbol{w}_1$, the next principal component direction $\boldsymbol{w}_2$ is chosen so that it maximises the variance of the projection $\boldsymbol{w}_2^\top \boldsymbol{x}$ and so that it reveals something "new" in the data, i.e. something that $\boldsymbol{w}_1$ has not uncovered. This puts a constraint on $\boldsymbol{w}_2$, and in PCA, the constraint is implemented by requiring that $\boldsymbol{w}_2$ is orthogonal to $\boldsymbol{w}_1$.

The second principal component direction is hence defined as the solution to the optimisation problem:

$$\begin{aligned} \underset{\boldsymbol{w}_2}{\text{maximise}} \quad & \boldsymbol{w}_2^\top \boldsymbol{C} \boldsymbol{w}_2 \\ \text{subject to} \quad & ||\boldsymbol{w}_2|| = 1 \\ & \boldsymbol{w}_2^\top \boldsymbol{w}_1 = 0 \end{aligned} \tag{2.11}$$

As before, we decompose $\boldsymbol{C}$ as $\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$ and write $\boldsymbol{w}_2$ as $\boldsymbol{w}_2 = \boldsymbol{U}\boldsymbol{b}$. Since $\boldsymbol{w}_1 = \boldsymbol{u}_1$, $\boldsymbol{w}_2^\top \boldsymbol{w}_1$ equals

$$\boldsymbol{b}^\top \boldsymbol{U}^\top \boldsymbol{u}_1 = \boldsymbol{b}^\top \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix} = b_1 \tag{2.12}$$

and the constraint $\boldsymbol{w}_2^\top \boldsymbol{w}_1 = 0$ becomes the constraint $b_1 = 0$. The optimisation problem in (2.11) can thus be equally expressed as:

$$\begin{aligned} \underset{b_1,\ldots,b_d}{\text{maximise}} \quad & \sum_{i=1}^{d} b_i^2 \lambda_i \\ \text{subject to} \quad & \sum_{i=1}^{d} b_i^2 = 1 \\ & b_1 = 0 \end{aligned} \tag{2.13}$$

We can insert the constraint $b_1 = 0$ directly into the other equations to obtain

$$\begin{aligned} \underset{b_2,\ldots,b_d}{\text{maximise}} \quad & \sum_{i=2}^{d} b_i^2 \lambda_i \\ \text{subject to} \quad & \sum_{i=2}^{d} b_i^2 = 1 \end{aligned} \tag{2.14}$$

The optimisation problem is structurally the same as in (2.7); now we just optimise over $b_2, \ldots, b_d$. As $\lambda_2 \geq \lambda_i,\ i = 3, \ldots d$, an optimal vector $\boldsymbol{b}$ is $(0, 1, 0, \ldots, 0)^\top$ and hence

$$\boldsymbol{w}_2 = \boldsymbol{U} \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \boldsymbol{u}_2 \tag{2.15}$$

is a solution to the optimisation problem. Furthermore, the value of $\boldsymbol{w}_2^\top \boldsymbol{C}\boldsymbol{w}_2$ is λ_2. As discussed for the first principle component, this solution is unique if $\lambda_2 > \lambda_i,\ i = 3, \dots d$, which we here assume to be the case.

The second principle component direction $\boldsymbol{w}_2$ is thus given by the eigenvector of the covariance matrix of $\boldsymbol{x}$ that has the second largest eigenvalue. Analogue to the first principle component z_1, the second principle component is $z_2 = \boldsymbol{w}_2^\top \boldsymbol{x}$ with mean $\mathbb{E}(z_2) = 0$ and variance $\mathbb{V}(z_2) = \lambda_2$. The principle components are uncorrelated:

$$\begin{aligned}
\mathbb{E}\left(z_1 z_2\right) &= \mathbb{E}\left(\boldsymbol{w}_1^\top \boldsymbol{x} \boldsymbol{w}_2^\top \boldsymbol{x}\right) && (2.16)\\
&= \mathbb{E}\left(\boldsymbol{w}_1^\top \boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{w}_2\right) = \boldsymbol{w}_1^\top \mathbb{E}\left(\boldsymbol{x}\boldsymbol{x}^\top\right) \boldsymbol{w}_2 && (2.17)\\
&= \boldsymbol{w}_1^\top \boldsymbol{C}\boldsymbol{w}_2 && (2.18)\\
&= \boldsymbol{u}_1^\top \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top \boldsymbol{u}_2 && (2.19)\\
&= \begin{pmatrix} 1 & 0 & 0 & \cdots \end{pmatrix} \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_d \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix} && (2.20)\\
&= \begin{pmatrix} \lambda_1 & 0 & 0 & \cdots \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix} && (2.21)\\
&= 0. && (2.22)
\end{aligned}$$

The procedure that we used to obtain $\boldsymbol{w}_2$ given $\boldsymbol{w}_1$ can be iterated to obtain further principle component directions. Assume that we have already computed $\boldsymbol{w}_1, \dots, \boldsymbol{w}_{m-1}$. The m-th principle component direction $\boldsymbol{w}_m$ is then defined as the solution to:

$$\begin{aligned}
\underset{\boldsymbol{w}_m}{\text{maximise}} \quad & \boldsymbol{w}_m^\top \boldsymbol{C}\boldsymbol{w}_m \\
\text{subject to} \quad & ||\boldsymbol{w}_m|| = 1 \\
& \boldsymbol{w}_m^\top \boldsymbol{w}_i = 0 \quad i = 1, \dots, m-1
\end{aligned} \tag{2.23}$$

Arguing as before, the m-th principle component direction $\boldsymbol{w}_m$ is given by eigenvector $\boldsymbol{u}_m$ that corresponds to the m-th largest eigenvalue of $\boldsymbol{C}$ (assuming that there are no ties with other eigenvalues). The random variable $z_m = \boldsymbol{w}_m^\top \boldsymbol{x}$ is called the m-th principle component, its variance $\mathbb{V}(z_m)$ is λ_m, it is of zero mean (because $\boldsymbol{x}$ is zero mean), and all principle components are uncorrelated. The $\boldsymbol{w}_m^\top \boldsymbol{x}_i, i = 1, \dots, n$, are the m-th principle component scores.

The total variance of the z_m, $m = 1, \dots, k$, is said to be the variance explained by the k principle components. It equals

$$\sum_{m=1}^{k} \mathbb{V}(z_m) = \sum_{m=1}^{k} \lambda_m. \tag{2.24}$$

The variance explained by the k principle components is often reported relative to the sum of the variances of the random variables x_i that make up the random

vector $\boldsymbol{x}$. The resulting number is the "fraction of variance explained". With (1.45), the total variance of $\boldsymbol{x}$ is

$$\sum_{i=1}^{d} \mathbb{V}(x_i) = \sum_{i=1}^{d} \lambda_i, \tag{2.25}$$

so that

$$\text{fraction of variance explained} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}. \tag{2.26}$$

The fraction of variance explained is a useful number to compute for assessing how much of the variability in the data is captured by the k principle components.

2.2 PCA by simultaneous variance maximisation

We first explain the principle and then show that sequential and simultaneous variance maximisation yield the same solution.

2.2.1 Principle

In the previous section, the k principle component directions $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k$ were determined in a sequential manner, each time maximising the variance of each projection. Instead of the sequential approach, we can also determine all directions concurrently by maximising the total variance of all projections, i.e. by solving the optimisation problem:

$$\begin{aligned} \underset{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_k}{\text{maximise}} \quad & \sum_{i=1}^{k} \boldsymbol{w}_i^\top \boldsymbol{C} \boldsymbol{w}_i \\ \text{subject to} \quad & ||\boldsymbol{w}_i|| = 1 \qquad i = 1, \ldots, k \\ & \boldsymbol{w}_i^\top \boldsymbol{w}_j = 0 \qquad i \neq j \end{aligned} \tag{2.27}$$

It turns out that the optimal $\boldsymbol{w}_1, \ldots \boldsymbol{w}_k$ from the sequential approach, i.e. the eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$, are also solving the joint optimisation problem in (2.27), so that the maximal variance of all projections is $\sum_{i=1}^{k} \lambda_k$. This result may be intuitively understandable, but there is a subtle technical point: The sequential approach corresponds to solving the optimisation problem in (2.27) in a greedy manner, and greedy algorithms are generally not guaranteed to yield the optimal solution. A proof that simultaneous and sequential variance maximisation yield the same solution is given below (optional reading).

2.2.2 Sequential maximisation yields simultaneous maximisation*

As in the sequential approach, we work in the orthogonal basis provided by the eigenvectors of $\boldsymbol{C}$, i.e.

$$\boldsymbol{w}_i = \boldsymbol{U} \boldsymbol{a}_i, \tag{2.28}$$

so that we can write the optimisation problem as

$$\begin{aligned} \underset{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_k}{\text{maximise}} \quad & \sum_{i=1}^{k} \boldsymbol{a}_i^\top \boldsymbol{\Lambda} \boldsymbol{a}_i \\ \text{subject to} \quad & ||\boldsymbol{a}_i|| = 1 \qquad i = 1, \ldots, k \\ & \boldsymbol{a}_i^\top \boldsymbol{a}_j = 0 \qquad i \neq j \end{aligned} \tag{2.29}$$

We see that the k vectors $\boldsymbol{a}_i$ are required to be orthonormal. They can be extended by orthonormal vectors $\boldsymbol{a}_{k+1}, \ldots, \boldsymbol{a}_d$ so that the matrix

$$\boldsymbol{A} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k, \boldsymbol{a}_{k+1}, \ldots, \boldsymbol{a}_d) \tag{2.30}$$

is orthogonal and thus satisfies $\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}_d$. This means that the row vectors of $\boldsymbol{A}$ have norm one,

$$\sum_{j=1}^{d} (\boldsymbol{A})_{ij}^2 = 1, \tag{2.31}$$

and thus that

$$\sum_{j=1}^{k} (\boldsymbol{A})_{ij}^2 \leq 1. \tag{2.32}$$

Below, we will denote $\sum_{j=1}^{k} (\boldsymbol{A})_{ij}^2$ by b_i. Note that $\sum_{i=1}^{d} b_i = k$ since the column vectors of $\boldsymbol{A}$ have unit norm.

Since $\boldsymbol{\Lambda}$ is a diagonal matrix, the objective in (2.29) can be written as

$$\sum_{j=1}^{k} \boldsymbol{a}_j^\top \boldsymbol{\Lambda} \boldsymbol{a}_j = \sum_{j=1}^{k} \sum_{i=1}^{d} (\boldsymbol{a}_j)_i^2 \lambda_i = \sum_{j=1}^{k} \sum_{i=1}^{d} (\boldsymbol{A})_{ij}^2 \lambda_i. \tag{2.33}$$

We now show that $\sum_{i=1}^{k} \lambda_i$ is the maximal sum that can be obtained by any set of k orthogonal vectors $\boldsymbol{a}_i$. This proves our claim about the solution of the optimisation problem in (2.27). We start with re-writing $\sum_{j=1}^{k} \boldsymbol{a}_j^\top \boldsymbol{\Lambda} \boldsymbol{a}_j$ as

$$\sum_{j=1}^{k} \boldsymbol{a}_j^\top \boldsymbol{\Lambda} \boldsymbol{a}_j = \sum_{j=1}^{k} \sum_{i=1}^{d} (\boldsymbol{A})_{ij}^2 \lambda_i \tag{2.34}$$

$$= \sum_{i=1}^{d} \underbrace{\sum_{j=1}^{k} (\boldsymbol{A})_{ij}^2}_{b_i} \lambda_i \tag{2.35}$$

$$= \sum_{i=1}^{d} b_i \lambda_i \tag{2.36}$$

$$= \sum_{i=1}^{k} b_i \lambda_i + \sum_{i=k+1}^{d} b_i \lambda_i \tag{2.37}$$

For $i > k$, $\lambda_i \leq \lambda_k$, as we assume that the eigenvalues are ordered from large to small. We thus obtain an upper bound for $\sum_{j=1}^{k} \boldsymbol{a}_j^\top \boldsymbol{\Lambda} \boldsymbol{a}_j$,

$$\sum_{j=1}^{k} \boldsymbol{a}_j^\top \boldsymbol{\Lambda} \boldsymbol{a}_j = \sum_{i=1}^{k} b_i \lambda_i + \sum_{i=k+1}^{d} b_i \lambda_i \tag{2.38}$$

$$\leq \sum_{i=1}^{k} b_i \lambda_i + \lambda_k \sum_{i=k+1}^{d} b_i. \tag{2.39}$$

We now write $\sum_{i=k+1}^{d} b_i = \sum_{i=1}^{d} b_i - \sum_{i=1}^{k} b_i$ and use that $\sum_{i=1}^{d} b_i = k$, so that

$$\sum_{i=k+1}^{d} b_i = k - \sum_{i=1}^{k} b_i \tag{2.40}$$

and hence

$$\sum_{j=1}^{k} \boldsymbol{a}_j^\top \boldsymbol{\Lambda} \boldsymbol{a}_j \leq \sum_{i=1}^{k} b_i \lambda_i + k\lambda_k - \sum_{i=1}^{k} b_i \lambda_k \tag{2.41}$$

$$= \sum_{i=1}^{k} b_i (\lambda_i - \lambda_k) + k\lambda_k. \tag{2.42}$$

Since $\lambda_i - \lambda_k \geq 0$ for $i \leq k$ and $0 \leq b_i \leq 1$ we have $b_i(\lambda_i - \lambda_k) \leq (\lambda_i - \lambda_k)$ so that

$$\sum_{j=1}^{k} \boldsymbol{a}_j^\top \boldsymbol{\Lambda} \boldsymbol{a}_j \leq \sum_{i=1}^{k} (\lambda_i - \lambda_k) + k\lambda_k \tag{2.43}$$

$$= \sum_{i=1}^{k} \lambda_i - \sum_{i=1}^{k} \lambda_k + k\lambda_k \tag{2.44}$$

$$= \sum_{i=1}^{k} \lambda_i - k\lambda_k + k\lambda_k, \tag{2.45}$$

from where the desired result follows:

$$\sum_{j=1}^{k} \boldsymbol{a}_j^\top \boldsymbol{\Lambda} \boldsymbol{a}_j \leq \sum_{i=1}^{k} \lambda_i. \tag{2.46}$$

The upper bound is achieved if $\boldsymbol{a}_j$ is the j-th unit vector, i.e. if $\boldsymbol{a}_1 = (1, 0, \ldots)^\top$, $\boldsymbol{a}_2 = (0, 1, 0, \ldots)^\top, \ldots$, that is, if $\boldsymbol{w}_i = \boldsymbol{u}_j$. They are the unique solution if there are not ties in the first eigenvalues, i.e. if $\lambda_1 > \cdots > \lambda_k > \lambda_{k+1}$.

2.3 PCA by minimisation of approximation error

We first explain the principle and then show the equivalence to PCA by variance maximisation.

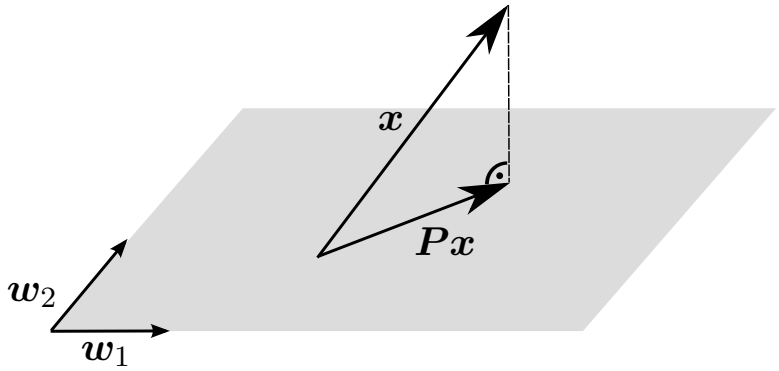


Figure 2.1: Orthogonal projection of $\boldsymbol{x}$ onto the subspace spanned by the two orthonormal vectors $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. The projection $\boldsymbol{Px}$ can be written as a linear combination of $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$, and the residual $\boldsymbol{x}-\boldsymbol{Px}$ is orthogonal to both vectors.

2.3.1 Principle

A set of k orthonormal vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k$ of dimension d spans a k-dimensional subspace of $\mathbb{R}^d$ denoted by $\text{span}(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k)$, see Section A.5. Moreover, the matrix $\boldsymbol{P}$,

$$\boldsymbol{P} = \sum_{i=1}^{k} \boldsymbol{w}_i \boldsymbol{w}_i^\top = \boldsymbol{W}_k \boldsymbol{W}_k^\top, \qquad \boldsymbol{W}_k = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k), \tag{2.47}$$

projects any vector onto said subspace. This means that we can decompose our data points $\boldsymbol{x}_i$ into elements $\hat{\boldsymbol{x}}_i = \boldsymbol{P}\boldsymbol{x}_i$ that belong to $\text{span}(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k)$ and "residual" vectors orthogonal to it (see Figure 2.1 and Section A.6). The projections $\hat{\boldsymbol{x}}_i$ are lower dimensional approximations of the $\boldsymbol{x}_i$ that can be represented by the k coordinates $\boldsymbol{w}_1^\top \boldsymbol{x}_i, \ldots, \boldsymbol{w}_k^\top \boldsymbol{x}_i$. Equivalently, the random vector $\boldsymbol{x}$ can be approximated by $\hat{\boldsymbol{x}} = \boldsymbol{P}\boldsymbol{x} = \sum_{i=1}^{k} \boldsymbol{w}_i \boldsymbol{w}_i^\top \boldsymbol{x}$.

We now ask which subspace yields the approximations with the smallest error on average? Or equivalently, which subspace yields the smallest expected approximation error? The question can be formulated as the optimisation problem:

$$\begin{aligned} \underset{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_k}{\text{minimise}} \quad & \mathbb{E}\left|\left|\boldsymbol{x} - \sum_{i=1}^{k} \boldsymbol{w}_i \boldsymbol{w}_i^\top \boldsymbol{x}\right|\right|^2 & \\ \text{subject to} \quad & ||\boldsymbol{w}_i|| = 1 & i = 1, \ldots, k \\ & \boldsymbol{w}_i^\top \boldsymbol{w}_j = 0 & i \neq j \end{aligned} \tag{2.48}$$

We show below that the optimisation problem is equivalent to the optimisation problem in (2.27), so that the optimal $\boldsymbol{w}_i$ are the first k eigenvectors $\boldsymbol{u}_i$ of the covariance matrix of $\boldsymbol{x}$, where "first k eigenvectors" means the eigenvectors with the k largest eigenvalues. For this to make sense, it is assumed that the k-th eigenvalue is larger than the $(k+1)$-th eigenvalue.

In other words, the optimal k-dimensional subspace is spanned by $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$, the optimal projection matrix is $\boldsymbol{P} = \boldsymbol{U}_k \boldsymbol{U}_k^\top$, and the optimal lower dimensional

representation of $\boldsymbol{x}$ is $\hat{\boldsymbol{x}} = \boldsymbol{P}\boldsymbol{x}$. Since

$$\hat{\boldsymbol{x}} = \boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{x} = \sum_{i=1}^{k} \boldsymbol{u}_i \boldsymbol{u}_i^\top \boldsymbol{x} = \sum_{i=1}^{k} \boldsymbol{u}_i z_i \tag{2.49}$$

the i-th principal component $z_i = \boldsymbol{u}_i^\top \boldsymbol{x}$ is the i-th coordinate of $\hat{\boldsymbol{x}}$ when represented in the basis $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$.

2.3.2 Equivalence to PCA by variance maximisation

To prove the equivalence of (2.27) and (2.48), we first write $\sum_{i=1}^{k} \boldsymbol{w}_i \boldsymbol{w}_i^\top \boldsymbol{x}$ more compactly as $\boldsymbol{W}_k \boldsymbol{W}_k^\top \boldsymbol{x}$ and expand the norm of the approximation error,

$$\begin{aligned}
||\boldsymbol{x} - \boldsymbol{W}_k \boldsymbol{W}_k^\top \boldsymbol{x}||^2 &= (\boldsymbol{x} - \boldsymbol{W}_k \boldsymbol{W}_k^\top \boldsymbol{x})^\top (\boldsymbol{x} - \boldsymbol{W}_k \boldsymbol{W}_k^\top \boldsymbol{x}) && (2.50)\\
&= \boldsymbol{x}^\top \boldsymbol{x} - 2\boldsymbol{x}^\top \boldsymbol{W}_k \boldsymbol{W}_k^\top \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{W}_k \underbrace{\boldsymbol{W}_k^\top \boldsymbol{W}_k}_{\boldsymbol{I}_k} \boldsymbol{W}_k^\top \boldsymbol{x} && (2.51)\\
&= \boldsymbol{x}^\top \boldsymbol{x} - \boldsymbol{x}^\top \boldsymbol{W}_k \boldsymbol{W}_k^\top \boldsymbol{x} && (2.52)
\end{aligned}$$

Using again that $\boldsymbol{W}_k \boldsymbol{W}_k^\top = \sum_{i=1}^{k} \boldsymbol{w}_i \boldsymbol{w}_i^\top$, we obtain

$$\begin{aligned}
||\boldsymbol{x} - \boldsymbol{W}_k \boldsymbol{W}_k^\top \boldsymbol{x}||^2 &= \boldsymbol{x}^\top \boldsymbol{x} - \boldsymbol{x}^\top \left(\sum_{i=1}^{k} \boldsymbol{w}_i \boldsymbol{w}_i^\top \right) \boldsymbol{x} && (2.53)\\
&= \boldsymbol{x}^\top \boldsymbol{x} - \sum_{i=1}^{k} (\boldsymbol{x}^\top \boldsymbol{w}_i)(\boldsymbol{w}_i^\top \boldsymbol{x}) && (2.54)\\
&= \boldsymbol{x}^\top \boldsymbol{x} - \sum_{i=1}^{k} \boldsymbol{w}_i^\top \boldsymbol{x} \boldsymbol{x}^\top \boldsymbol{w}_i && (2.55)
\end{aligned}$$

and the expected approximation error is

$$\begin{aligned}
\mathbb{E}\,||\boldsymbol{x} - \boldsymbol{W}_k \boldsymbol{W}_k^\top \boldsymbol{x}||^2 &= \mathbb{E}(\boldsymbol{x}^\top \boldsymbol{x}) - \mathbb{E}\left(\sum_{i=1}^{k} \boldsymbol{w}_i^\top \boldsymbol{x} \boldsymbol{x}^\top \boldsymbol{w}_i \right) && (2.56)\\
&= \mathbb{E}(\boldsymbol{x}^\top \boldsymbol{x}) - \sum_{i=1}^{k} \boldsymbol{w}_i^\top \,\mathbb{E}(\boldsymbol{x}\boldsymbol{x}^\top) \boldsymbol{w}_i && (2.57)
\end{aligned}$$

due to the linearity of the expectation. As we assume that the expected value $\mathbb{E}(\boldsymbol{x})$ is zero, due to the centring, we have $\boldsymbol{C} = \mathbb{E}(\boldsymbol{x}\boldsymbol{x}^\top)$ and

$$\mathbb{E}\,||\boldsymbol{x} - \boldsymbol{W}_k \boldsymbol{W}_k^\top \boldsymbol{x}||^2 = \mathbb{E}(\boldsymbol{x}^\top \boldsymbol{x}) - \sum_{i=1}^{k} \boldsymbol{w}_i^\top \boldsymbol{C} \boldsymbol{w}_i. \tag{2.58}$$

Since $\mathbb{E}(\boldsymbol{x}^\top \boldsymbol{x})$ is a constant, minimising the expected approximation error is equivalent to maximising $\sum_{i=1}^{k} \boldsymbol{w}_i^\top \boldsymbol{C} \boldsymbol{w}_i$, which is the total variance of the projections $\boldsymbol{w}_i^\top \boldsymbol{x}$ and the objective in (2.27). The constraints in (2.48) and (2.27) are also the same so that the two optimisation problems are equivalent.

From Section 2.1.2, and (1.45), we know that $\mathbb{E}(\boldsymbol{x}^\top \boldsymbol{x}) = \sum_{i=1}^{d} \lambda_i$ so that the smallest expected approximation error when orthogonally projecting $\boldsymbol{x}$ onto a k dimensional subspace is

$$\mathbb{E}\,||\boldsymbol{x} - \boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{x}||^2 = \sum_{i=1}^{d} \lambda_i - \sum_{i=1}^{k} \lambda_i = \sum_{i=k+1}^{d} \lambda_i, \tag{2.59}$$

which is the sum of the eigenvalues whose eigenvectors were omitted from the optimal subspace. Computing the relative approximation error highlights the connection between minimising approximation error and maximising the variance explained by the principal components,

$$\frac{\mathbb{E}\,||\boldsymbol{x} - \boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{x}||^2}{\mathbb{E}(\boldsymbol{x}^\top \boldsymbol{x})} = 1 - \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} = 1 - \text{fraction of variance explained.} \tag{2.60}$$

The fraction of variance explained by a principle component (direction) thus equals the relative reduction in approximation error that is achieved by including it into the subspace.

2.4 PCA by low rank matrix approximation

This section uses the theory of low rank matrix approximation to provide a complementary view on the PCA principles of variance maximisation and minimisation of approximation error.

2.4.1 Approximating the data matrix

We will here see that the principle component directions and scores together yield the best low rank approximation of the data matrix, and that the PC directions and scores can be computed by a singular value decomposition (SVD).

Let $\boldsymbol{X}$ be the $d \times n$ data matrix that contains the centred data points $\boldsymbol{x}_i$ in its columns,

$$\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n). \tag{2.61}$$

We can express $\boldsymbol{X}$ via its singular value decomposition as

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top. \tag{2.62}$$

The $d \times d$ matrix $\boldsymbol{U}$ and the $n \times n$ matrix $\boldsymbol{V}$ are orthonormal with the vectors $\boldsymbol{u}_i \in \mathbb{R}^d$ and $\boldsymbol{v}_i \in \mathbb{R}^n$ in their columns. The $\boldsymbol{u}_i$ are called the left singular vectors while the $\boldsymbol{v}_i$ are called the right singular vectors. The matrix $\boldsymbol{S}$ is $d \times n$ and zero everywhere but in the first r diagonal elements,

$$\boldsymbol{S} = \left(\begin{array}{ccc|c} s_1 & & & \\ & \ddots & & \mathbf{0} \\ & & s_r & \\ \hline & \mathbf{0} & & \mathbf{0} \end{array}\right). \tag{2.63}$$

The s_i are the singular values. They are non-negative and assumed ordered from large to small. The number $r \leq \min(d, n)$ is the rank of $\boldsymbol{X}$. The matrix $\boldsymbol{X}$ can further be written as

$$\boldsymbol{X} = \sum_{i=1}^{r} s_i \boldsymbol{u}_i \boldsymbol{v}_i^\top. \tag{2.64}$$

Section A.7 provides further background on the SVD.

Assume we would like to approximate $\boldsymbol{X}$ by a matrix $\hat{\boldsymbol{X}}$ of rank $k < r$. Judging the accuracy of the approximation by the sum of squared differences in the individual matrix elements, we can determine $\hat{\boldsymbol{X}}$ by solving the optimisation problem

$$\begin{aligned} \underset{\boldsymbol{M}}{\text{minimise}} \quad & \sum_{ij} ((\boldsymbol{X})_{ij} - (\boldsymbol{M})_{ij})^2 \\ \text{subject to} \quad & \text{rank}(\boldsymbol{M}) = k \end{aligned} \tag{2.65}$$

The sum of squared differences $\sum_{ij}((\boldsymbol{X})_{ij} - (\boldsymbol{M})_{ij})^2$ is called the Frobenius norm between $\boldsymbol{X}$ and $\boldsymbol{M}$ and typically denoted by $||\boldsymbol{X} - \boldsymbol{M}||_F$.

It is known from linear algebra that the optimal low rank approximation is given by $\hat{\boldsymbol{X}}$,

$$\hat{\boldsymbol{X}} = \sum_{i=1}^{k} s_i \boldsymbol{u}_i \boldsymbol{v}_i^\top, \tag{2.66}$$

and that the corresponding approximation error is

$$||\boldsymbol{X} - \hat{\boldsymbol{X}}||_F = \sum_{i=k+1}^{r} s_i^2, \tag{2.67}$$

see (A.61) and (A.63) in Section A.10. The solution to the optimisation problem is thus rather simple: We just keep the first k terms in (2.64).

How does this relate to principal component analysis? It turns out that

- the left singular vectors $\boldsymbol{u}_i$ are the eigenvectors of the (estimated) covariance matrix and hence equal to the principal component directions,
- the squared singular values s_i^2 are related to the eigenvalues λ_i of the covariance matrix by
$$\lambda_i = \frac{s_i^2}{n}, \tag{2.68}$$
- and that the principal component scores $\boldsymbol{z}_i^\top = \boldsymbol{u}_i^\top \boldsymbol{X}$ for principal component direction i are equal to the i-th right singular vector after scaling,
$$\boldsymbol{z}_i^\top = s_i \boldsymbol{v}_i^\top. \tag{2.69}$$

We can thus write the approximate data matrix $\hat{\boldsymbol{X}}$ in (2.66) as

$$\hat{\boldsymbol{X}} = \sum_{i=1}^{k} \boldsymbol{u}_i \boldsymbol{z}_i^\top, \tag{2.70}$$

which underlines how the k principal component directions $\boldsymbol{u}_i$ and corresponding principal component scores $\boldsymbol{z}_i$ together approximately represent the data $\boldsymbol{X}$.

The stated connections can be seen as follows: As we assume that the data points are centred, an estimate of the covariance matrix is given by the sample covariance matrix,

$$\boldsymbol{C} \approx \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^\top = \frac{1}{n}\boldsymbol{X}\boldsymbol{X}^\top. \tag{2.71}$$

Using $\boldsymbol{C}$ to denote both the covariance and the sample covariance matrix, we have

$$\boldsymbol{C} = \frac{1}{n}\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top(\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^\top) = \boldsymbol{U}\left(\frac{1}{n}\boldsymbol{S}\boldsymbol{S}\right)\boldsymbol{U}^\top \tag{2.72}$$

so that the eigenvectors of $\boldsymbol{C}$ are the left singular vectors $\boldsymbol{u}_i$ of the data matrix $\boldsymbol{X}$ with eigenvalues λ_i as in (2.68). The i-th principle component scores were defined as the projections $\boldsymbol{w}_i^\top \boldsymbol{x}_j$ of the data points $\boldsymbol{x}_j$ onto the i-th principle component direction $\boldsymbol{w}_i$. Collecting all scores into the $1 \times n$ row vector $\boldsymbol{z}_i^\top$, we have

$$\boldsymbol{z}_i^\top = \boldsymbol{w}_i^\top \boldsymbol{X} = \boldsymbol{u}_i^\top \boldsymbol{X} = \boldsymbol{u}_i^\top \boldsymbol{U}\boldsymbol{S}\boldsymbol{V} = \boldsymbol{u}_i^\top \sum_{j=1}^{r} \boldsymbol{u}_j s_j \boldsymbol{v}_j^\top = s_i \boldsymbol{v}_i^\top. \tag{2.73}$$

which means that the i-th principle component scores are given by the i-th right singular vector when multiplied with its singular value s_i as claimed in (2.69).

2.4.2 Approximating the sample covariance matrix

Approximating the data matrix with a matrix of lower rank yielded the first k principle component directions and scores. We here show that the first principle component directions can be also be obtained by a low rank approximation of the sample covariance matrix. This provides a complementary view to PCA, in that finding directions in the data space with maximal variance is also maximally preserving the variance structure of the original data. This approach does, however, not directly yield principle component scores.

The optimisation problem that we aim to solve is:

$$\begin{aligned} \underset{\boldsymbol{M}}{\text{minimise}} \quad & ||\boldsymbol{C} - \boldsymbol{M}||_F \\ \text{subject to} \quad & \text{rank}(\boldsymbol{M}) = k \\ & \boldsymbol{M}^\top = \boldsymbol{M} \end{aligned} \tag{2.74}$$

Much like the first k components of the SVD are solving the optimisation problem in (2.65), results from linear algebra tell us that the optimal low rank approximation of $\boldsymbol{C}$ is given by the first k components of its eigenvalue decomposition $\boldsymbol{U}\Lambda\boldsymbol{U}^\top$, i.e. by $\sum_{i=1}^{k} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\top$, see (A.64) in Section A.10.

2.4.3 Approximating the Gram matrix

The Gram matrix is defined as the $n \times n$ matrix $\boldsymbol{G}$,

$$\boldsymbol{G} = \boldsymbol{X}^\top \boldsymbol{X}. \tag{2.75}$$

Its (ij)-th element $(\boldsymbol{G})_{ij}$ is the inner product between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. The matrix is positive semidefinite, i.e. its eigenvalues are non-negative. It is here shown that

the first principle component scores provide an optimal low rank approximation of $\boldsymbol{G}$. Hence, finding coordinates that minimise the average approximation error of the data points $\boldsymbol{x}_i$ is also maximally preserving the inner product structure between them.

With the singular value decomposition of $\boldsymbol{X}$ in (2.62), the Gram matrix has the following eigenvalue decomposition

$$\boldsymbol{G} = (\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top)^\top(\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top) = (\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^\top)(\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top) = \boldsymbol{V}\boldsymbol{S}\boldsymbol{S}\boldsymbol{V}^\top = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{V}^\top \tag{2.76}$$

i.e. its eigenvectors are the right-singular vectors $\boldsymbol{v}_i$ of $\boldsymbol{X}$, and the diagonal matrix $\boldsymbol{\Sigma} = \boldsymbol{S}\boldsymbol{S}$ contains the eigenvalues s_i^2 ordered from large to small.

Like for the sample covariance matrix, we can determine the best rank k approximation of the Gram matrix $\boldsymbol{G}$. It is given by $\hat{\boldsymbol{G}}$,

$$\hat{\boldsymbol{G}} = \sum_{i=1}^{k} \boldsymbol{v}_i s_i^2 \boldsymbol{v}_i^\top. \tag{2.77}$$

In (2.69), we have seen that $\boldsymbol{z}_i = s_i\boldsymbol{v}_i$ is the column vector with all i-th principle component scores. We thus have

$$\hat{\boldsymbol{G}} = \sum_{i=1}^{k} \boldsymbol{z}_i \boldsymbol{z}_i^\top, \tag{2.78}$$

which shows that the k principle scores together maximally preserve the inner product structure of the data.

Denote by $\boldsymbol{\Sigma}_k$ the diagonal matrix with the top k eigenvalues of $\boldsymbol{G}$, and by $\boldsymbol{V}_k$,

$$\boldsymbol{V}_k = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) \tag{2.79}$$

the matrix with the corresponding eigenvectors. The $k \times n$ matrix with the principle component scores as its rows is then

$$\boldsymbol{Z} = \sqrt{\boldsymbol{\Sigma}_k}\boldsymbol{V}_k^\top. \tag{2.80}$$

We have the square root because the singular values s_i are the square root of the eigenvalues of $\boldsymbol{G}$. Hence, we can compute the principle component scores directly from the Gram matrix of the centred data, without first computing the principle component directions. This can be done without knowing $\boldsymbol{X}$ as long as $\boldsymbol{G}$ is available.

Chapter 3

Dimensionality reduction

Dimensionality reduction is about representing the data in a lower dimensional space in such a way that certain properties of the data are preserved as much as possible. Dimensionality reduction can be used to visualise high-dimensional data if the plane is chosen as the lower dimensional space. Taking principal component analysis (PCA) as starting point, several nonlinear dimensionality reduction methods are presented.

3.1 Dimensionality reduction by PCA

We can represent d-dimensional data by their first k principle components, or more precisely, their first k principle component scores. The principle components can be computed when the data are given in form of data vectors, and, importantly, also when given in form of inner products or distances between them.

3.1.1 From data points

Denote the uncentred data by $\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n$ and the corresponding data matrix by $\tilde{\boldsymbol{X}}$,

$$\tilde{\boldsymbol{X}} = (\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_n). \tag{3.1}$$

We first centre the data and form the matrix $\boldsymbol{X}$,

$$\boldsymbol{X} = \tilde{\boldsymbol{X}}\boldsymbol{H}_n \qquad\qquad \boldsymbol{H}_n = \boldsymbol{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top, \tag{3.2}$$

where $\boldsymbol{H}_n$ is the centring matrix from (1.58). Depending on the application, we may want to further process the data, e.g. by some form of standardisation that was introduced in Section 1.3.2.

We can now compute the principal components via an eigenvalue decomposition of the covariance matrix $\boldsymbol{C}$,

$$\boldsymbol{C} = \frac{1}{n}\boldsymbol{X}\boldsymbol{X}^\top. \tag{3.3}$$

Denoting the matrix with the top k eigenvectors $\boldsymbol{u}_i$ by $\boldsymbol{U}_k$,

$$\boldsymbol{U}_k = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k), \tag{3.4}$$

the matrix with the principle component (PC) scores is

$$\boldsymbol{Z} = \boldsymbol{U}_k^\top \boldsymbol{X}. \tag{3.5}$$

While $\boldsymbol{X}$ is $d \times n$, $\boldsymbol{Z}$ is $k \times n$. The column vectors of $\boldsymbol{Z}$ have dimension $k \leq d$ and form a lower dimensional representation of the data.

In dimensionality reduction, we are mostly interested in the PC scores, rather than the PC directions. We can thus bypass the computation of the PC directions and compute the PC scores directly from the Gram matrix $\boldsymbol{G}$ introduced in (2.75),

$$\boldsymbol{G} = \boldsymbol{X}^\top \boldsymbol{X}. \tag{3.6}$$

With (2.80), the $k \times n$ matrix $\boldsymbol{Z}$ in (3.5) equals

$$\boldsymbol{Z} = \sqrt{\boldsymbol{\Sigma}_k} \boldsymbol{V}_k^\top, \tag{3.7}$$

where the diagonal $k \times k$ matrix $\boldsymbol{\Sigma}_k$ contains the top k eigenvalues of $\boldsymbol{G}$ ordered from large to small, and the $n \times k$ matrix $\boldsymbol{V}_k$ contains the corresponding eigenvectors.

3.1.2 From inner products

The elements $(\boldsymbol{G})_{ij}$ of the Gram matrix $\boldsymbol{G} = \boldsymbol{X}^\top \boldsymbol{X}$ are the inner products between the centred data points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$,

$$(\boldsymbol{G})_{ij} = \boldsymbol{x}_i^\top \boldsymbol{x}_j = (\tilde{\boldsymbol{x}}_i - \boldsymbol{m})^\top (\tilde{\boldsymbol{x}}_j - \boldsymbol{m}). \tag{3.8}$$

Since we can compute the principle component scores (but not the directions) by an eigenvalue decomposition of the Gram matrix, we can do dimensionality reduction without actually having seen the data points $\boldsymbol{x}_i$. Knowing their inner products is enough.

But what should we do if we are only given the inner products between the original data points and not between the centred data points? That is, what should we do if we are only given the matrix $\tilde{\boldsymbol{G}}$,

$$\tilde{\boldsymbol{G}} = \tilde{\boldsymbol{X}}^\top \tilde{\boldsymbol{X}}, \tag{3.9}$$

and not $\boldsymbol{G}$?

It turns out that we can compute $\boldsymbol{G}$ from $\tilde{\boldsymbol{G}}$. With $\boldsymbol{X} = \tilde{\boldsymbol{X}} \boldsymbol{H}_n$, where $\boldsymbol{H}_n$ is the centring matrix, we have

$$\boldsymbol{G} = \boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{H}_n^\top \tilde{\boldsymbol{X}}^\top \tilde{\boldsymbol{X}} \boldsymbol{H}_n = \boldsymbol{H}_n \tilde{\boldsymbol{X}}^\top \tilde{\boldsymbol{X}} \boldsymbol{H}_n = \boldsymbol{H}_n \tilde{\boldsymbol{G}} \boldsymbol{H}_n, \tag{3.10}$$

where we have used that $\boldsymbol{H}_n$ is a symmetric matrix. This operation is called double centring: Multiplying $\tilde{\boldsymbol{G}}$ with $\boldsymbol{H}_n$ from the right makes all rows have zero average while multiplying it from the left makes all columns have a zero average.

Since inner products can be used to measure the similarity between data points, matrices like $\tilde{\boldsymbol{G}}$ and $\boldsymbol{G}$ are sometimes called similarity matrices. We can thus say that we can do dimensionality reduction by PCA given a similarity matrix (with inner products) only.

3.1.3 From distances

We here show that we can exactly recover the PC scores if we are only given the squared distances δ_{ij}^2 between the data points $\tilde{\boldsymbol{x}}_i$ and $\tilde{\boldsymbol{x}}_j$,

$$\delta_{ij}^2 = ||\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j||^2 = (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j)^\top (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j). \tag{3.11}$$

The matrix $\boldsymbol{\Delta}$ with elements δ_{ij}^2 is called a distance matrix. (Matrices with non-squared δ_{ij} are also called distance matrices. There is some ambiguity in the terminology.)

The trick is to recover the Gram matrix $\boldsymbol{G}$ in (2.75) from the distance matrix $\boldsymbol{\Delta}$: First, we note that the δ_{ij}^2 equal the squared distances between the centred data points $\boldsymbol{x}_i$,

$$\delta_{ij}^2 = ||(\tilde{\boldsymbol{x}}_i - \boldsymbol{m}) - (\tilde{\boldsymbol{x}} - \boldsymbol{m})||^2 = ||\boldsymbol{x}_i - \boldsymbol{x}_j||^2 = (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top (\boldsymbol{x}_i - \boldsymbol{x}_j). \tag{3.12}$$

Multiplying out yields

$$\delta_{ij}^2 = ||\boldsymbol{x}_i||^2 + ||\boldsymbol{x}_j||^2 - 2\boldsymbol{x}_i^\top \boldsymbol{x}_j. \tag{3.13}$$

Importantly the first term $||\boldsymbol{x}_i||^2$ is constant along row i of the matrix $\boldsymbol{\Delta}$. We can thus eliminate it by multiplying $\boldsymbol{\Delta}$ with the centring matrix $\boldsymbol{H}_n$ from the right. This is because

$$(\boldsymbol{\Delta}\boldsymbol{H}_n)_{ij} = (\boldsymbol{\Delta})_{ij} - \frac{1}{n}\sum_{j=1}^{n}(\boldsymbol{\Delta})_{ij}, \tag{3.14}$$

as, for example, in (1.65). In more detail, let us compute

$$\frac{1}{n}\sum_{j=1}^{n}(\boldsymbol{\Delta})_{ij} = \frac{1}{n}\sum_{j=1}^{n}\delta_{ij}^2 \tag{3.15}$$

$$= ||\boldsymbol{x}_i||^2 + \frac{1}{n}\sum_{j=1}^{n}||\boldsymbol{x}_j||^2 - 2\frac{1}{n}\sum_{j=1}^{n}\boldsymbol{x}_i^\top \boldsymbol{x}_j \tag{3.16}$$

which equals

$$\frac{1}{n}\sum_{j=1}^{n}(\boldsymbol{\Delta})_{ij} = ||\boldsymbol{x}_i||^2 + \frac{1}{n}\sum_{j=1}^{n}||\boldsymbol{x}_j||^2 - 2\boldsymbol{x}_i^\top \Big(\underbrace{\frac{1}{n}\sum_{j=1}^{n}\boldsymbol{x}_j}_{0}\Big) \tag{3.17}$$

$$= ||\boldsymbol{x}_i||^2 + \frac{1}{n}\sum_{j=1}^{n}||\boldsymbol{x}_j||^2, \tag{3.18}$$

because the $\boldsymbol{x}_j$ are centred and hence $\sum_j \boldsymbol{x}_j = 0$. We thus find that

$$(\boldsymbol{\Delta}\boldsymbol{H}_n)_{ij} = ||\boldsymbol{x}_i||^2 + ||\boldsymbol{x}_j||^2 - 2\boldsymbol{x}_i^\top \boldsymbol{x}_j - ||\boldsymbol{x}_i||^2 - \frac{1}{n}\sum_{j=1}^{n}||\boldsymbol{x}_j||^2 \tag{3.19}$$

$$= ||\boldsymbol{x}_j||^2 - 2\boldsymbol{x}_i^\top \boldsymbol{x}_j - \frac{1}{n}\sum_{j=1}^{n}||\boldsymbol{x}_j||^2. \tag{3.20}$$

Now, the terms $||\boldsymbol{x}_j||^2$ and $1/n \sum_{j=1}^{n} ||\boldsymbol{x}_j||^2$ are constant along column j of the matrix $\boldsymbol{\Delta}\boldsymbol{H}_n$. We can thus eliminate them by multiplying $\boldsymbol{\Delta}\boldsymbol{H}_n$ with $\boldsymbol{H}_n$ from the left. Calculations as above show that

$$(\boldsymbol{H}_n\boldsymbol{\Delta}\boldsymbol{H}_n)_{ij} = (\boldsymbol{\Delta}\boldsymbol{H}_n)_{ij} - \frac{1}{n}\sum_i (\boldsymbol{\Delta}\boldsymbol{H}_n)_{ij} = -2\boldsymbol{x}_i^\top \boldsymbol{x}_j. \tag{3.21}$$

We thus have $\boldsymbol{H}_n\boldsymbol{\Delta}\boldsymbol{H}_n = -2\boldsymbol{G}$, and hence obtain the desired result,

$$\boldsymbol{G} = -\frac{1}{2}\boldsymbol{H}_n\boldsymbol{\Delta}\boldsymbol{H}_n. \tag{3.22}$$

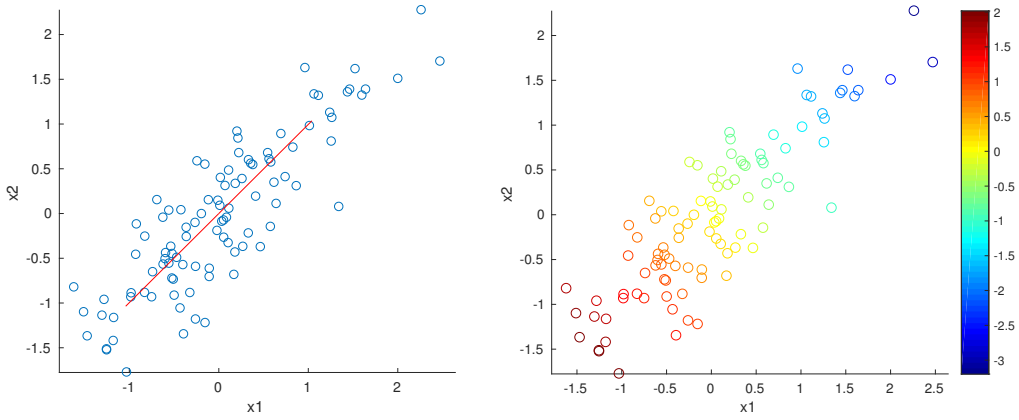
In the previous section, we double centred the similarity matrix $\tilde{\boldsymbol{G}}$ to obtain $\boldsymbol{G}$. Here, we double centre the distance matrix $\boldsymbol{\Delta}$, and swap the signs to convert distances to similarities. Once $\boldsymbol{G}$ is available, we compute the principle component scores as before via an eigenvalue decomposition, see the previous section or Equations (2.76) and (2.80).

3.1.4 Example

Figure 3.1(a) shows data that can be well represented by one principal component. The data vary mostly along the diagonal and projecting them onto the first principal component (red line) captures most of the variability. In figure 3.1(b) we colour-code each data point by its principal component score. The scores are the coordinates of the data with respect to the basis given by the principal component direction. We can see that there is a good correspondence between the value of the scores and the location of the data points. The scores change smoothly from large to small as we move along the diagonal: they faithfully represent the data and capture their structure well.

The data in Figure 3.2, on the other hand, are not as well represented by the first principal component. The first principal component direction captures the direction of maximal variance but the data are treated as a cloud of points and the scores roughly indicate the location of the data along the y-axis but not their position on the circle. The principal component scores do not capture the circular structure of the data.

Why is PCA doing better for data as in Figure 3.1 than for data as in Figure 3.2? This can be understood by considering that PCA projects the data onto a lower-dimensional subspace (Section 2.3). Subspaces are closed under addition and multiplication, which means that any point on a line going through two points from the subspace is also included in the subspace (see e.g. Section A.5). For the data in Figure 3.2, however, there are gaps of empty space between two data points that are unlikely to be filled even if we had more data. Such kind of data are said to lie on a manifold, and lines between two points on a manifold may not be part of the manifold (see, for example, Chapter 16 of Izenman (2008) or Chapter 1 of Lee and Verleysen (2007)). If the data are part of a subspace, it is reasonable to judge the distance between two points by the length of the straight line connecting them, like in PCA, but if the data are on a manifold, straight lines are a poor measure of their distance. In the same vein, the linear projections that are used to compute the principal component scores do not take the manifold structure of the data into account.



(a) Principle component direction (b) Principle component scores

Figure 3.1: Dimensionality reduction by principal component analysis. (a) The red line shows the direction of the first PC direction. (b) The colours indicate the value of the PC score assigned to each data point.

3.2 Dimensionality reduction by kernel PCA

Principal component analysis uses linear projections to compute the lower dimensional representation of the data. We here discuss kernel PCA where the projections are typically nonlinear.

3.2.1 Idea

The principle components represent the data so that the variance is maximally preserved. Assume that we expand the dimensionality of the $\boldsymbol{x}_i$ by transforming them to features $\phi(\boldsymbol{x}_i)$, for example,

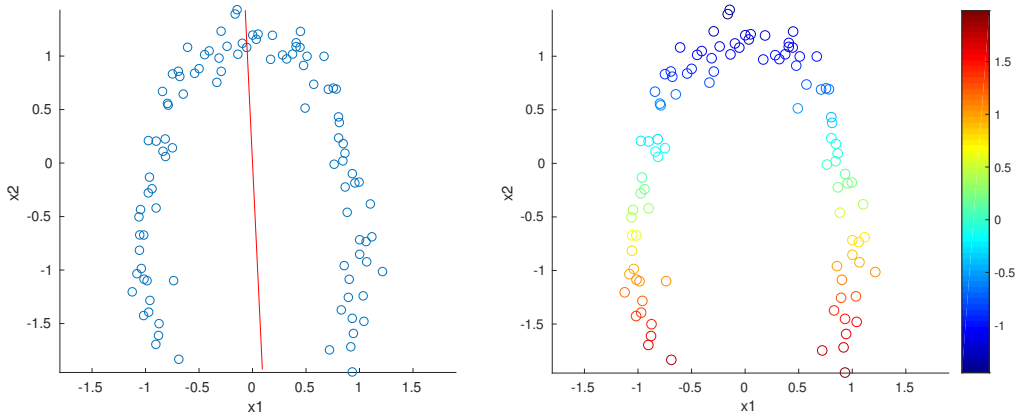
$$\phi(\boldsymbol{x}) = (x_1, \cdots, x_d, x_1x_2, \cdots, x_1x_d, \cdots, x_dx_d)^\top, \tag{3.23}$$

where $\boldsymbol{x} = (x_1, \cdots, x_d)^\top$. The much higher dimensionality of the $\boldsymbol{\phi}_i = \phi(\boldsymbol{x}_i)$ does not matter as long as we only compute k principal components from them.

Importantly, the k principal components maximally preserve the variance of the $\boldsymbol{\phi}_i$ that contains much more information about the data than the variance of the $\boldsymbol{x}_i$. The covariance matrix for the particular $\phi(\boldsymbol{x})$ above, for example, contains terms like $\mathbb{E}(x_1x_2x_3^2)$ that measure non-linear correlations between the different dimensions of the data. Similarly, the principal components best approximate the $\boldsymbol{\phi}_i$ which is much harder than approximating the $\boldsymbol{x}_i$, so that the principal components computed from the $\boldsymbol{\phi}_i$ must capture more information about the data than the components computed from the $\boldsymbol{x}_i$.

Hence, we can power up PCA dimensionality reduction by choosing a transformation ϕ that maps the data points $\boldsymbol{x}_i$ to $\boldsymbol{\phi}_i = \phi(\boldsymbol{x}_i)$, and then computing the principle component scores from the new "data matrix" $\boldsymbol{\Phi}$,

$$\boldsymbol{\Phi} = (\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_n), \tag{3.24}$$



(a) Principle component direction (b) Principle component scores

Figure 3.2: Dimensionality reduction by principal component analysis. (a) The red line shows the direction of the first PC direction. (b) The colours indicate the value of the PC score assigned to each data point.

rather than from the original data matrix $\boldsymbol{X}$. We call this approach dimensionality reduction by nonlinear PCA. (Note that "nonlinear PCA" sometimes refers to other kinds of methods too.)

3.2.2 Kernel trick

Since we can compute the principal components scores from the Gram matrix of $\boldsymbol{\Phi}$, we actually do not need to know the individual $\boldsymbol{\phi}_i$, but only the inner products $\boldsymbol{\phi}_i^\top \boldsymbol{\phi}_j = \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)$.

The theory of reproducing kernel Hilbert spaces tells us that for some functions ϕ, the inner product can be computed as

$$\phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j), \tag{3.25}$$

where $k(\boldsymbol{x}, \boldsymbol{x}')$ is called the kernel function (see, e.g. Schölkopf and Smola, 2002). This means that for some functions ϕ, we actually do not need to know the transformed data points $\boldsymbol{\phi}_i$ to compute the inner product between them, it is enough to know the kernel $k(\boldsymbol{x}, \boldsymbol{x}')$. This is called the "kernel trick" and can be used to compute the (uncentred) Gram matrix of $\boldsymbol{\Phi}$ as

$$(\tilde{\boldsymbol{G}})_{ij} = \boldsymbol{\phi}_i^\top \boldsymbol{\phi}_j = \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{3.26}$$

Performing PCA via a Gram matrix defined by kernels as above is called kernel PCA and has been introduced by Schölkopf, Smola, and Müller (1997).

Examples of kernels are the polynomial and Gaussian kernels,

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}')^a, \qquad k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{x}'||^2}{2\sigma^2}\right), \tag{3.27}$$

where the exponent a and width-parameter σ^2 are hyperparameters that need to be chosen by the user. We see that the two kernels only require the inner

products or distances between the data points $\boldsymbol{x}_i$ so that kernel PCA can also be used if that information is available only.

After specification of $\tilde{\boldsymbol{G}}$, we proceed exactly as in Section 3.1:

- Double centre $\tilde{\boldsymbol{G}}$ to compute

$$\boldsymbol{G} = \boldsymbol{H}_n \tilde{\boldsymbol{G}} \boldsymbol{H}_n. \tag{3.28}$$

- Compute the matrix $\boldsymbol{Z}$ with the (kernel) PC scores by an eigenvalue decomposition of $\boldsymbol{G}$,

$$\boldsymbol{Z} = \sqrt{\boldsymbol{\Sigma}_k} \boldsymbol{V}_k^\top, \tag{3.29}$$

  where, as before, the diagonal $k \times k$ matrix $\boldsymbol{\Sigma}_k$ contains the top k eigenvalues of $\boldsymbol{G}$ ordered from large to small, and the $n \times k$ matrix $\boldsymbol{V}_k$ contains the corresponding eigenvectors.

3.2.3 Example

Let us reconsider the circularly structured data of Figure 3.2 and use nonlinear PCA to compute a one-dimensional representation. We map the data points $\boldsymbol{x}_i$ to features $\boldsymbol{\phi}_i$ using the function $\phi(\boldsymbol{x}) = \phi(x_1, x_2)$,

$$\phi(x_1, x_2) = \begin{pmatrix} x_1, & x_2, & \sqrt{x_1^2 + x_2^2}, & \operatorname{atan}(x_2, x_1) \end{pmatrix}^\top, \tag{3.30}$$

where x_1, x_2 are the first and second element of the vector $\boldsymbol{x}$. The last two elements in the vector $\phi(\boldsymbol{x})$ are the polar coordinates of $\boldsymbol{x}$, which should be helpful given the circular structure of the data. Figure 3.3 visualises the first principle component scores computed from the transformed data matrix $\boldsymbol{\Phi}$. We can see that the lower dimensional representation by the first PC scores is reflecting the circular structure of the data. But there is a discontinuity in the scores around the point $(-1, 0)$, and the scores still ignore that a piece of the circle is missing: data points on the lower left are assigned similar values as data points on the lower right.

Figure 3.4 visualises the one-dimensional representation achieved by kernel PCA with the Gaussian kernel in (3.27). The hyperparameter σ^2 was determined from the quantiles of all distances between all (different) data points. The results do not seem better than the results with ordinary PCA in Figure 3.2.

Centring was the only preprocessing for the results above. I next also scaled each variable to unit variance before dimensionality reduction by kernel PCA (see Section 1.3.2 on data standardisation). Figure 3.5 shows that kernel PCA on the standardised data yielded a mostly meaningful one-dimensional representation for a wide range of different tuning parameters σ^2. The kernel PC scores change smoothly as we move on the data manifold. But the representation does ignore the gap between the lower-left and lower-right branch, so that points on the lower-left of the manifold (where $x_2 < -1.5$ and $x_1 \approx -1$) are considered closer to points on the lower-right of the manifold than points further up on the left branch. This may be considered a drawback of the representation.

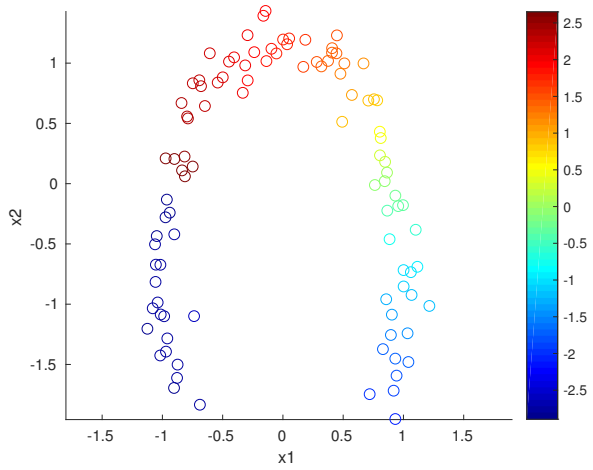


Figure 3.3: Dimensionality reduction by nonlinear PCA. Visualisation as in Figure 3.2(b).

3.3 Multidimensional scaling

Multidimensional scaling (MDS) is an umbrella term for several methods that operate on dissimilarities δ_{ij}. Euclidean distances are examples of dissimilarities but dissimilarities are more general in that they can be any kind of measure of difference between two data items. The goal of MDS is to find a configuration of points in the plane, or more generally the Euclidean space, so that their distances well represent the original dissimilarities.

3.3.1 Metric MDS

In metric MDS, the numerical values of the dissimilarities are assumed to carry information. This is in contrast to nonmetric MDS below where only the rank-order of the dissimilarities matters.

Denote the pairwise dissimilarities between n data points by δ_{ij}. A basic version of metric MDS consists in finding n points $\boldsymbol{z}_i \in \mathbb{R}^k$ that solve:

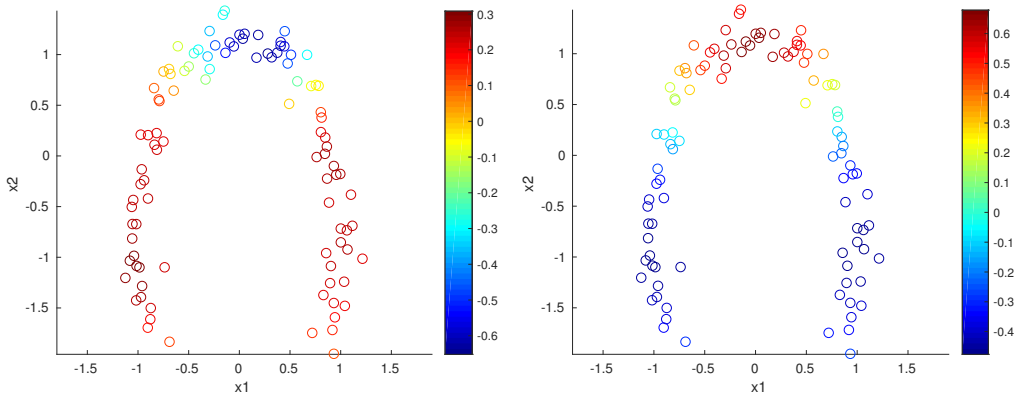
$$\underset{\boldsymbol{z}_1,\ldots,\boldsymbol{z}_n}{\text{minimise}} \quad \sum_{i<j} w_{ij}(||\boldsymbol{z}_i - \boldsymbol{z}_j|| - \delta_{ij})^2, \tag{3.31}$$

where $||\boldsymbol{z}_i - \boldsymbol{z}_j||$ is the Euclidean distance between $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$,

$$||\boldsymbol{z}_i - \boldsymbol{z}_j|| = \sqrt{(\boldsymbol{z}_i - \boldsymbol{z}_j)^\top (\boldsymbol{z}_i - \boldsymbol{z}_j)}. \tag{3.32}$$

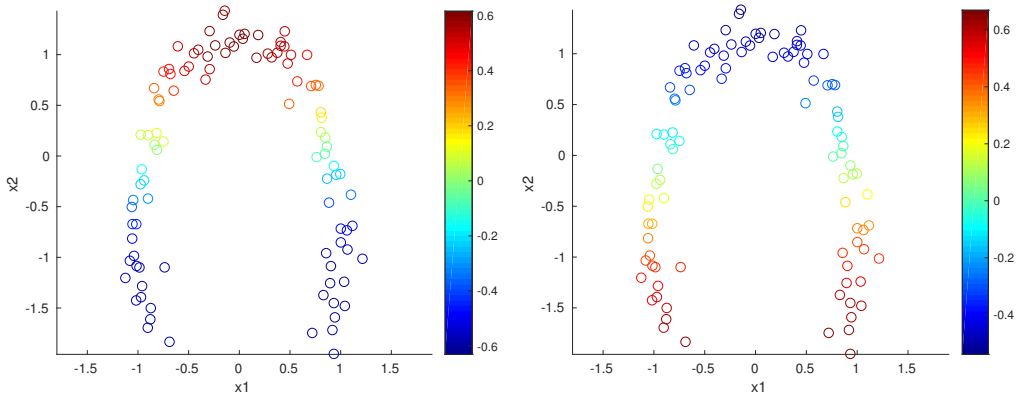
The $w_{ij} \geq 0$ are some weights specified by the user. The dimensionality k is typically set to two so that the data can be visualised on the plane. More complex versions of metric MDS exist where the dissimilarities δ_{ij} enter into the equation only after transformation with some monotonic function that is learned as well, see e.g. (Izenman, 2008, Section 13.7) or (Borg and Groenen, 2005, Chapter 9). The optimisation problem is typically solved by gradient descent.

For $w_{ij} = 1/\delta_{ij}$, the solution for the optimisation problem in (3.31) is called the Sammon nonlinear mapping. This choice of weights emphasises the faithful representation of small dissimilarities.



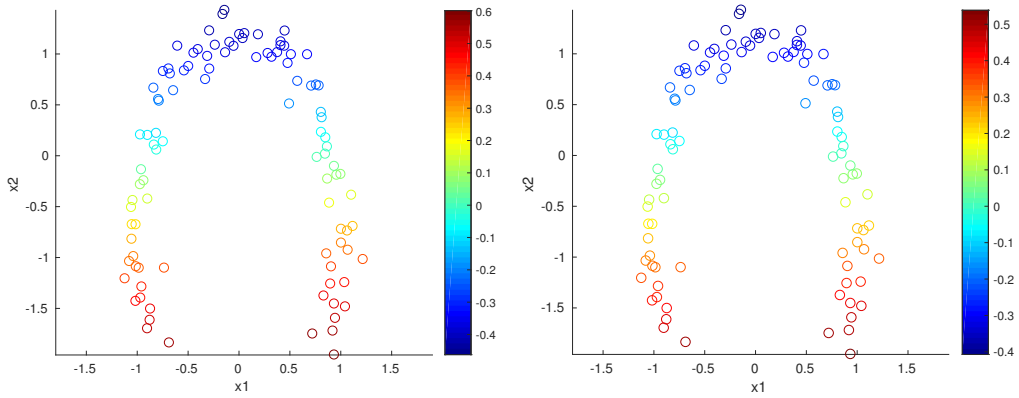
(a) σ^2: 0.01 quantile of all distances

(b) σ^2: 0.05 quantile



(c) σ^2: 0.1 quantile

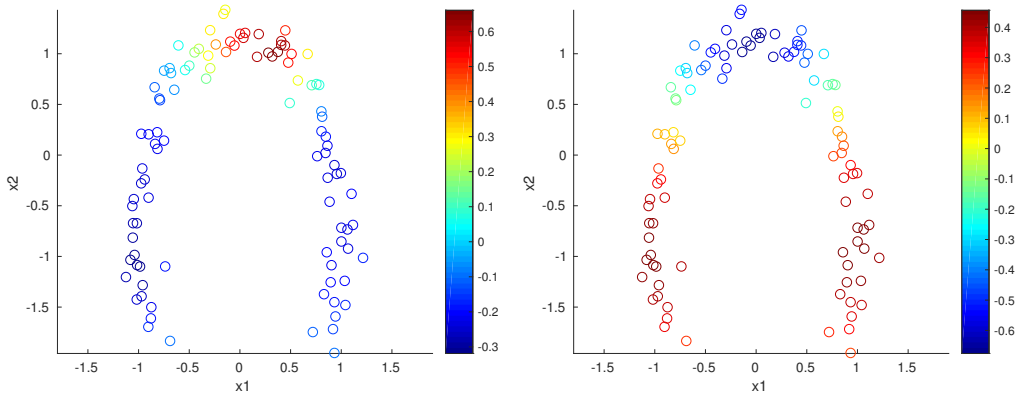
(d) σ^2: 0.25 quantile



(e) σ^2: 0.5 quantile

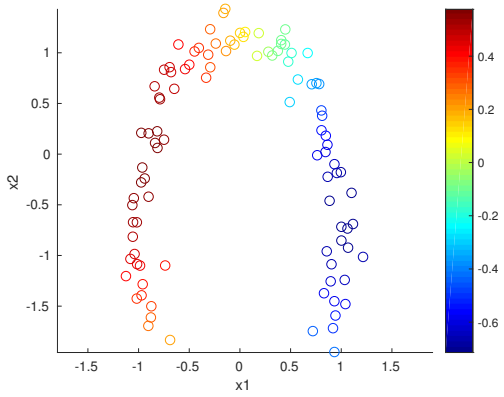
(f) σ^2: 0.75 quantile

Figure 3.4: Dimensionality reduction by kernel principal component analysis. The Gaussian kernel was used where σ^2 was determined by the quantiles of the distances. The colours indicate the value of the (kernel) principal component score assigned to a data point. The sign of the scores in each panel is arbitrary.

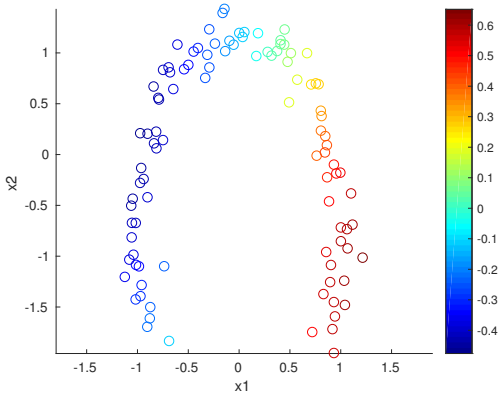


(a) σ^2: 0.01 quantile of all distances

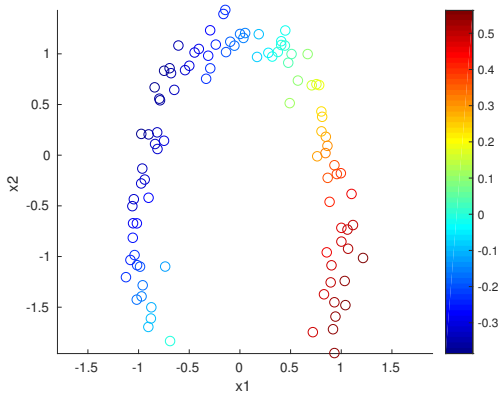
(b) σ^2: 0.05 quantile



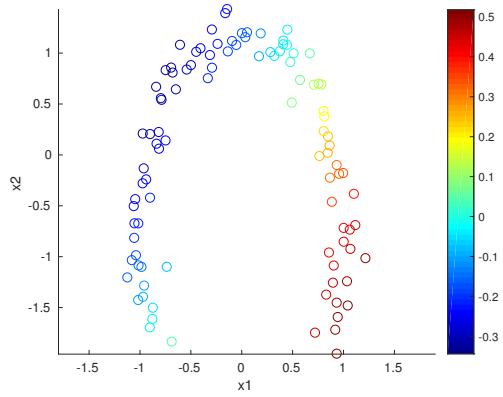
(c) σ^2: 0.1 quantile



(d) σ^2: 0.25 quantile



(e) σ^2: 0.5 quantile



(f) σ^2: 0.75 quantile

Figure 3.5: Dimensionality reduction by kernel principal component analysis on standardised data. The setup and visualisation is otherwise as in Figure 3.4.

3.3.2 Nonmetric MDS

In nonmetric MDS, only the relation between the δ_{ij} is assumed to matter, i.e. whether $\delta_{12} \geq \delta_{13}$ or $\delta_{12} \leq \delta_{13}$, and not the actual values of the dissimilarities. Such data are obtained, for example, when people are asked to rate the dissimilarity on a scale from 0 ("identical") to 5 ("very different").

Since the actual values of δ_{ij} do not matter, in nonmetric MDS, the optimisation problem in (3.31) is modified to

$$\underset{\boldsymbol{z}_1,\ldots,\boldsymbol{z}_n,f}{\text{minimise}} \quad \sum_{i<j} w_{ij}(||\boldsymbol{z}_i - \boldsymbol{z}_j|| - f(\delta_{ij}))^2, \tag{3.33}$$

where f is a monotonic (non-decreasing) function that converts the dissimilarities to distances. The optimisation problem is typically solved by iterating between optimisation with respect to the $\boldsymbol{z}_i$ and optimisation with respect to f, which can be done by regression (for further information, see, e.g. Izenman, 2008, Section 13.9).

3.3.3 Classical MDS

Classical MDS is also called classical scaling. It operates on the same kind of data as in metric scaling, that is, the actual numerical values of the dissimilarities are assumed to matter.

Classical scaling posits that the dissimilarities δ_{ij} are (squared) Euclidean distances between some unknown, hypothetical vectors of unknown dimensionality. Identifying the dissimilarity matrix $\boldsymbol{\Delta}$ that is formed by the δ_{ij} with a distance matrix between the unknown vectors brings us back to the setting from Section 3.1.3, and we can use the developed theory to determine the lower dimensional $\boldsymbol{z}_i \in \mathbb{R}^k, i = 1 \ldots n$.

1. Compute the hypothetical Gram matrix $\boldsymbol{G}'$ of the unknown centred data points,

$$\boldsymbol{G}' = -\frac{1}{2}\boldsymbol{H}_n\boldsymbol{\Delta}\boldsymbol{H}_n, \qquad \boldsymbol{H}_n = \boldsymbol{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top, \tag{3.34}$$

   as in (3.22). (The $'$ should emphasise that $\boldsymbol{G}'$ is a hypothetical Gram matrix, it does not denote the transpose of the matrix.)

2. Compute the top k eigenvalues σ_k^2 and corresponding eigenvectors $\boldsymbol{v}_k \in \mathbb{R}^n$ of $\boldsymbol{G}$, and form the matrices $\boldsymbol{\Sigma}_k = \text{diag}(\sigma_1^2, \ldots, \sigma_k^2)$ and $\boldsymbol{V}_k = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$.

3. The $k \times n$ matrix $\boldsymbol{Z}$ with the $\boldsymbol{z}_i$ as its columns,

$$\boldsymbol{Z} = (\boldsymbol{z}_1, \cdots, \boldsymbol{z}_n), \tag{3.35}$$

   is then given by $\boldsymbol{Z} = \sqrt{\boldsymbol{\Sigma}_k}\boldsymbol{V}_k^\top$, as in (2.80).

Classical MDS can thus turn any dissimilarity matrix $\boldsymbol{\Delta}$ into a configuration of lower-dimensional vectors $\boldsymbol{z}_i$ that represent the dissimilarities. It also has the nice property that it produces nested solutions because the classical MDS solution for $k' < k$ is directly given by the first k' coordinates of the k-dimensional $\boldsymbol{z}_i$.

There is one subtle technical caveat: The matrix $\boldsymbol{\Delta}$ is symmetric but not necessarily positive semidefinite. This is because we only pretended that $\boldsymbol{\Delta}$ corresponds to Euclidean distances in some unknown space, but this may only hold approximately. As $\boldsymbol{\Delta}$ is not necessarily positive semidefinite, some of its eigenvalues may be negative so that taking square roots as above in the third step would not yield meaningful representations. The simple fix is to choose k small enough that all eigenvalues contained in $\boldsymbol{\Sigma}_k$ are indeed positive.

For negative definite matrices $\boldsymbol{\Delta}$, eigenvectors corresponding to negative eigenvalues are thus excluded. We can think of this operation as a way to approximate $\boldsymbol{\Delta}$ by a positive semidefinite matrix. It turns out that the simple operation of excluding directions with negative eigenvalues is actually the optimal positive semidefinite approximation of $\boldsymbol{\Delta}$ with respect to the Frobenius norm (see Section A.10.3). Further results from linear algebra show that the lower dimensional representation $\boldsymbol{Z} = \sqrt{\boldsymbol{\Sigma}_k}\boldsymbol{V}_k^\top$ yields the best low rank approximation of $\boldsymbol{G}'$ with respect to the Frobenius norm (see Section A.10.4). That is, $\boldsymbol{Z}$ is the solution to

$$\begin{aligned} \underset{\boldsymbol{M}}{\text{minimise}} \quad & ||(-\frac{1}{2}\boldsymbol{H}\boldsymbol{\Delta}\boldsymbol{H}) - \boldsymbol{M}^\top\boldsymbol{M}||_F \\ \text{subject to} \quad & \text{rank}(\boldsymbol{M}^\top\boldsymbol{M}) = k \end{aligned} \tag{3.36}$$

This is a different optimisation problem than the one in (3.31) for metric MDS, and the solution returned by classical and metric MDS are generally not the same.

3.3.4 Example

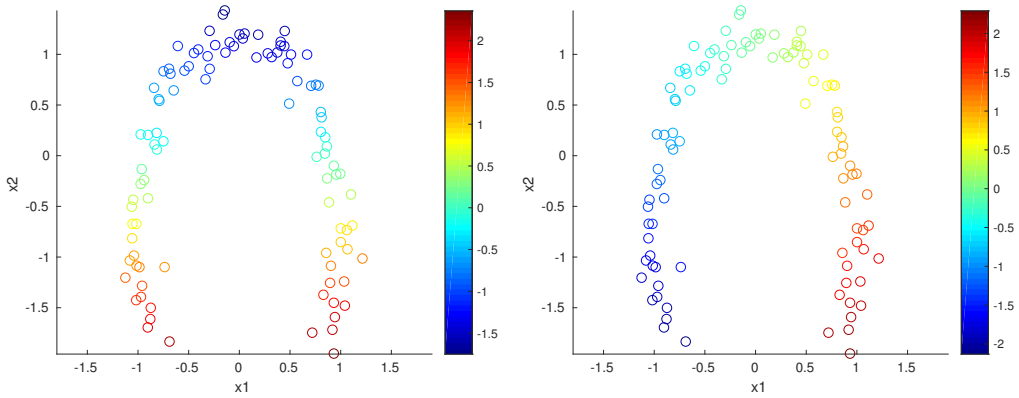
I applied the Sammon nonlinear mapping to the circularly shaped data in Figure 3.2, reducing the dimension from two to one. The Sammon mapping is the solution of the optimisation problem in (3.31), which can have multiple local minima. I thus ran the algorithm multiple times and Figure 3.6 shows the two different solutions typically obtained. The solution in Figure 3.6(a) basically corresponds to the PCA-solution. The solution figure (b), however, shows that the learned one-dimensional representation, i.e. the points $z_1, \dots z_n$ in (3.31), do take the manifold structure of the data into account.

The solution in 3.6(a) assigns the same low dimensional coordinate to the point on the left and right branch of the manifold. Their distance is thus practically zero even though in the original data space, their distance is rather large. The solution (b) assigns different values to the points on the left and the right half of the circle, so that their distances in the lower dimensional space better matches their distances in the original space.

Figure 3.7 plots the distances in the original space against the distances in the lower dimensional space. The phenomenon described above is well visible in that the solution from 3.6(a) has distances equal to zero even though the original distances are around two.

3.4 Isomap

Classical MDS is used as part of the isometric feature mapping (Isomap) algorithm (Tenenbaum, Silva, and Langford, 2000) where the dissimilarity δ_{ij} between



(a) Local optimum (b) Better local optimum

Figure 3.6: Dimensionality reduction by the Sammon nonlinear mapping. The method is prone to local optima. The solution in (b) has a small cost than (a). The colours indicate the value of the one-dimensional coordinate assigned to a data point.

two data points $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\boldsymbol{x}_j \in \mathbb{R}^d$ is measured by the shortest distance between them when only allowed to travel on the data manifold from one neighbouring data point to the next. This is called a geodesic distance. The neighbourhood of a data point can be taken to be the m-nearest neighbours or also all points that are within a certain (Euclidean) distance. The set of neighbourhoods defines a graph on which one is allowed to move. For further information on Isomap, see the original algorithm or the books by Izenman (2008, Section 16.6) and Lee and Verleysen (2007, Section 4.3).

Figure 3.8 shows the graphs for the circularly shaped data in Figure 3.2. We see that for a neighbourhood that is specified by 5 nearest neighbours, the graph has two unconnected components. In this case, Isomap is often applied to each component separately.

Figure 3.9 visualises the one-dimensional coordinates $z_1, \ldots, z_n$ that are obtained by applying classical MDS on the geodesic distances. They well represent the circular structure when the learned graph is connected.

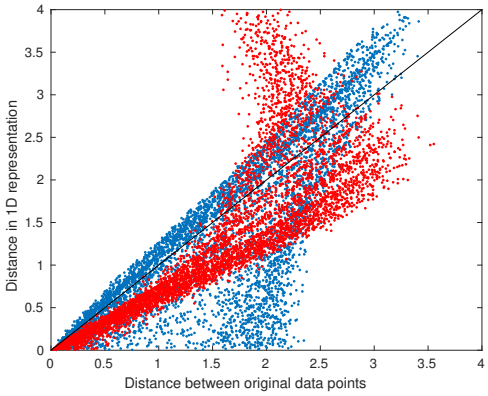
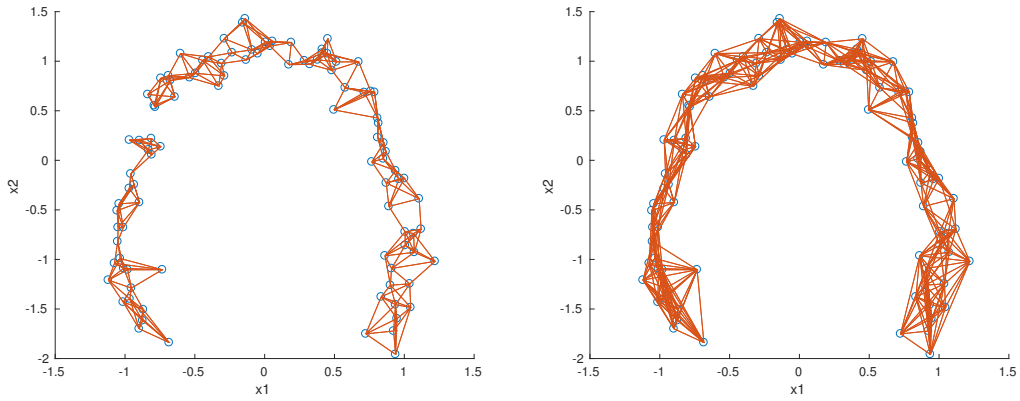
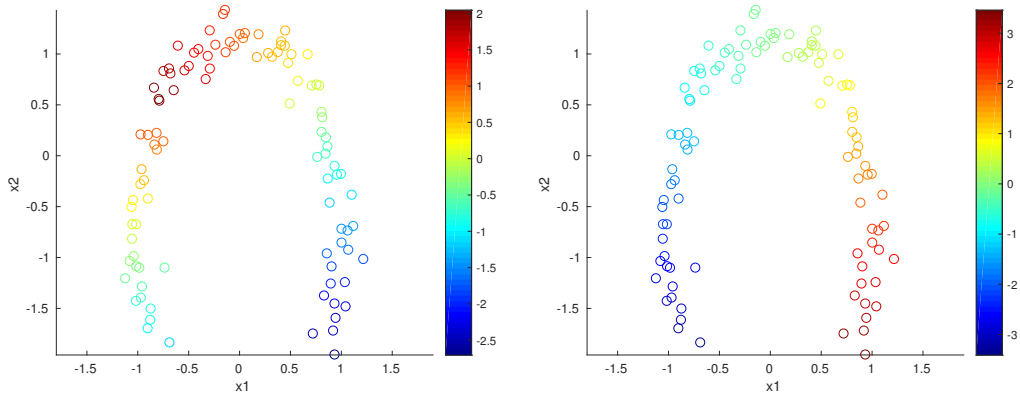


Figure 3.7: Dimensionality reduction by the Sammon nonlinear mapping. Comparison of the distances in the original and lower dimensional space. The blue points correspond to the solution in Figure 3.6(a); the red points to the solution in Figure 3.6(b)



(a) 5 neighbours (b) 10 neighbours

Figure 3.8: Dimensionality reduction by Isomap. Comparison of graphs constructed from different neighbourhoods.



(a) 5 neighbours (b) 10 neighbours

Figure 3.9: Dimensionality reduction with Isomap. The colours indicate the value of the one-dimensional coordinate assigned to each data point.

References

[1] I. Borg and P.J.F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[2] A.J. Izenman. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer, 2008.

[3] J.A. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction*. Springer, 2007.

[4] B. Schölkopf, A. Smola, and K.-R. Müller. "Kernel principal component analysis". In: *Artificial Neural Networks ICANN'97*. Springer Berlin Heidelberg, 1997, pp. 583–588.

[5] B. Schölkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.

[6] J. B. Tenenbaum, V. de Silva, and J. C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science* 290.5500 (2000), pp. 2319–2323.

Chapter 4

Performance evaluation in predictive modelling

Regression and classification are typical examples of predictive modelling. The general goal in predictive modelling of data is to identify a relationship between some predictor (input) and some target (output) variables that enables one to accurately predict the values of the target variables for some newly observed values of the predictor variables. This chapter is about evaluating the performance of prediction models and methods, and about selecting among competing alternatives.

4.1 Prediction and training loss

The key notions of prediction and training loss are introduced.

4.1.1 Prediction loss

Let us denote the predictor variables by $\boldsymbol{x}$ and let us assume that we are only interested in a single target variable y. In regression, y is real-valued while in classification, y is the class label, e.g. minus one and one. Both $\boldsymbol{x}$ and y are considered random variables that have a joint probability density function $p(\boldsymbol{x}, y)$. For any fixed value of $\boldsymbol{x}$, the target variable y thus follows the conditional distribution $p(y|\boldsymbol{x})$. Both the joint pdf and the conditional pdf are unknown.

From a probabilistic perspective, the goal of predictive modelling is to estimate the conditional distribution $p(y|\boldsymbol{x})$ from observed data. In many cases, however, we need to report a single estimated value of y rather than a whole distribution. That is, we are looking for a prediction function $h(\boldsymbol{x})$ that provides an estimate $\hat{y} = h(\boldsymbol{x})$ for any value of $\boldsymbol{x}$.

Making a prediction $\hat{y}$ may incur a loss $\mathcal{L}(\hat{y}, y)$ so that certain prediction functions are better than others. Due to the stochasticity of the predictors and the target, the quality of a prediction function h is measured via the expected value of $\mathcal{L}(\hat{y}, y)$,

$$\mathcal{J}(h) = \mathbb{E}_{\hat{y},y}\left[\mathcal{L}(\hat{y}, y)\right] = \mathbb{E}_{\boldsymbol{x},y}\left[\mathcal{L}(h(\boldsymbol{x}), y)\right], \tag{4.1}$$

which is called the prediction loss. The term $\mathbb{E}_{\boldsymbol{x},y}$ means expectation with respect to $p(\boldsymbol{x}, y)$.

The goal of predictive modelling can be formulated as the optimisation problem

$$\underset{h}{\text{minimise}}\, \mathcal{J}(h). \tag{4.2}$$

While concise, the formulation hides some fundamental issues: First, we generally cannot compute the expectation over $(\boldsymbol{x}, y)$ analytically. Secondly, the loss function $\mathcal{L}$ may not be easy to evaluate – it could, for example, be given by user ratings that indicate the quality of a prediction $\hat{y}$. And thirdly, minimising the prediction loss with respect to a function is generally difficult.

4.1.2 Training loss

The objective in (4.2) can typically not be computed and the optimisation problem not be solved exactly. We make a number of approximations to obtain a computable loss function for which optimisation is, at least in principle, feasible.

If n samples $(\boldsymbol{x}_i, y_i)$ are available that are each independently drawn from $p(\boldsymbol{x}, y)$,

$$(\boldsymbol{x}_i, y_i) \overset{iid}{\sim} p(\boldsymbol{x}, y), \tag{4.3}$$

the expectation in the definition of the prediction loss can be approximated by a sample average,

$$\mathcal{J}(h) \approx \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(h(\boldsymbol{x}_i), y_i). \tag{4.4}$$

The samples $(\boldsymbol{x}_i, y_i)$ are called the training data $\mathcal{D}^{\text{train}}$,

$$\mathcal{D}^{\text{train}} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}. \tag{4.5}$$

In the sample-average approximation, we assumed that training data are available that come from the same distribution $p(\boldsymbol{x}, y)$ as the data for which we would like to perform predictions. In many cases, however, this assumption is violated and the training data come from a different distribution. This can lead to inaccurate predictions, so that care should be taken that at least parts of the training data are representative of the conditions for which the prediction function will be ultimately used.

Instead of minimising the (approximate) prediction loss with respect to any function h, we typically search for h inside model families that are parametrised by some parameters $\boldsymbol{\theta}$, so that $h(\boldsymbol{x}) = h_{\boldsymbol{\lambda}}(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{\lambda}$ is a vector of hyperparameters indicating the model family and some tuning parameters associated with it. The hyperparameters could for example indicate whether we use regression trees or neural networks. And when using neural networks, they could additionally indicate the number of hidden units, whereas $\boldsymbol{\theta}$ would correspond to the weights in the network.

The number of parameters $\boldsymbol{\theta}$ may be rather large so that gradient information is needed in the optimisation. But some loss functions $\mathcal{L}$, like for example classification error, are not differentiable so that gradient descent is not possible. Other loss functions $\mathcal{L}$ may be expensive to evaluate, like for example when based on user ratings. For practical reasons, we may thus prefer to determine $\boldsymbol{\theta}$ by minimising a proxy loss function L rather than the loss function $\mathcal{L}$ that we are really interested in.

In summary, instead of working with $\mathcal{J}(h)$ we work with the training loss function $J_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$,

$$J_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(h_{\boldsymbol{\lambda}}(\boldsymbol{x}_i;\boldsymbol{\theta}), y_i). \tag{4.6}$$

Minimisation of $J_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$ is typically done by minimising the loss function with respect to $\boldsymbol{\theta}$ separately for fixed values of the hyperparameters. We then obtain a set of prediction function $\hat{h}_{\boldsymbol{\lambda}}(\boldsymbol{x})$ indexed by $\boldsymbol{\lambda}$,

$$\hat{h}_{\boldsymbol{\lambda}}(\boldsymbol{x}) = h_{\boldsymbol{\lambda}}(\boldsymbol{x};\hat{\boldsymbol{\theta}}_{\boldsymbol{\lambda}}), \qquad \hat{\boldsymbol{\theta}}_{\boldsymbol{\lambda}} = \operatorname*{argmin}_{\boldsymbol{\theta}} J_{\boldsymbol{\lambda}}(\boldsymbol{\theta}). \tag{4.7}$$

Determining $\hat{h}_{\boldsymbol{\lambda}}(\boldsymbol{x})$ from training data is called model estimation. The associated minimal value of the training loss function is the training loss $J^*_{\boldsymbol{\lambda}}$,

$$J^*_{\boldsymbol{\lambda}} = \min_{\boldsymbol{\theta}} J_{\boldsymbol{\lambda}}(\boldsymbol{\theta}). \tag{4.8}$$

The training loss function $J_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$, the prediction function $\hat{h}_{\boldsymbol{\lambda}}(\boldsymbol{x})$, and the corresponding training loss $J^*_{\boldsymbol{\lambda}}$ all depend on the training data $\mathcal{D}^{\text{train}}$. Different training data sets will result in different loss functions, different prediction functions, and different training losses. This means that they are all random quantities whose stochasticity is induced by the variability of the training data.

Minimising $J_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$ for several $\boldsymbol{\lambda}$ yields a set of prediction functions $\hat{h}_{\boldsymbol{\lambda}}(\boldsymbol{x})$. Choosing from them the prediction function $\hat{h}(\boldsymbol{x})$ that is actually used for making predictions is done by a process called hyperparameter selection. If the hyperparameter indicates the model family, the process is called model selection. We will see below that choosing the hyperparameters that yield the smallest training loss is generally a bad idea because the corresponding prediction function tends to be highly specific to the particular training data used and thus may perform poorly when making predictions for new (unseen) values of $\boldsymbol{x}$.

4.1.3 Example

Let us illustrate the above concepts on a simple example where the joint distribution of the prediction and target variable is given by

$$p(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right), \tag{4.9}$$

$$p(y|x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y - g(x))^2\right), \qquad g(x) = \frac{1}{4}x + \frac{3}{4}x^2 + x^3. \tag{4.10}$$

The function $g(x)$ is the conditional mean $\mathbb{E}(y|x)$. It minimises the expected square loss, i.e. (4.1) for

$$\mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2. \tag{4.11}$$

We assume that we have a training data set $\mathcal{D}^{\text{train}}$ with n data points (x_i, y_i). Figure 4.1(a) shows $g(x)$ and an example training set.

Training loss for linear regression

Let us first work with a linear prediction model so that

$$h_1(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x, \qquad \boldsymbol{\theta} = (\theta_0, \theta_1)^\top, \tag{4.12}$$

where θ_0 is the intercept and θ_1 the slope parameter. When using the quadratic loss, the training loss function is

$$J_1(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \theta_0 - \theta_1 x_i)^2. \tag{4.13}$$

For any value of θ_1, the optimal value of the constant θ_0 is

$$\hat{\theta}_0 = \bar{y} - \theta_1 \bar{x}, \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i, \qquad \bar{x} = \sum_{i=1}^{n}\frac{1}{n} x_i, \tag{4.14}$$

so that θ_1 is the only unknown when working with centred data. The training loss function becomes

$$J_1(\theta_1) = \frac{1}{n}\sum_{i=1}^{n}((y_i - \bar{y}) - \theta_1(x_i - \bar{x}))^2. \tag{4.15}$$

Minimising $J_1(\theta_1)$ gives $\hat{\theta}_1$,

$$\hat{\theta}_1 = \operatorname*{argmin}_{\theta_1} J_1(\theta_1), \tag{4.16}$$

and the estimated prediction model $\hat{h}_1(x)$,

$$\hat{h}_1(x) = \hat{\theta}_0 + \hat{\theta}_1 x = \bar{y} + \hat{\theta}_1(x - \bar{x}). \tag{4.17}$$

Figure 4.2(a) shows the training loss function $J_1(\theta_1)$ and the estimated regression function $\hat{h}_1(x)$.

The training loss function $J_1(\boldsymbol{\theta})$ in (4.13) varies as the training data vary. The training loss function $J_1(\boldsymbol{\theta})$ is a random quantity. Its minimiser inherits the randomness, and the minimal training loss

$$J_1^* = \min_{\boldsymbol{w}} J_1(\boldsymbol{w}) \tag{4.18}$$

is a random variable too. The randomness is due to the variability of the training data $\mathcal{D}^{\text{train}}$. Random quantities have a probability distribution, and Figure 4.3 visualises the distribution of the training loss function and the distribution of its minima J_1^*. The probability density function of J_1^* was estimated from a histogram using Equation (1.26) from Chapter 1.

Training loss for polynomial regression

Instead of working with a linear prediction model, let us now consider more general prediction models of the form

$$h_\lambda(x;\boldsymbol{\theta}) = \sum_{k=0}^{\lambda}\theta_k x^k, \qquad \boldsymbol{\theta} = (\theta_0, \ldots, \theta_\lambda)^\top. \tag{4.19}$$

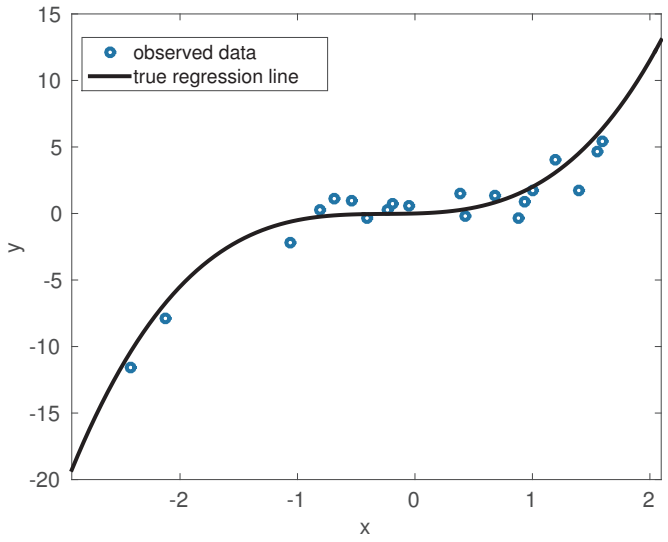


Figure 4.1: Example nonlinear regression (prediction) problem. The true regression curve is shown in black and the example training data in blue. The size of the training data is $n = 20$.

The functions $h_\lambda(x;\boldsymbol{\theta})$ are polynomials of degree λ. The $h_\lambda(x;\boldsymbol{\theta})$ correspond to a set of prediction models: We have one prediction model for each value of λ. Its complexity and number of free parameters increases with increasing values of λ. For $\lambda = 0$, the prediction model is a constant, and for $\lambda = 1$ we obtain the linear model used above.

We can estimate the prediction models by minimising the average square loss as before,

$$J_\lambda(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{k=0}^{\lambda}\theta_k x_i^k - y_i\right)^2, \tag{4.20}$$

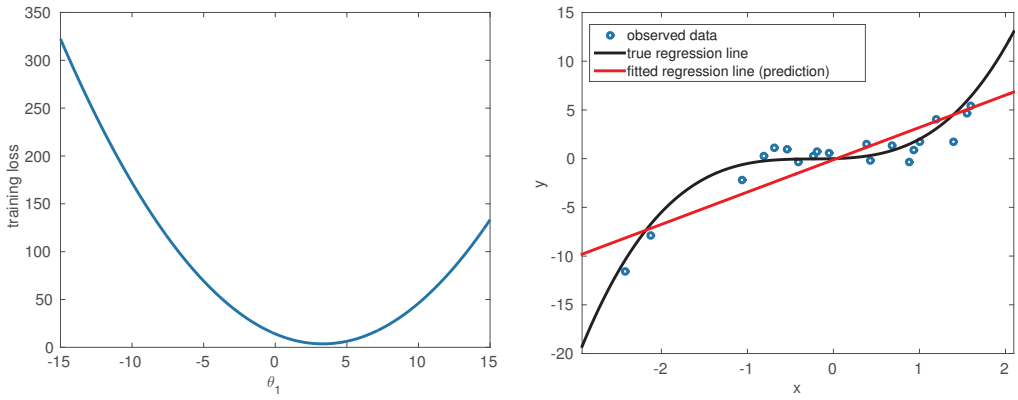
Minimising $J_\lambda(\boldsymbol{\theta})$ yields the prediction functions $\hat{h}_\lambda$ with training loss J_λ^*.

Figure 4.4(a) shows the estimated probability density function (pdf) of the training loss J_λ^*. The pdf for the polynomial of degree one is the same as in Figure 4.3(b). We see that the training loss tends to become smaller if the complexity of the model increases.

Another view is provided in Figure 4.4(b). The figure shows the training loss J_λ^* as a function of the degree of the polynomials. We see that both the variance and the mean of the training loss becomes smaller with increasing complexity of the model. The same holds for more general models than the polynomial one used here. Indeed, it is generally possible to increase the complexity of the model to the point where the minimal training loss becomes zero (see Section 4.2.2).

4.2 Generalisation performance

The training loss can be made smaller by using more complex models. But we are ultimately interested in the prediction rather than in the training loss. In other



(a) Training loss function (b) Fitted prediction model

Figure 4.2: The true regression curve is shown in black and the estimated regression curve in red.

words, we are interested in how well we perform on unseen data after learning. This is called generalisation performance.

4.2.1 Generalisation for prediction functions and algorithms

The training loss function in (4.6) was used as a proxy of the prediction loss $\mathcal{J}(h)$ that we are really interested in minimising. We say that a prediction function $\hat{h}$ generalises well if its prediction loss $\mathcal{J}(\hat{h})$ is small,

$$\mathcal{J}(\hat{h}) = \mathbb{E}_{\boldsymbol{x},y}\left[\mathcal{L}(\hat{h}(\boldsymbol{x}), y)\right]. \tag{4.21}$$

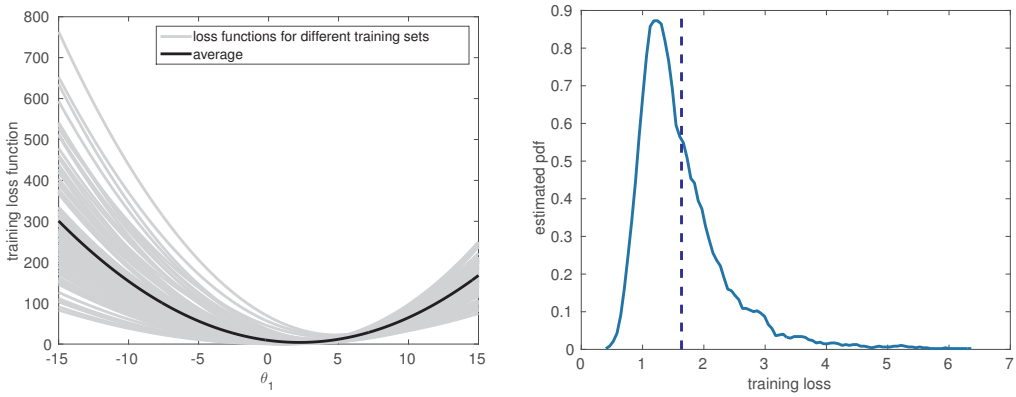
The prediction loss $\mathcal{J}(\hat{h})$ is called the generalisation loss or the test loss of $\hat{h}$. This is because $\mathcal{J}(\hat{h})$ measures whether the performance of $\hat{h}$ generalises from the training data $\mathcal{D}^{\text{train}}$ and training loss function L to new "test" data $(\boldsymbol{x}, y) \sim p(\boldsymbol{x}, y)$ and the prediction loss function $\mathcal{L}$. As argued above, $\mathcal{J}(\hat{h})$ can generally not be computed. But unlike before, we here do not need to solve an optimisation problem. We only need to evaluate $\mathcal{J}$ at $\hat{h}$, which is considerably easier. It amounts to estimating the expected value of $\mathcal{L}(\hat{h}(\boldsymbol{x}), y)$, which can be done with hold-out data (see Section 4.3.1).

Since the prediction function $\hat{h}$ depends on the training data $\mathcal{D}^{\text{train}}$, the prediction loss $\mathcal{J}(\hat{h})$ depends on the training data too. The prediction loss $\mathcal{J}(\hat{h})$ is thus a random variable whose stochasticity is induced by the variability of the training sets. We will see now that its expected value $\bar{\mathcal{J}}$ can be used to measure the generalisation performance of prediction algorithms.

Let us denote the algorithm that is used to turn training data $\mathcal{D}^{\text{train}}$ into a prediction function $\hat{h}$ by $\mathcal{A}$ so that

$$\hat{h} = \mathcal{A}(\mathcal{D}^{\text{train}}). \tag{4.22}$$

The algorithm $\mathcal{A}$ subsumes all operations needed to turn training data into a prediction function, including for example the minimisation of the loss function or the selection of hyperparameters. Think of it as a piece of code that takes



(a) Training loss functions (b) Training loss

Figure 4.3: The loss functions and their minima are random quantities. The figures illustrate their distribution. (a) Loss functions for different training sets. (b) Distribution of the square root of the training loss J_1^* for different training sets. The dashed vertical line indicates the mean of the estimated distribution.

training data as input and returns a prediction function $\hat{h}$ as output. We can then write the expected prediction loss $\bar{\mathcal{J}}$ as a function of $\mathcal{A}$

$$\bar{\mathcal{J}}(\mathcal{A}) = \mathbb{E}_{\mathcal{D}^{\text{train}}}\left[\mathcal{J}(\hat{h})\right] = \mathbb{E}_{\mathcal{D}^{\text{train}}}\left[\mathcal{J}\left(\mathcal{A}(\mathcal{D}^{\text{train}})\right)\right]. \tag{4.23}$$

While $\mathcal{J}(\hat{h})$ in (4.21) measures the performance of a specific $\hat{h}$, $\bar{\mathcal{J}}(\mathcal{A})$ measures the performance of the process, or algorithm, that is used to obtain $\hat{h}$ from the training data. The purpose of the two performance measures in (4.21) and (4.23) is thus different: $\mathcal{J}(\hat{h})$ can be used to compare different prediction functions while $\bar{\mathcal{J}}(\mathcal{A})$ can be used to compare different prediction algorithms.

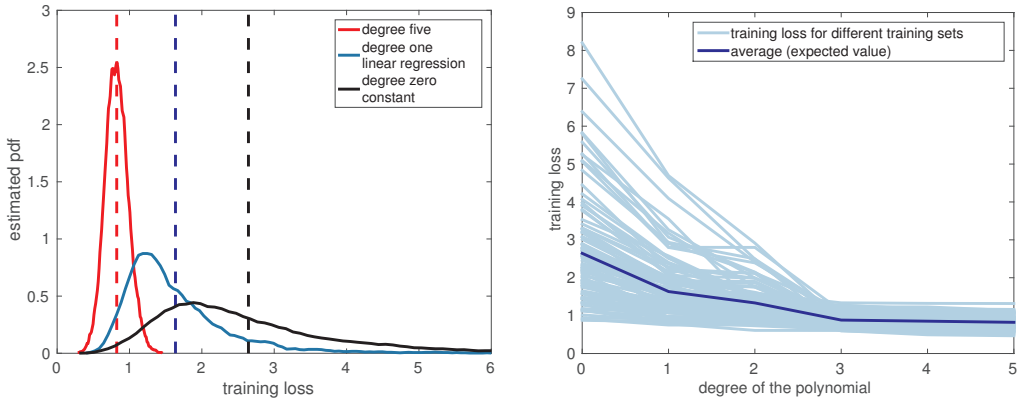
If we consider algorithms $\mathcal{A}_{\boldsymbol{\lambda}}$ that operate with different (fixed) hyperparameters $\boldsymbol{\lambda}$, we can use $\bar{\mathcal{J}}(\mathcal{A}_{\boldsymbol{\lambda}})$ to compare and select among them. Like $\mathcal{J}(\hat{h})$, however, the expected prediction loss $\bar{\mathcal{J}}(\mathcal{A})$ can typically not be computed in closed form and needs to be estimated, for which cross-validation can be used (see Section 4.3.1).

4.2.2 Overfitting and underfitting

Let us consider the training and (expected) prediction loss of the prediction functions $\hat{h}_{\boldsymbol{\lambda}}(\boldsymbol{x})$ in (4.7) for different models. By using a model with n free parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)^\top$, we can make the training loss always equal to zero. Indeed, if

$$h_{\text{flexible}}(\boldsymbol{x}; \boldsymbol{\theta}) = \begin{cases} \theta_i & \text{if } \boldsymbol{x} = \boldsymbol{x}_i \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

we can set $\hat{\theta}_i = y_i$ and the training loss is zero (assuming that $L(y_i, y_i) = 0$). Unless $\boldsymbol{x}$ and y can only take discrete values that are all included in the training data, the (expected) prediction loss of $\hat{h}_{\text{flexible}}$ will be large. The prediction function is overfitting the training data. More generally, a model has been overfitted



(a) Distribution of the training loss

(b) Training loss for different models

Figure 4.4: Distribution of the training loss for different degrees of the polynomial prediction model. The complexity of the model increases with the degree of the polynomial.

to the training data if reducing its complexity reduces the (expected) prediction loss.

On the other hand, a prediction model

$$h_{\text{rigid}}(\boldsymbol{x};\theta) = \theta, \tag{4.25}$$

which always takes on a constant value, will have a training loss that is rather large. Unless the response variable y does indeed not depend on the predictors $\boldsymbol{x}$, the (expected) prediction loss will be large, too, and could be decreased by choosing a more flexible model that better captures the relationship between $\boldsymbol{x}$ and y. Prediction functions like $h_{\text{rigid}}(\boldsymbol{x};\theta)$ are said to underfit the training data.

The problem of over- and underfitting can be addressed by model selection and by means of regularisation. In regularisation, we work with flexible models but augment the training loss function $J_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$, which measures the quality of the prediction, with an additional term that penalises flexibility of the prediction function. For training, we thus solve the optimisation problem

$$\underset{\boldsymbol{\theta}}{\text{minimise}}\, J_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) + \lambda_{\text{reg}} R(\boldsymbol{\theta}), \tag{4.26}$$

where $R(\boldsymbol{\theta})$ is the penalty term on the parameters of $h_{\boldsymbol{\lambda}}(\boldsymbol{x};\boldsymbol{\theta})$ and λ_{reg} indicates the strength of the regularisation. Typical penalty terms are

$$R(\boldsymbol{\theta}) = \sum_i \theta_i^2 \qquad (L_2 \text{ or Tikhonov regularisation}) \tag{4.27}$$

$$R(\boldsymbol{\theta}) = \sum_i |\theta_i| \qquad (L_1 \text{ regularisation}) \tag{4.28}$$

but also terms that penalises rapidly varying functions. The amount of regularisation depends on λ_{reg}. We can consider it to be another hyperparameter that we can select in order to maximise generalisation performance.

4.2.3 Example

We continue the example of polynomial regression to illustrate how the generalisation performance depends on the model complexity and the size of the training data.

Generalisation performance and model complexity

Figure 4.5(a) shows the training and prediction loss of the fitted polynomial regression model $\hat{h}_\lambda$ as a function of the degree of the polynomial (model complexity) λ. We can see that the prediction loss and training loss are generally not the same, i.e.

$$\mathcal{J}(\hat{h}_\lambda) \neq J^*_\lambda. \tag{4.29}$$

In the figure, the prediction loss is smallest for $\lambda = 4$, and while a five-degree polynomial has the smallest training loss, it has the largest prediction loss. Such a mismatch between training and prediction performance is due to overfitting. The estimated model $\hat{h}_\lambda$ is highly tuned to the specific training data $\mathcal{D}^{\text{train}}$ and does not reflect the general relationship between the predictor and the target variable. In contrast, we see that increasing the complexity of the degree-zero or degree-one polynomial will decrease the prediction loss. That is, these models are underfitting the training data.

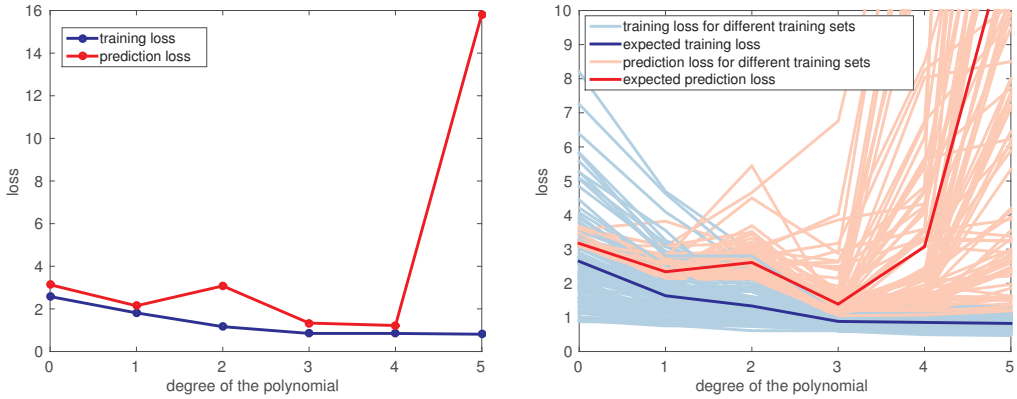
While Figure 4.5(a) depicts the training and prediction loss for a particular training set, Figure 4.5(b) shows their distribution over different training data sets. We can see that the variability of the prediction loss increases with the flexibility of the model. This is due to overfitting because the estimated model then depends strongly on the particularities of each training set that are bound to vary when the training data change. Underfitting, in contrast, leads to a small variability of the prediction loss because the fitted model captures comparably few properties of the training data.

The red solid line in Figure 4.5(b) shows the expected (average) prediction loss $\bar{\mathcal{J}}$ as a function of λ. While a model of degree $\lambda = 4$ performed best for the particular training data used in (a), models of degree $\lambda = 3$ yield the best performance on average. We see that there is here a difference between the generalisation performance of a specific fitted model and the generalisation performance of a model-family across different training sets, which reflects the general difference between $\mathcal{J}(\hat{h}_{\boldsymbol{\lambda}})$ and $\bar{\mathcal{J}}(\mathcal{A}_{\boldsymbol{\lambda}})$ discussed in Section 4.2.1.

Generalisation performance and the size of the training data

The results so far were obtained for training sets of size $n = 20$. We saw that flexible models tended to overfit the training data, so that there was stark difference between training and prediction performance. Here, we illustrate how the size of the training data influences the generalisation performance.

Figure 4.6 shows the expected training and prediction loss as a function of the size n of the training data for polynomial models of different degree. We can generally see that the training and prediction loss approach each other as the sample size increases. Note that they may generally not reach the same limit as n increases because the training and prediction loss functions L and $\mathcal{L}$, for example, may not be the same.



(a) Single training set

(b) Distribution over training sets

Figure 4.5: Training versus prediction performance of different prediction models.

Figure 4.6(a) shows that increasing the model complexity decreases the prediction loss for the models of degree zero and one. Moreover, their prediction loss does not decrease below a certain level even if the size of the training data increases. Both phenomena are a sign of underfitting.

Figure 4.6(b) shows the average training and prediction loss for the polynomial model of degree five. The large difference between training and prediction loss for small sample sizes is due to overfitting. As the size of the training data increases, however, the gap between the two losses becomes smaller, which means that the amount of overfitting decreases.

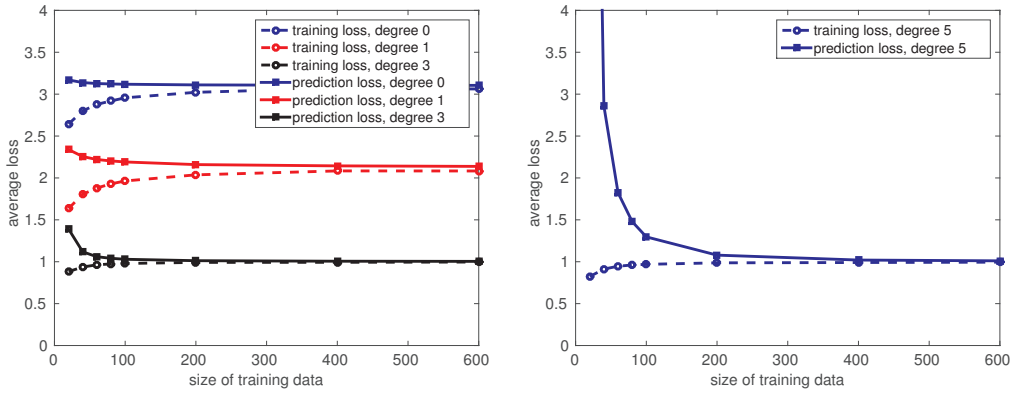
Comparing Figure 4.6(a) and (b) shows us further that even for large samples, on average, the model of degree five does here not achieve a smaller prediction loss than the model of degree three. Hence, for this problem, there is no advantage in using a more complex model than the model of degree three. In general, we can use model selection to choose among candidate models, or regularisation to avoid overfitting flexible models on small training data. Both model selection and choosing the right amount of regularisation correspond to hyperparameter selection.

4.3 Estimating the generalisation performance

We typically need to estimate the generalisation performance twice: Once for hyperparameter selection, and once for final performance evaluation. We first discuss two methods for estimating the generalisation performance and then apply them to the two aforementioned tasks.

4.3.1 Methods for estimating the generalisation performance

The hold-out and the cross-validation approach to estimate the generalisation performance are presented.



(a) Polynomials of degree zero, one, and three

(b) Polynomial of degree five

Figure 4.6: Average training versus average prediction performance for different sizes of the training data.

Hold-out approach

Assume that the prediction function $\hat{h}$ has been obtained using training data $\mathcal{D}^{\text{train}}$, i.e.

$$\hat{h} = \mathcal{A}(\mathcal{D}^{\text{train}}). \tag{4.30}$$

If another data set $\tilde{\mathcal{D}}$ is available with $\tilde{n}$ samples $(\tilde{\boldsymbol{x}}_i, \tilde{y}_i) \sim p(\boldsymbol{x}, y)$ that are statistically independent from the samples in $\mathcal{D}^{\text{train}}$, we can use $\tilde{\mathcal{D}}$ to estimate the prediction loss $\mathcal{J}(\hat{h})$ via a sample average

$$\hat{\mathcal{J}}(\hat{h}; \tilde{\mathcal{D}}) = \frac{1}{\tilde{n}} \sum_{i=1}^{\tilde{n}} \mathcal{L}(\hat{h}(\tilde{\boldsymbol{x}}_i), \tilde{y}_i). \tag{4.31}$$

Depending on the context, $\tilde{\mathcal{D}}$ is called a test or a validation set.

We are typically given the union of the two data sets $\mathcal{D}^{\text{train}}$ and $\tilde{\mathcal{D}}$, and it is up to us how to split them into the two sets. Common split ratios are $n/\tilde{n} = 60/40$, $70/30$, or $80/20$. If the number of (hyper) parameters is large, it is better to increase the ratio so that more data are available for training.

While the splitting is often done randomly, particularly in classification, it is important that the different values of the target variable (e.g. the class labels) represented in a balanced way in both $\mathcal{D}^{\text{train}}$ and $\tilde{\mathcal{D}}$. Stratification methods can be used so that e.g. the classes are present in the same proportions in both $\mathcal{D}^{\text{train}}$ and $\tilde{\mathcal{D}}$.

The value of the estimated prediction loss in (4.31) may vary strongly for different hold-out data sets $\tilde{\mathcal{D}}$ unless $\tilde{n}$ is large. This is often seen as a drawback of the hold-out approach. Figure 4.7 illustrates the variability that can be introduced by randomly splitting a data set into a training set $\mathcal{D}^{\text{train}}$ and test set $\tilde{\mathcal{D}}$. Cross-validation is often used to avoid such issues.

Cross-validation

Cross-validation consists in randomly dividing the data that are available for training into K (roughly) equally-sized subset (folds) $\mathcal{D}_1, \ldots, \mathcal{D}_K$ without overlap.

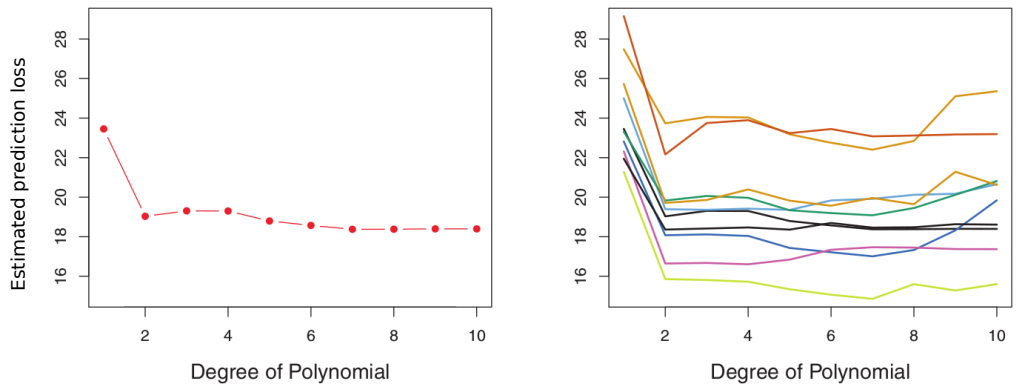


Figure 4.7: Possible variability in the estimated prediction loss. (a) The estimated prediction loss for a classification problem with a polynomial prediction model. (b) Variability induced by random splitting of the available data (392 data points) into training set and test set (here: of equal size). Each curve shows a different realisation of the random variable $\hat{\mathcal{J}}(\hat{h}; \tilde{\mathcal{D}})$. Adapted from (James, Witten, and Hastie, 2016, Figure 5.2).

For the same reasons as in the hold-out approach, we may want to use here stratification. From the folds, we construct K pairs of training data sets $\mathcal{D}_k^{\text{train}}$ and hold-out (validation) sets $\mathcal{D}_k^{\text{val}}$,

$$\mathcal{D}_k^{\text{train}} = \bigcup_{i \neq k} \mathcal{D}_i, \qquad \mathcal{D}_k^{\text{val}} = \mathcal{D}_k, \tag{4.32}$$

as illustrated in Figure 4.8. The K training sets are used to obtain K prediction functions $\hat{h}_k$,

$$\hat{h}_k = \mathcal{A}(\mathcal{D}_k^{\text{train}}), \tag{4.33}$$

whose performance $\hat{\mathcal{J}}_k$ is evaluated on the data $\mathcal{D}_k^{\text{val}}$ that was held-out during training,

$$\hat{\mathcal{J}}_k = \hat{\mathcal{J}}(\hat{h}_k; \mathcal{D}_k^{\text{val}}). \tag{4.34}$$

The performance $\hat{\mathcal{J}}(\hat{h}_k; \mathcal{D}_k^{\text{val}})$ is computed via (4.31). We are essentially repeating the hold-out approach K times, each time with different data. The cross-validation (cv) score CV is then the average of all $\hat{\mathcal{J}}_k$,

$$\text{CV} = \frac{1}{K} \sum_{i=1}^{K} \hat{\mathcal{J}}_k. \tag{4.35}$$

The cv score is sometimes used as an improved version of $\hat{\mathcal{J}}$ in (4.31). But it is actually rather an estimate of the expected prediction performance $\bar{\mathcal{J}}(\mathcal{A})$ in (4.23). The cv score does indeed not depend on a prediction function but on the prediction algorithm $\mathcal{A}$. This can be more clearly seen when writing

$$\hat{\mathcal{J}}_k = \hat{\mathcal{J}}(\hat{h}_k; \mathcal{D}_k^{\text{val}}) = \hat{\mathcal{J}}\left(\mathcal{A}(\mathcal{D}_k^{\text{train}}); \mathcal{D}_k^{\text{val}}\right) \tag{4.36}$$

so that

$$\text{CV} = \frac{1}{K} \sum_{i=1}^{K} \hat{\mathcal{J}}_k = \frac{1}{K} \sum_{i=1}^{K} \hat{\mathcal{J}}\left(\mathcal{A}(\mathcal{D}_k^{\text{train}}); \mathcal{D}_k^{\text{val}}\right), \tag{4.37}$$

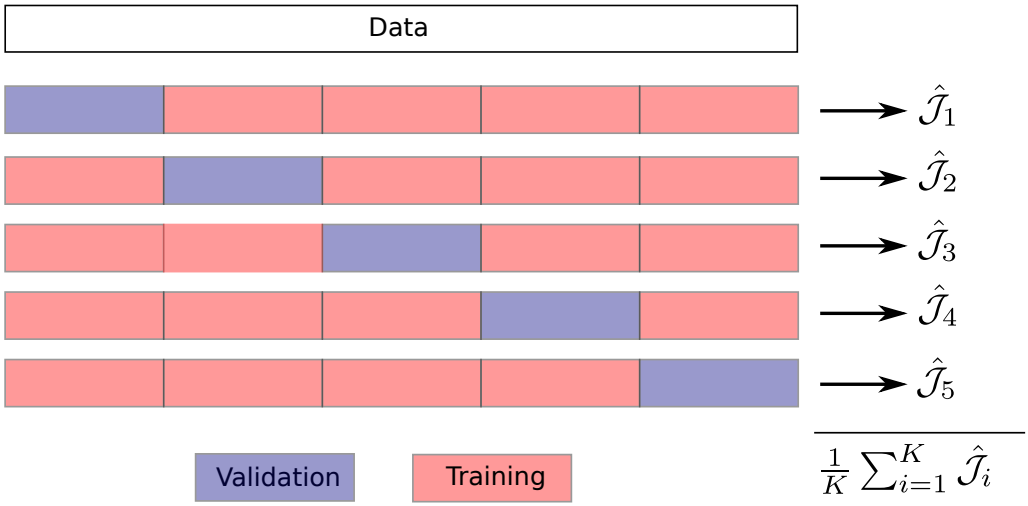


Figure 4.8: Sketch of K-fold cross-validation for $K = 5$.

which does depend on the algorithm $\mathcal{A}$ but not on a particular prediction function $\hat{h}$. The cv score is thus an estimate $\hat{\bar{\mathcal{J}}}(\mathcal{A})$ of $\bar{\mathcal{J}}(\mathcal{A})$

$$\hat{\bar{\mathcal{J}}}(\mathcal{A}) = \mathrm{CV}. \tag{4.38}$$

The cross-validation score CV, and hence the estimate of $\bar{\mathcal{J}}$, depends on the particular assignment of the data points into the K folds, so that the score is a random variable. One can assess its distribution by performing cross-validation several times but this tends to be a computationally expensive procedure.

Alternatively, if K is not too large, e.g. $K = 5$, one can assess the variability of the cv-score by estimating its variance as

$$\mathbb{V}(\mathrm{CV}) \approx \frac{1}{K}\,\mathbb{V}(\hat{\mathcal{J}}), \qquad \mathbb{V}(\hat{\mathcal{J}}) \approx \frac{1}{K}\sum_{k=1}^{K}(\hat{\mathcal{J}}_k - \mathrm{CV})^2. \tag{4.39}$$

We have here approximations because the formulae assume statistical independence of the $\hat{\mathcal{J}}_k$, which is not the case as they were all computed from the same data. The square root of $\mathbb{V}(\mathrm{CV})$ is called the standard error of the cv-score.

The value of K is a tuning parameter. A typical choice is $K = 5$, so that the training sets $\mathcal{D}_k^{\mathrm{train}}$ consist of 4/5 of all data points available and the validation sets $\mathcal{D}_k^{\mathrm{val}}$ of 1/5 of them. If the validation sets consist of one data point only, the method is called leave-one-out cross-validation (LOOCV). While generally very expensive, for some problems, the computation can be done quickly. For a further discussion of the choice of K, see e.g. Section 7.10 in the textbook by Hastie, Tibshirani, and Friedman (2009).

4.3.2 Hyperparameter selection and performance evaluation

We consider a scenario where we have several prediction models $h_{\boldsymbol{\lambda}}(\boldsymbol{x};\boldsymbol{\theta})$ that we can possibly use for solving our prediction task, and that we need to select among them. An algorithm that depends on the hyperparameters $\boldsymbol{\lambda}$ will be denoted by

$\mathcal{A}_{\boldsymbol{\lambda}}$. Two approaches to hyperparameter selection and performance evaluation of the final prediction function are presented: The first uses hold-out data to select the hyperparameters and hold-out data for performance evaluation while the second uses cross-validation for hyperparameter selection and hold-out data for performance evaluation.

Two times hold-out

This approach to hyperparameter selection and performance evaluation proceeds as follows:

1. From all the data $\mathcal{D}$ that are available to us, we split off some test data $\mathcal{D}^{\text{test}}$ to estimate the performance our final prediction function $\hat{h}$. The test data will never be touched until the final performance evaluation. A typical size of the test data is 20% of $\mathcal{D}$.

2. We split the remaining data into a training set $\mathcal{D}^{\text{train}}$ and a validation set $\mathcal{D}^{\text{val}}$, using, for example, again the 80/20 ratio ($\mathcal{D}^{\text{train}}$ contains 80% of the data that remain after the initial splitting while $\mathcal{D}^{\text{val}}$ contains 20% of them).

3. Running an algorithm with tuning parameters $\boldsymbol{\lambda}$ on $\mathcal{D}^{\text{train}}$ returns a set of functions
$$\hat{h}_{\boldsymbol{\lambda}} = \mathcal{A}_{\boldsymbol{\lambda}}(\mathcal{D}^{\text{train}}) \tag{4.40}$$
indexed by the hyperparameters $\boldsymbol{\lambda}$.

4. We evaluate the performance of $\hat{h}_{\boldsymbol{\lambda}}$ on $\mathcal{D}^{\text{val}}$ by computing the estimated prediction loss $\text{PL}(\boldsymbol{\lambda})$
$$\text{PL}(\boldsymbol{\lambda}) = \hat{J}(\hat{h}_{\boldsymbol{\lambda}}; \mathcal{D}^{\text{val}}), \tag{4.41}$$
where $\hat{J}$ is defined in (4.31). We choose $\boldsymbol{\lambda}$ by minimising $\text{PL}(\boldsymbol{\lambda})$,
$$\hat{\boldsymbol{\lambda}} = \operatorname*{argmin}_{\boldsymbol{\lambda}} \text{PL}(\boldsymbol{\lambda}). \tag{4.42}$$

5. Using $\hat{\boldsymbol{\lambda}}$, we re-estimate the parameters $\boldsymbol{\theta}$ on the union of the training and validation data $\mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{val}}$. By using more data, we can estimate the prediction model more accurately. Denote the resulting prediction function by $\hat{h}$,
$$\hat{h} = \mathcal{A}_{\hat{\boldsymbol{\lambda}}}(\mathcal{D}^{\text{train}} \cup \mathcal{D}^{\text{val}}). \tag{4.43}$$

6. We take the test data $\mathcal{D}^{\text{test}}$ out of the vault to compute an estimate $\hat{J}$ of the prediction loss of $\hat{h}$,
$$\hat{J} = \hat{J}(\hat{h}; \mathcal{D}^{\text{test}}), \tag{4.44}$$
using (4.31).

7. We re-estimate $\hat{h}$ using all data available,
$$\hat{h}(\boldsymbol{x}) = \mathcal{A}_{\hat{\boldsymbol{\lambda}}}(\mathcal{D}), \tag{4.45}$$
which provides us with the final prediction function $\hat{h}$. An estimate of its generalisation performance is given by $\hat{J}$ in (4.44).

In some cases the re-estimation needs to be skipped because of computational reasons. Optimisation over the hyperparameters $\boldsymbol{\lambda}$ is typically not possible by gradient descent. Grid search can be used if the number of hyperparameters is small. Alternative methods are random search where different values of the hyperparameters are randomly tried out (Bergstra and Bengio, 2012), or Bayesian optimisation where the functional relationship between the hyperparameters and the prediction loss is modelled via (Gaussian process) regression, which is used to guide the optimisation (e.g. Snoek, Larochelle, and Adams, 2012).

Cross-validation and hold-out

In this approach, we choose the hyperparameters by cross-validation and estimate the prediction performance by a hold-out test set. In more detail, we proceed as follows:

1. As above, from all the data $\mathcal{D}$ that are available to us, we split off some test data $\mathcal{D}^{\text{test}}$ to estimate the performance our final prediction function $\hat{h}$. The test data will never be touched until the final performance evaluation. A typical size of the test data is 20% of $\mathcal{D}$.

2. We use the remaining data, call it $\mathcal{D}^{\text{train}}$, to compute the cv-score CV as a function of the hyperparameters. The cv-score is an estimate $\hat{\bar{\mathcal{J}}}$ of the expected prediction loss $\bar{\mathcal{J}}$, see (4.38). Let us denote it by $\text{EPL}(\boldsymbol{\lambda})$,

$$\text{EPL}(\boldsymbol{\lambda}) = \text{CV} = \hat{\bar{\mathcal{J}}}(\mathcal{A}_{\boldsymbol{\lambda}}). \tag{4.46}$$

3. We choose $\hat{\boldsymbol{\lambda}}$ by minimising $\text{EPL}(\boldsymbol{\lambda})$. Since the cv-score is an estimate with standard-deviation $\sqrt{\mathbb{V}(\text{CV})}$, an alternative method is to choose the hyperparameters so that they result in the simplest model while still having a cv-score that is within one standard deviation of the minimal cv-score.

4. Using $\hat{\boldsymbol{\lambda}}$, we re-estimate the parameters $\boldsymbol{\theta}$ from $\mathcal{D}^{\text{train}}$. Denote the resulting prediction function by $\hat{h}$,

$$\hat{h} = \mathcal{A}_{\hat{\boldsymbol{\lambda}}}(\mathcal{D}^{\text{train}}). \tag{4.47}$$

5. We take the test data $\mathcal{D}^{\text{test}}$ out of the vault to compute an estimate $\hat{\mathcal{J}}$ of the prediction loss of $\hat{h}$,

$$\hat{\mathcal{J}} = \hat{\mathcal{J}}(\hat{h}; \mathcal{D}^{\text{test}}), \tag{4.48}$$

   using (4.31).

6. We re-estimate $\hat{h}$ using all data available,

$$\hat{h} = \mathcal{A}_{\hat{\boldsymbol{\lambda}}}(\mathcal{D}), \tag{4.49}$$

   which provides us with the final prediction function $\hat{h}$. An estimate of its generalisation performance is given by $\hat{\mathcal{J}}$ in (4.48).

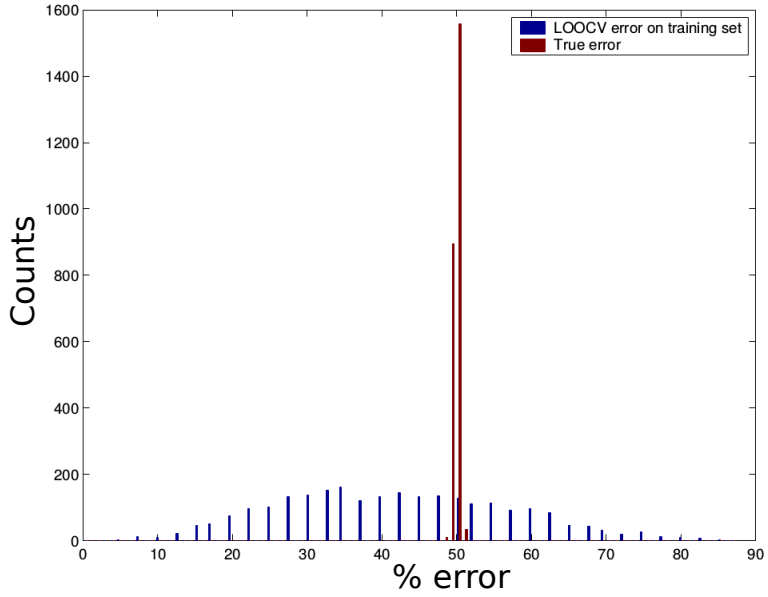


Figure 4.9: Distribution of the minimal cv-score (blue) and true prediction losses (prediction errors, red) for several artificially generated classification problems where the true prediction error is 0.5. Sample size was 40 and classification was done by support-vector machines. The figure is from Varma and Simon (2006), Figure 2.

In some cases the re-estimation needs to be skipped because of computational reasons. Minimisation of the cv-score can typically not be done by gradient descent. As before, gradient-free minimisation methods such as grid search, random search, or Bayesian optimisation can be used.

Like a training loss, the minimal cv-score is typically an optimistic estimate of the prediction loss because the hyperparameters are chosen such that the cv-score is minimised. The prediction loss tends to be underestimated as illustrated in Figure 4.9. That is why we need the hold-out test data $\mathcal{D}^{\text{test}}$ to determine the generalisation performance.

4.4 Loss functions in predictive modelling

This section provides a brief overview of loss functions that are widely used in regression and classification.

4.4.1 Loss functions in regression

Typical loss functions L in regression are

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 \qquad \text{(square loss)} \tag{4.50}$$

$$L(\hat{y}, y) = |\hat{y} - y| \qquad \text{(absolute loss)} \tag{4.51}$$

$$L(\hat{y}, y) = \begin{cases} \frac{1}{2}(\hat{y} - y)^2 & \text{if } |\hat{y} - y| < \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \qquad \text{(Huber loss)} \tag{4.52}$$

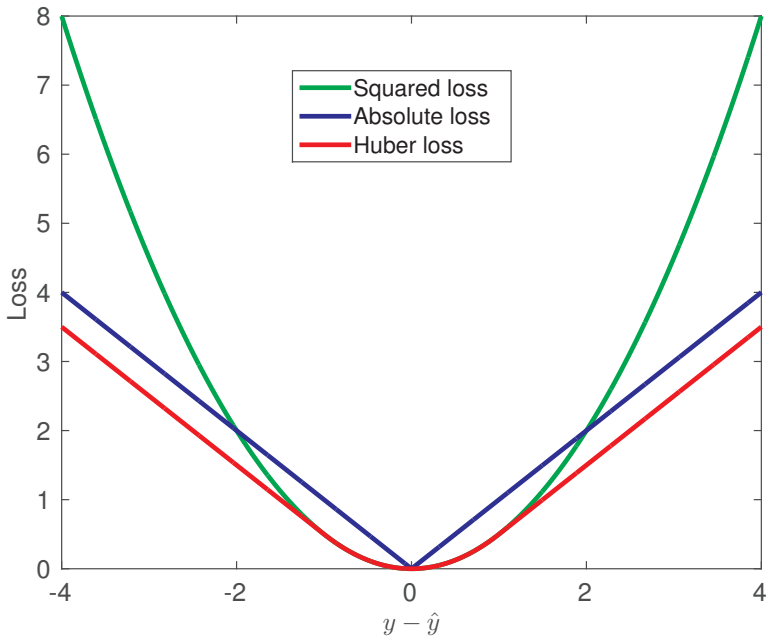


Figure 4.10: Loss functions that are often used in regression.

Figure 4.10 shows plots of the different loss functions. The absolute loss is more robust than the square loss since it does not grow as quickly, but it is not differentiable when the residual $\hat{y} - y$ is zero. The Huber loss combines the good properties of the square and the absolute loss.

4.4.2 Loss functions in classification

We distinguish between loss functions that differentiable with respect to parameters of the classifier and those which are not.

Non-differentiable loss functions

We assume here that y and $\hat{y}$ can take K different values, for instance $\{1, \ldots, K\}$. This corresponds to classification with K different classes. The loss function $L(\hat{y}, y)$ can then be represented as a $K \times K$ matrix $\boldsymbol{L}$,

$$\boldsymbol{L} = \begin{pmatrix} L(1,1) & L(1,2) & \cdots & L(1,K) \\ L(2,1) & L(2,2) & \cdots & L(2,K) \\ \vdots & \vdots & & \vdots \\ L(K,1) & L(K,2) & \cdots & L(K,K) \end{pmatrix}. \tag{4.53}$$

The diagonal elements $L(i,i)$ are zero as they correspond to correct predictions. The off-diagonal elements $L(i,j)$ are positive; they correspond to the loss incurred when predicting i instead of j. Since $\hat{y}$ takes on discrete values, we cannot compute derivatives with respect to parameters $\boldsymbol{\theta}$ that might govern the classifier.

If $L(i,j) = 1$ for $i \neq j$ and zero otherwise, the loss is said to be the zero-one

loss. Its expectation $\mathcal{J}(h)$ equals

$$\begin{aligned}
\mathcal{J}(h) &= \mathbb{E}_{\boldsymbol{x},y}\, L(h(\boldsymbol{x}), y) && (4.54)\\
&= \mathbb{E}_{\hat{y},y}\, L(\hat{y}, y) && (4.55)\\
&= \sum_{i,j} L(i,j)p(i,j) && (4.56)\\
&= \sum_{i \neq j} p(i,j) && (4.57)\\
&= \mathbb{P}\left(y \neq \hat{y}\right), && (4.58)
\end{aligned}$$

which is the misclassification or error rate. The term $p(i,j) = \mathbb{P}(\hat{y} = i, y = j)$ denotes the joint probability of $(\hat{y}, y)$. The joint probability of $(\hat{y}, y)$ is induced by the joint probability of $(\boldsymbol{x}, y)$ and the prediction function h. The $p(i,j)$ for $i \neq j$ indicate the probabilities that h wrongly predicts i if the true class is j. We generally want h to be such that these probabilities are small.

If there are only two classes, for example $\{-1, 1\}$, the random variables $(\hat{y}, y)$ can take four possible values and the predictions are typically called "true positive", "false negative", "false positive", or "true negative", see Table 4.1. The possible conditional probabilities $p(\hat{y}|y)$ are:

$$\begin{aligned}
&\text{true-positive rate of } h\text{:} && \mathbb{P}(\hat{y} = 1 | y = 1) && (4.59)\\
&\text{true-negative rate of } h\text{:} && \mathbb{P}(\hat{y} = -1 | y = -1) && (4.60)\\
&\text{false-positive rate of } h && \mathbb{P}(\hat{y} = 1 | y = -1) = 1 - \text{true-negative rate} && (4.61)\\
&\text{false-negative rate of } h\text{:} && \mathbb{P}(\hat{y} = -1 | y = 1) = 1 - \text{true-positive rate} && (4.62)
\end{aligned}$$

The probabilities all depend on h since $\hat{y} = h(\boldsymbol{x})$. The true-positive rate is also called sensitivity, hit rate, or recall. Another name for the true-negative rate is specificity. The false-positive rate is the probability that h says wrongly "1". It is also called the type 1 error. The false-negative rate is the probability that h says wrongly "-1". It is also called the type 2 error. While the true-positive and true-negative rates correspond to the benefits of h, the false-positive and false-negative rates correspond to the costs associated with using h.

The loss function $L(\hat{y}, y)$ can be defined such that $\mathcal{J}(h)$ penalises false-positive and false-negative rates. If we let

$$\boldsymbol{L} = \begin{pmatrix} 0 & \frac{1}{\mathbb{P}(y=-1)} \\ \frac{1}{\mathbb{P}(y=1)} & 0 \end{pmatrix} \tag{4.63}$$

the expected loss equals the sum of the false-positive and the false-negative rate:

$$\begin{aligned}
\mathcal{J}(h) = &= \mathbb{E}_{\boldsymbol{x},y}\, L(h(\boldsymbol{x}), y) && (4.64)\\
&= \mathbb{E}_{\hat{y},y}\, L(\hat{y}, y) && (4.65)\\
&= \sum_{i,j} L(i,j)p(i,j) && (4.66)\\
&= \frac{p(1,-1)}{\mathbb{P}(y=-1)} + \frac{p(-1,1)}{\mathbb{P}(y=1)} && (4.67)\\
&= \mathbb{P}(\hat{y} = 1 | y = -1) + \mathbb{P}(\hat{y} = -1 | y = 1) && (4.68)
\end{aligned}$$

$\hat{y}$	y	meaning	probability	shorthand notation
1	1	true positive	$\mathbb{P}(\hat{y}=1, y=1)$	$p(1,1)$
1	-1	false positive	$\mathbb{P}(\hat{y}=1, y=-1)$	$p(1,-1)$
-1	1	false negative	$\mathbb{P}(\hat{y}=-1, y=1)$	$p(-1,1)$
-1	-1	true negative	$\mathbb{P}(\hat{y}=-1, y=-1)$	$p(-1,-1)$

Table 4.1: Possible events and their probabilities in binary classification.

Such a cost function can be advantageous over the misclassification rate if there is, for instance, an imbalance between the probabilities for $y=1$ and $y=-1$.

Minimising the false-positive (or false-negative) rate alone is not be very meaningful strategy: The reason is that the trivial classifier $h(\boldsymbol{x}) = \hat{y} = -1$ would be the optimal solution. But for such a classifier the true-positive rate would be zero. There is generally a trade-off between true-positive and false-positive rates. This trade-off can be visualised by plotting the false-positive rate, or "cost" of h versus the true-positive rate, or "benefit" of h, see Figure 4.11. Such a plot is said to visualise the classifier in the "ROC space", where ROC stands for "receiver operating characteristic".

For classifiers or models with a hyperparameter, the performance of the classifier in the ROC space traces out a curve as the value of the hyperparameter is changed. The curve can be used for hyperparameter selection because classifiers that are located closest to the upper-left corner have the best trade-off between true-positive and false-positive rate. Classifiers that are located on a line parallel to the diagonal trade a better true-positive rate against a larger false-positive rate. We may consider such classifiers to be equivalent and the choice of working with one rather than the other is problem dependent. The area under the curve in the ROC space can be used to compare two classifiers or models irrespective of the value of the hyperparameter.

Differentiable loss functions in classification

For simplicity, we consider here binary classification only. Let us assume that $\hat{y} \in \{-1, 1\}$ is given by

$$\hat{y}(\boldsymbol{x}) = \text{sign}(h(\boldsymbol{x})) \tag{4.69}$$

where $h(\boldsymbol{x})$ is real-valued.

An input $\boldsymbol{x}$ gets correctly classified if $h(\boldsymbol{x})$ takes positive values for $y=1$ and negative values for $y=-1$. That is,

$$\text{correct classification of } \boldsymbol{x} \iff yh(\boldsymbol{x}) > 0.$$

The quantity $yh(\boldsymbol{x})$ is called the margin and it plays a similar role as the residual $y - h(\boldsymbol{x})$ in regression. The zero-one loss introduced above can be obtained by operating on the margin rather than on $\hat{y}$. Indeed, the zero-one loss is obtained for

$$L(h(\boldsymbol{x}), y) = \begin{cases} 1 & \text{if } yh(\boldsymbol{x}) < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.70}$$

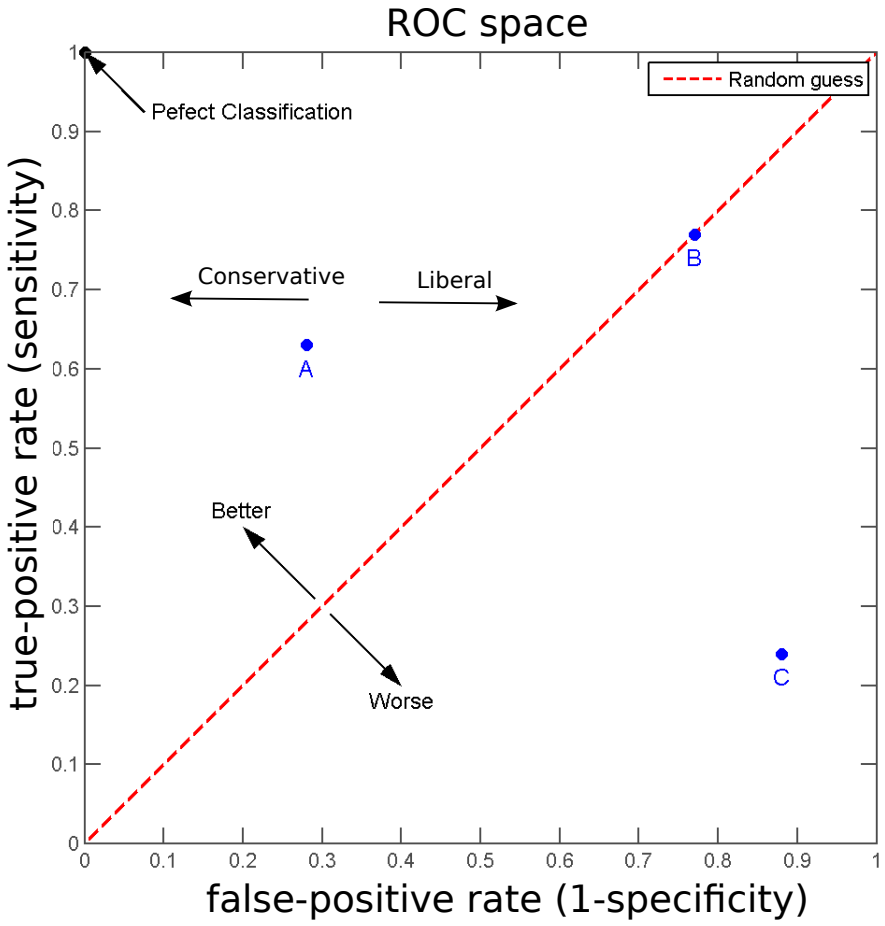


Figure 4.11: Plotting the false-positive rate ("cost") of a classifier versus its true-positive rate ("benefit"). Classifier B is obtained by setting $\hat{y} = 1$ with probability 0.8 irrespective of the data. Classifier A takes advantage of the data and its benefit outweighs its cost while classifier C incurs a larger cost than benefit. Adapted from https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Several loss functions operate on the margin $yh(\boldsymbol{x})$. Typical ones are:

$$L(h(\boldsymbol{x}), y) = (h(\boldsymbol{x}) - y)^2 = (1 - yh(\boldsymbol{x}))^2 \quad \text{(square loss)} \tag{4.71}$$

$$L(h(\boldsymbol{x}), y) = \log(1 + \exp(-yh(\boldsymbol{x}))) \quad \text{(log loss)} \tag{4.72}$$

$$L(h(\boldsymbol{x}), y) = \exp(-yh(\boldsymbol{x})) \quad \text{(exponential loss)} \tag{4.73}$$

$$L(h(\boldsymbol{x}), y) = \max(0, 1 - yh(\boldsymbol{x})) \quad \text{(hinge loss)} \tag{4.74}$$

$$L(h(\boldsymbol{x}), y) = \max(0, 1 - yh(\boldsymbol{x}))^2 \quad \text{(square hinge loss)} \tag{4.75}$$

$$L(h(\boldsymbol{x}), y) = \begin{cases} -4yh(\boldsymbol{x}) & \text{if } yh(\boldsymbol{x}) < -1 \\ \max(0, 1 - yh(\boldsymbol{x}))^2 & \text{otherwise} \end{cases} \quad \text{(Huberised square hinge loss)}$$

(Hastie, Tibshirani, and Friedman, 2009, Section 10.6 and Table 12.1). The different loss functions are visualised in Figure 4.12. Unlike the standard hinge loss, the square hinge loss is differentiable everywhere. The remaining loss functions are differentiable with respect to h, so that a smoothly parametrised model $h(\boldsymbol{x}; \boldsymbol{\theta})$ can be optimised by gradient-based optimisation methods. The different loss functions can be considered to approximate the zero-one loss. Most of them assign a loss to small positive margins, thus encouraging more confident decisions about the label. The square loss function is both sensitive to outliers and penalises large (positive) margins, which can be seen as a key disadvantage of the loss function.

Minimising the log-loss over a sample of n data points $(\boldsymbol{x}_i, y_i)$, drawn from $p(\boldsymbol{x}, y)$, is equivalent to maximising the log-likelihood in logistic regression. In logistic regression, we model the conditional probabilities of $y|\boldsymbol{x}$ as

$$\mathbb{P}(y = 1|\boldsymbol{x}; h) = \frac{1}{1 + \exp(-h(\boldsymbol{x}))} \qquad \mathbb{P}(y = -1|\boldsymbol{x}; h) = \frac{1}{1 + \exp(h(\boldsymbol{x}))} \tag{4.76}$$

and estimate h by maximising the log-likelihood

$$\ell(h) = \sum_{\boldsymbol{x}_i : y_i = 1} \log \mathbb{P}(y_i = 1|\boldsymbol{x}_i; h) + \sum_{\boldsymbol{x}_i : y_i = -1} \log \mathbb{P}(y_i = -1|\boldsymbol{x}_i; h) \tag{4.77}$$

$$= -\sum_{\boldsymbol{x}_i : y_i = 1} \log\left(1 + \exp(-h(\boldsymbol{x}_i))\right) - \sum_{\boldsymbol{x}_i : y_i = -1} \log\left(1 + \exp(h(\boldsymbol{x}_i))\right) \tag{4.78}$$

$$= -\sum_{\boldsymbol{x}_i} \log\left(1 + \exp(-y_i h(\boldsymbol{x}_i))\right). \tag{4.79}$$

We can see that $\ell(h)$ is n times the negated sample average of the log loss.

References

[1] J. Bergstra and Y. Bengio. "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305.

[2] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[3] G. James, D. Witten, and T. Hastie. *An Introduction to Statistical Learning: with Applications in R*. 6th ed. Springer, 2016.

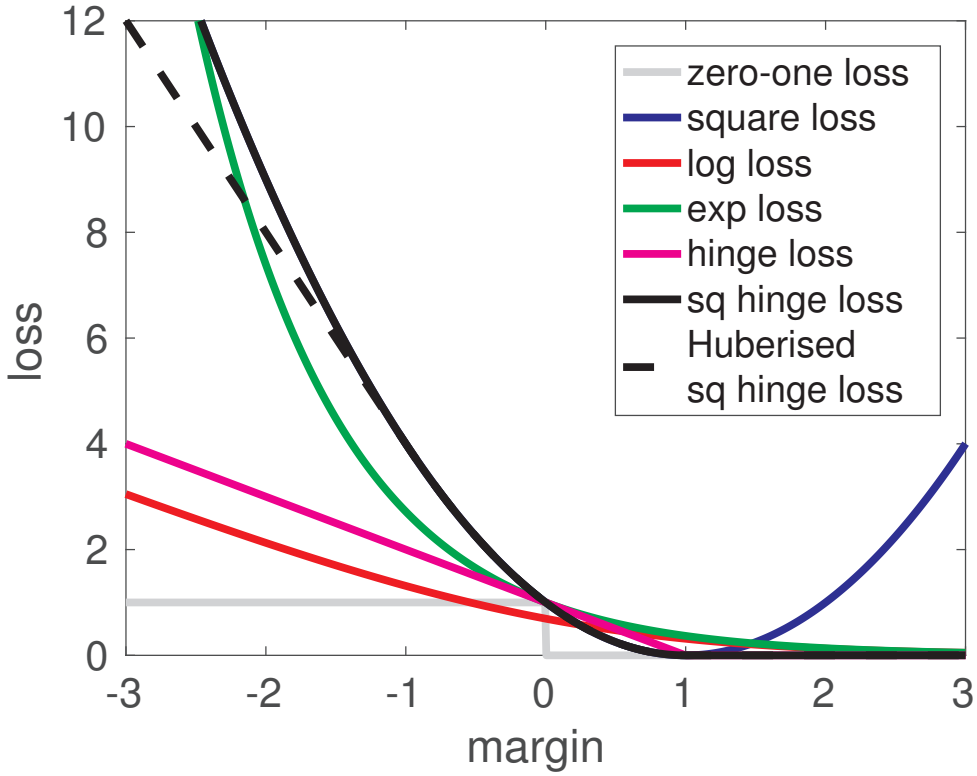


Figure 4.12: Loss functions that are often used in classification.

[4] J. Snoek, H. Larochelle, and R.P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems (NIPS)*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 2951–2959.

[5] S. Varma and R. Simon. "Bias in error estimation when using cross-validation for model selection". In: *BMC Bioinformatics* 7.91 (2006).

Appendix A

Linear algebra

The material in this chapter is mostly a refresher of some basic results from linear algebra. But it also contains some proofs of results that may be harder to find. The proofs are not examinable.

A.1 Matrices

A $m \times n$ matrix $\boldsymbol{A}$ is a $m \times n$ array of numbers arranged into m rows and n columns. The element at row i and column j is denoted by a_{ij} so that

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}. \tag{A.1}$$

We will sometimes use the indexing notation $(\boldsymbol{A})_{ij}$ to refer to element a_{ij}. The transpose $\boldsymbol{A}^\top$ of a matrix $\boldsymbol{A}$ is the matrix where the entries of $\boldsymbol{A}$ are mirrored at the diagonal, i.e. $(\boldsymbol{A}^\top)_{ij} = (\boldsymbol{A})_{ji}$. If $\boldsymbol{A}^\top = \boldsymbol{A}$, the matrix is said to be symmetric.

Multiplying a matrix with a scalar produces a matrix where each element is scaled by said scalar, for example

$$\alpha\boldsymbol{A} = \begin{pmatrix} \alpha a_{11} & \alpha a_{12} & \dots & \alpha a_{1n} \\ \vdots & & & \vdots \\ \alpha a_{m1} & \alpha a_{m2} & \dots & \alpha a_{mn}. \end{pmatrix} \tag{A.2}$$

Two matrices of the same size can be added together by adding their corresponding elements, for example

$$\boldsymbol{A} + \boldsymbol{B} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ \vdots & & & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix} \tag{A.3}$$

$$= \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1n} + b_{1n} \\ \vdots & & & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \dots & a_{mn} + b_{mn}. \end{pmatrix} \tag{A.4}$$

If matrix $\boldsymbol{A}$ has size $m \times n$ and matrix $\boldsymbol{B}$ size $n \times p$, the two matrices can be multiplied together. The results is a $m \times p$ matrix $\boldsymbol{C} = \boldsymbol{AB}$ whose elements $(\boldsymbol{C})_{ij} = c_{ij}$ are given by

$$(\boldsymbol{C})_{ij} = \sum_{k=1}^{n} (\boldsymbol{A})_{ik}(\boldsymbol{B})_{kj}, \qquad \text{or, equivalently,} \qquad c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}. \tag{A.5}$$

The equations mean that to compute the (ij)-th element of $\boldsymbol{C}$, we multiply the elements of the i-th row of $\boldsymbol{A}$ with the elements of the j-th column of $\boldsymbol{B}$ and sum them all up.

The trace of a $m \times m$ matrix $\boldsymbol{A}$ is the sum of its diagonal elements,

$$\text{trace}(\boldsymbol{A}) = \sum_{i=1}^{m} a_{ii}. \tag{A.6}$$

The trace of $\boldsymbol{AB}$ equals the trace of $\boldsymbol{BA}$: Let $\boldsymbol{A}$ be $m \times n$ and $\boldsymbol{B}$ $n \times m$. We then have

$$\text{trace}(\boldsymbol{AB}) = \sum_{i=1}^{m} (\boldsymbol{AB})_{ii} = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij}b_{ji} \right) = \sum_{j=1}^{n} \sum_{i=1}^{m} b_{ji}a_{ij}, \tag{A.7}$$

which equals $\sum_{j=1}^{n} (\boldsymbol{BA})_{jj}$ and hence

$$\text{trace}(\boldsymbol{AB}) = \text{trace}(\boldsymbol{BA}) \tag{A.8}$$

as claimed.

A.2 Vectors

A n-dimensional vectors $\boldsymbol{v}$ can be seen as $n \times 1$ matrix. We denote its i-th element by v_i or sometimes also by $(\boldsymbol{v})_i$. By default, $\boldsymbol{v}$ is a column vector, i.e.

$$\boldsymbol{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}. \tag{A.9}$$

It's transpose $\boldsymbol{v}^\top$ is the row vector $(v_1, \ldots, v_n)$. Like matrices, vectors can be scaled, added or multiplied together. The product between a $1 \times n$ vector $\boldsymbol{u}$ and a $n \times 1$ vector $\boldsymbol{v}$ is with (A.5) a number equal to

$$\boldsymbol{uv} = \sum_{i=1}^{n} u_i v_i. \tag{A.10}$$

The inner product or scalar product $\boldsymbol{u}^\top \boldsymbol{v}$ between two n dimensional vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ is

$$\boldsymbol{u}^\top \boldsymbol{v} = \sum_{i=1}^{n} u_i v_i, \tag{A.11}$$

that is, the vector $\boldsymbol{u}$ is first transposed to be row vector after which (A.5) is applied. Importantly, it does not matter whether $\boldsymbol{u}$ or $\boldsymbol{v}$ is transposed, i.e.

$$\boldsymbol{u}^\top \boldsymbol{v} = \boldsymbol{v}^\top \boldsymbol{u}. \tag{A.12}$$

The outer product $\boldsymbol{u}\boldsymbol{v}^\top$ between a m dimensional vector $\boldsymbol{u}$ and a n dimensional vector $\boldsymbol{v}$ is a $m \times n$ matrix

$$\boldsymbol{u}\boldsymbol{v}^\top = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix} \begin{pmatrix} v_1 & v_2 & \dots & v_n \end{pmatrix} = \begin{pmatrix} u_1 v_1 & u_1 v_2 & \dots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \dots & u_2 v_n \\ \vdots & \vdots & & \vdots \\ u_m v_1 & u_m v_2 & \dots & u_m v_n \end{pmatrix}. \tag{A.13}$$

It can be seen that the (i, j)-th element of the matrix is equal to $u_i v_j$ in line with (A.5).

Equation (A.5) also tells us that the product between a $m \times n$ matrix $\boldsymbol{A}$ and n-dimensional vector $\boldsymbol{v}$ equals a m-dimensional vector $\boldsymbol{v}$ with elements v_i,

$$v_i = \sum_{j=1}^{n} a_{ij} u_j \qquad i = 1, \dots, m. \tag{A.14}$$

A.3 Matrix operations as operations on column vectors

It is often helpful to consider a $m \times n$ matrix $\boldsymbol{A}$ as a collection of n column vectors $\boldsymbol{a}_j$ of dimension m that are arranged next to each other,

$$\boldsymbol{A} = (\boldsymbol{a}_1, \dots, \boldsymbol{a}_n). \tag{A.15}$$

Note that the i-th element of the j-th column of $\boldsymbol{A}$ is $(\boldsymbol{A})_{ij} = (\boldsymbol{a}_j)_i$.

A.3.1 Matrix-vector products

By computing the i-th element, we see that $\boldsymbol{v} = \boldsymbol{A}\boldsymbol{u}$ can be written as weighted combination of the column vectors $\boldsymbol{a}_j$,

$$\boldsymbol{A}\boldsymbol{u} = \sum_{j=1}^{n} \boldsymbol{a}_j u_j = \underbrace{\begin{pmatrix} a_{11} \\ \vdots \\ a_{m1} \end{pmatrix}}_{\boldsymbol{a}_1} u_1 + \dots + \underbrace{\begin{pmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{pmatrix}}_{\boldsymbol{a}_j} u_j + \dots + \underbrace{\begin{pmatrix} a_{1n} \\ \vdots \\ a_{mn} \end{pmatrix}}_{\boldsymbol{a}_n} u_n, \tag{A.16}$$

The equation shows that for vectors $\boldsymbol{u}$ that are zero everywhere but in slot k, $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{a}_k u_k$, which means that we can "pick" column k of $\boldsymbol{A}$ by multiplication with the k unit vector.

A.3.2 Matrix-matrix products

Products between matrices can also be written in terms of operations on the column vectors. Let $\boldsymbol{B}$ be a $n \times p$ matrix with column vectors $\boldsymbol{b}_i \in \mathbb{R}^n$,

$$\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_p). \tag{A.17}$$

By computing the (i, j)-th element, we see that $\boldsymbol{AB}$ can be written as a collection of column vectors $\boldsymbol{Ab}_j$,

$$\boldsymbol{AB} = (\boldsymbol{Ab}_1, \ldots, \boldsymbol{Ab}_p). \tag{A.18}$$

Indeed, the i-th element of the j-th column is $(\boldsymbol{Ab}_j)_i$ and

$$(\boldsymbol{Ab}_j)_i = \sum_{k=1}^{n} (\boldsymbol{A})_{ik} (\boldsymbol{b}_j)_k = \sum_{k=1}^{n} (\boldsymbol{A})_{ik} (\boldsymbol{B})_{kj}, \tag{A.19}$$

which equals $(\boldsymbol{AB})_{ij}$.

Assume for a moment that matrix $\boldsymbol{B}$ is zero everywhere but in a $r \times r$ block in the upper left,

$$\boldsymbol{B} = \left(\begin{array}{ccc} b_1 & & \\ & \ddots & \mathbf{0} \\ & b_r & \\ \hdashline \mathbf{0} & & \mathbf{0} \end{array}\right) \tag{A.20}$$

That is, the first r column vectors $\boldsymbol{b}_j$ are zero everywhere but in slot j where they equal b_j, i.e. $\boldsymbol{b}_1 = (b_1, 0, \ldots)^\top$, $\boldsymbol{b}_2 = (0, b_2, 0, \ldots)^\top$ and so on, and the remaining column vectors $\boldsymbol{b}_{r+1}, \ldots, \boldsymbol{b}_p$ are all zero. From (A.18) and (A.16), it follows that

$$\boldsymbol{AB} = (b_1 \boldsymbol{a}_1, b_2 \boldsymbol{a}_2, \ldots, b_r \boldsymbol{a}_r, \mathbf{0}, \ldots, \mathbf{0}). \tag{A.21}$$

This shows that we can weigh each column vector of the matrix $\boldsymbol{A}$, or set it to zero, by multiplying it with a matrix that is zero everywhere but in the first r diagonal elements.

A.3.3 Outer product representation of a matrix-matrix product

Assume we want to compute the matrix product $\boldsymbol{AB}^\top$ where $\boldsymbol{A}$ is $m \times n$ as before but $\boldsymbol{B}$ is $p \times n$. Let us denote the n columns of $\boldsymbol{B}$ by $\boldsymbol{b}_j \in \mathbb{R}^p$,

$$\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n). \tag{A.22}$$

From (A.5), we know that

$$(\boldsymbol{AB}^\top)_{ij} = \sum_{k=1}^{n} (\boldsymbol{A})_{ik} (\boldsymbol{B}^\top)_{kj} = \sum_{k=1}^{n} (\boldsymbol{A})_{ik} (\boldsymbol{B})_{jk} \tag{A.23}$$

We now show that $\boldsymbol{AB}^\top$ can also be written as sum of outer products between the column vectors of $\boldsymbol{A}$ and $\boldsymbol{B}$,

$$\boldsymbol{AB}^\top = \sum_{k=1}^{n} \boldsymbol{a}_k \boldsymbol{b}_k^\top. \tag{A.24}$$

This identity can be verified by computing the (i, j)-th element of the matrix on the right-hand-side:

$$\left(\sum_{k=1}^{n} \boldsymbol{a}_k \boldsymbol{b}_k^\top\right)_{i,j} = \sum_{k=1}^{n} \left(\boldsymbol{a}_k \boldsymbol{b}_k^\top\right)_{i,j} \tag{A.25}$$

$$= \sum_{k=1}^{n} (\boldsymbol{a}_k)_i (\boldsymbol{b}_k)_j. \tag{A.26}$$

Since $(\boldsymbol{a}_k)_i$ is the i-th element of the k-th column of $\boldsymbol{A}$, we have $(\boldsymbol{a}_k)_i = (\boldsymbol{A})_{ik}$. For the same reason, $(\boldsymbol{b}_k)_j = (\boldsymbol{B})_{jk}$, so that $(\boldsymbol{a}_k)_i (\boldsymbol{b}_k)_j = (\boldsymbol{A})_{ik} (\boldsymbol{B})_{jk}$ and

$$\left(\sum_{k=1}^{n} \boldsymbol{a}_k \boldsymbol{b}_k^\top\right)_{i,j} = \sum_{k=1}^{n} (\boldsymbol{A})_{ik} (\boldsymbol{B})_{jk}, \tag{A.27}$$

which equals (A.23) and thus proves the identity in (A.24).

A.4 Orthogonal basis

Two vectors $\boldsymbol{u}_1 \in \mathbb{R}^n$ and $\boldsymbol{u}_2 \in \mathbb{R}^n$ are said to be orthogonal if their inner product (scalar product) $\boldsymbol{u}_1^\top \boldsymbol{u}_2$ is zero. If additionally the vectors are of unit norm,

$$||\boldsymbol{u}_i|| = \sqrt{\boldsymbol{u}_i^\top \boldsymbol{u}_i}, \qquad i = 1, 2, \tag{A.28}$$

the vectors are said to be orthonormal. A set of n orthonormal vectors $\boldsymbol{u}_i \in \mathbb{R}^n$ forms an orthogonal basis of $\mathbb{R}^n$. This means that any vector $\boldsymbol{x} \in \mathbb{R}^n$ can be written as a weighted combinations of the $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$,

$$\boldsymbol{x} = \sum_{i=1}^{n} c_i \boldsymbol{u}_i. \tag{A.29}$$

The weights c_i are the coordinates of $\boldsymbol{x}$ with respect to the basis. Due to the orthogonality of the $\boldsymbol{u}_i$, the coordinates c_i can computed via an inner product between the $\boldsymbol{u}_i$ and $\boldsymbol{x}$,

$$c_i = \boldsymbol{u}_i^\top \boldsymbol{x}, \quad i = 1, \ldots, n, \tag{A.30}$$

We can form a matrix $\boldsymbol{U}$ by putting all the orthonormal basis vectors next to each other as the columns of the matrix,

$$\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n). \tag{A.31}$$

The matrix $\boldsymbol{U}$ is said to be an orthogonal matrix. Since the vectors $\boldsymbol{u}_i$ have unit norm and are orthogonal to each other, we have that $\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}_n$ where $\boldsymbol{I}_n$ is the n-dimensional identity matrix.

Collecting all coordinates c_i into the vector $\boldsymbol{c} = (c_1, \ldots, c_n)^\top$, we have with (A.30)

$$\boldsymbol{c} = \boldsymbol{U}^\top \boldsymbol{x}. \tag{A.32}$$

With (A.16), we can similarly write (A.29) more compactly as

$$\boldsymbol{x} = \boldsymbol{U}\boldsymbol{c}. \tag{A.33}$$

It follows that $\boldsymbol{x} = \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}$, from where we see that not only $\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}_n$ but also $\boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}_n$ for orthogonal matrices $\boldsymbol{U}$.

A.5 Subspaces

An orthogonal basis $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ enables us to represent any vector $\boldsymbol{x} \in \mathbb{R}^n$ as a weighted combination of the vectors. If we do not have n orthonormal vectors but only k of them, e.g. $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$, we cannot represent all n-dimensional vectors but only those vectors $\boldsymbol{z} \in \mathbb{R}^n$ that can be written as

$$\boldsymbol{z} = \sum_{i=1}^{k} a_i \boldsymbol{u}_i, \qquad a_i \in \mathbb{R}. \tag{A.34}$$

This set of vectors is said to be spanned by the $\boldsymbol{u}_1, \ldots \boldsymbol{u}_k$ and denoted by $\text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$. In other words,

$$\text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k) = \{\boldsymbol{z} \in \mathbb{R}^n : \boldsymbol{z} = \sum_{i=1}^{k} a_i \boldsymbol{u}_i\}. \tag{A.35}$$

If $\boldsymbol{z}_1 \in \text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$ and $\boldsymbol{z}_2 \in \text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$, i.e. if

$$\boldsymbol{z}_1 = \sum_{i=1}^{k} a_i \boldsymbol{u}_i, \qquad \boldsymbol{z}_2 = \sum_{i=1}^{k} b_i \boldsymbol{u}_i, \tag{A.36}$$

their weighted sum $\alpha \boldsymbol{z}_1 + \beta \boldsymbol{z}_2$ equals

$$\alpha \boldsymbol{z}_1 + \beta \boldsymbol{z}_2 = \sum_{i=1}^{k} \alpha a_i \boldsymbol{u}_i + \sum_{i=1}^{k} \beta b_i \boldsymbol{u}_i = \sum_{i=1}^{k} (\alpha a_i + \beta b_i) \boldsymbol{u}_i \tag{A.37}$$

and thus belongs to $\text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$ as well. This means that the span is closed under addition and scalar multiplication, which makes it a subspace of $\mathbb{R}^n$. Since any vector $\boldsymbol{z}$ of $\text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$ can be expressed using k coordinates only, namely the $\boldsymbol{u}_i^\top \boldsymbol{z}, i = 1, \ldots k$, $\text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$ is a k-dimensional subspace of $\mathbb{R}^n$.

We now show that any vector $\boldsymbol{x} \in \mathbb{R}^n$ can be split into a part $\boldsymbol{x}_\parallel$ that belongs to $\text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$ and a part $\boldsymbol{x}_\perp$ that belongs to $\text{span}(\boldsymbol{u}_{k+1}, \ldots \boldsymbol{u}_n)$, the span of the remaining basis vectors $\boldsymbol{u}_{k+1}, \ldots, \boldsymbol{u}_n$. Since

$$\boldsymbol{x} = \sum_{j=1}^{n} \boldsymbol{u}_j c_j = \sum_{j=1}^{k} \boldsymbol{u}_j c_j + \sum_{j=k+1}^{n} \boldsymbol{u}_j c_j \tag{A.38}$$

we have that

$$\boldsymbol{x} = \boldsymbol{x}_\parallel + \boldsymbol{x}_\perp, \qquad \boldsymbol{x}_\parallel = \sum_{j=1}^{k} \boldsymbol{u}_j c_j, \qquad \boldsymbol{x}_\perp = \sum_{j=k+1}^{n} \boldsymbol{u}_j c_j. \tag{A.39}$$

As $\boldsymbol{x}_\parallel$ is a weighted sum of the $\boldsymbol{u}_1, \ldots \boldsymbol{u}_k$, and $\boldsymbol{x}_\perp$ a weighted sum of the $\boldsymbol{u}_{k+1}, \ldots \boldsymbol{u}_n$, the vectors $\boldsymbol{x}_\parallel$ and $\boldsymbol{x}_\perp$ are orthogonal to each other. The subspace $\text{span}(\boldsymbol{u}_{k+1}, \ldots \boldsymbol{u}_n)$ is said to be orthogonal to $\text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$ and is thus also denoted by $\text{span}(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)^\perp$.

A.6 Orthogonal projections

Let us collect the k vectors $\boldsymbol{u}_k$ into the $n \times k$ matrix $\boldsymbol{U}_k$,

$$\boldsymbol{U}_k = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k). \tag{A.40}$$

Since the $\boldsymbol{u}_k$ are orthonormal, $\boldsymbol{U}_k^\top \boldsymbol{U}_k = \boldsymbol{I}_k$, but, unlike for orthogonal matrices, $\boldsymbol{U}_k \boldsymbol{U}_k^\top$ is not the identity matrix. We next show that $\boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{x}$ equals the part $\boldsymbol{x}_\parallel$ of $\boldsymbol{x}$ that belongs to the k-dimensional subspace span$(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$.

This can be most easily seen by writing $\boldsymbol{U}_k \boldsymbol{U}_k^\top$ as a sum of elementary matrices $\boldsymbol{u}_i \boldsymbol{u}_i^\top$,

$$\boldsymbol{U}_k \boldsymbol{U}_k^\top = \sum_{i=1}^{k} \boldsymbol{u}_i \boldsymbol{u}_i^\top, \tag{A.41}$$

which we can do according to (A.24). Applying $\boldsymbol{U}_k \boldsymbol{U}_k^\top$ on a vector $\boldsymbol{x}$ thus gives

$$\boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{x} \overset{\text{(A.41)}}{=} \sum_{i=1}^{k} \boldsymbol{u}_i \boldsymbol{u}_i^\top \boldsymbol{x} \tag{A.42}$$

$$\overset{\text{(A.29)}}{=} \sum_{i=1}^{k} \boldsymbol{u}_i \boldsymbol{u}_i^\top \sum_{j=1}^{n} \boldsymbol{u}_j c_j \tag{A.43}$$

$$= \sum_{i=1}^{k} \boldsymbol{u}_i \sum_{j=1}^{n} \boldsymbol{u}_i^\top \boldsymbol{u}_j c_j \tag{A.44}$$

$$= \sum_{i=1}^{k} \boldsymbol{u}_i c_i \tag{A.45}$$

$$\overset{\text{(A.39)}}{=} \boldsymbol{x}_\parallel, \tag{A.46}$$

where we have used that $\boldsymbol{u}_i^\top \boldsymbol{u}_j$ equals zero unless $j = i$. The mapping of $\boldsymbol{x}$ to $\boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{x} = \boldsymbol{x}_\parallel$ is called the orthogonal projection of $\boldsymbol{x}$ onto span$(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)$. It follows that $(\boldsymbol{I}_d - \boldsymbol{U}_k \boldsymbol{U}_k^\top)\boldsymbol{x}$ equals $\boldsymbol{x}_\perp$, and that the matrix $(\boldsymbol{I}_d - \boldsymbol{U}_k \boldsymbol{U}_k^\top)$ is the orthogonal projection of $\boldsymbol{x}$ onto span$(\boldsymbol{u}_1, \ldots \boldsymbol{u}_k)^\perp$.

A.7 Singular value decomposition

The singular value decomposition (SVD) of a $m \times n$ matrix $\boldsymbol{A}$ is the factorisation of the matrix into the product $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$,

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top, \tag{A.47}$$

The $m \times n$ matrix $\boldsymbol{S}$ is zero everywhere but in the first r diagonal elements $(\boldsymbol{S})_{ii}$ that are positive. We denote the $(\boldsymbol{S})_{ii}$ by s_i so that

$$\boldsymbol{S} = \left(\begin{array}{ccc|c} s_1 & & & \\ & \ddots & & \mathbf{0} \\ & & s_r & \\ \hline & \mathbf{0} & & \mathbf{0} \end{array}\right) \tag{A.48}$$

The diagonal elements are called the singular values of $\boldsymbol{A}$ and are typically ordered so that $s_1 \geq s_2 \geq \cdots \geq s_r$. Matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are both orthogonal. We denote the column vectors of the two matrices correspondingly by $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$,

$$\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m), \qquad \boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n). \tag{A.49}$$

The vectors $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ form an orthogonal basis for $\mathbb{R}^m$ and $\mathbb{R}^n$, and are called the left-singular vectors and right-singular vectors, respectively. The number $r \leq \min(m, n)$ is called the rank of the matrix $\boldsymbol{A}$.

Due to the structure of the matrix $\boldsymbol{S}$ only the $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ with $i \leq r$ actually contribute to the factorisation. Indeed, with (A.21), the $m \times n$ matrix $\boldsymbol{US}$ equals

$$\boldsymbol{US} = (s_1\boldsymbol{u}_1, \ldots, s_r\boldsymbol{u}_r, \boldsymbol{0}, \ldots, \boldsymbol{0}). \tag{A.50}$$

and with (A.24), $\boldsymbol{USV}^\top$ is

$$\boldsymbol{USV}^\top = \sum_{i=1}^{r} s_i\boldsymbol{u}_i\boldsymbol{v}_i^\top + \sum_{i=r+1}^{n} \boldsymbol{0}\boldsymbol{v}_i^\top \tag{A.51}$$

so that $\boldsymbol{A} = \boldsymbol{USV}^\top$ is

$$\boldsymbol{A} = \sum_{i=1}^{r} s_i\boldsymbol{u}_i\boldsymbol{v}_i^\top = \boldsymbol{U}_r\boldsymbol{S}_r\boldsymbol{V}_r^\top \tag{A.52}$$

where

$$\boldsymbol{U}_r = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r), \qquad \boldsymbol{S}_r = \begin{pmatrix} s_1 & & \\ & \ddots & \\ & & s_r \end{pmatrix}, \qquad \boldsymbol{V}_r = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r). \tag{A.53}$$

This is called the compact, "thin", or "skinny" SVD of $\boldsymbol{A}$.

A.8 Eigenvalue decomposition

The eigenvalue decomposition is a factorisation for symmetric matrices. The eigenvalue decomposition of the symmetric $m \times m$ matrix $\boldsymbol{A}$ of rank r is

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top, \tag{A.54}$$

where $\boldsymbol{\Lambda}$ is a $m \times m$ diagonal matrix with r non-zero elements λ_i that we can assume to be ordered as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r$. Note that the λ_i may be positive or negative. Matrix $\boldsymbol{U}$ is orthogonal with orthonormal column vectors $\boldsymbol{u}_i$. As for the SVD, the vectors $\boldsymbol{u}_i$ for which $(\boldsymbol{\Lambda})_{ii} = 0$ can actually be ignored so that

$$\boldsymbol{A} = \sum_{i=1}^{r} \lambda_i\boldsymbol{u}_i\boldsymbol{u}_i^\top = \boldsymbol{U}_r\boldsymbol{\Lambda}_r\boldsymbol{U}_r^\top, \tag{A.55}$$

where

$$\boldsymbol{U}_r = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r), \qquad \boldsymbol{\Lambda}_r = \begin{pmatrix} \lambda_i & & \\ & \ddots & \\ & & \lambda_r \end{pmatrix} \tag{A.56}$$

The vectors $\boldsymbol{u}_i$ are called the eigenvectors and the λ_i the eigenvalues. It follows from (A.55) that

$$\boldsymbol{A}\boldsymbol{u}_k = \lambda_k \boldsymbol{u}_k, \tag{A.57}$$

i.e. the matrix $\boldsymbol{A}$ only scales the vectors $\boldsymbol{u}_i$ by their corresponding eigenvalue λ_i.

A.9 Positive semi-definite and definite matrices

A symmetric $m \times m$ matrix is called positive semi-definite if all m eigenvalues are non-negative and positive definite if they are all positive. A positive definite matrix has full rank, $r = m$, and the eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$ form an orthogonal basis of $\mathbb{R}^m$.

If a matrix $\boldsymbol{M}$ has the singular value decomposition $\boldsymbol{M} = \boldsymbol{U}_r \boldsymbol{S}_r \boldsymbol{V}_r^\top$ as in (A.52), the eigenvalue decomposition of $\boldsymbol{M}\boldsymbol{M}^\top$ is

$$\boldsymbol{M}\boldsymbol{M}^\top = \boldsymbol{U}_r \boldsymbol{S}_r \underbrace{\boldsymbol{V}_r^\top \boldsymbol{V}_r}_{\boldsymbol{I}_r} \boldsymbol{S}_r \boldsymbol{U}_r^\top = \boldsymbol{U}_r \boldsymbol{S}_r^2 \boldsymbol{U}_r^\top, \tag{A.58}$$

on the other hand, the eigenvalue decomposition of $\boldsymbol{M}^\top\boldsymbol{M}$ is

$$\boldsymbol{M}^\top\boldsymbol{M} = \boldsymbol{V}_r \boldsymbol{S}_r \underbrace{\boldsymbol{U}_r^\top \boldsymbol{U}_r}_{\boldsymbol{I}_r} \boldsymbol{S}_r \boldsymbol{V}_r^\top = \boldsymbol{V}_r \boldsymbol{S}_r^2 \boldsymbol{V}_r^\top, \tag{A.59}$$

where in both cases $\boldsymbol{S}_r^2$ refers to the diagonal matrix with elements s_i^2. Both $\boldsymbol{M}^\top\boldsymbol{M}$ and $\boldsymbol{M}\boldsymbol{M}^\top$ have the s_i^2 as eigenvalues. We see that the eigenvalues are non-negative so that $\boldsymbol{M}^\top\boldsymbol{M}$ and $\boldsymbol{M}\boldsymbol{M}^\top$ are positive semi-definite matrices.

A.10 Matrix approximations

A.10.1 Low rank approximation of general matrices

The singular value decomposition allows us to decompose a $m \times n$ matrix $\boldsymbol{A}$ of rank r as

$$\boldsymbol{A} = \sum_{i=1}^{r} s_i \boldsymbol{u}_i \boldsymbol{v}_i^\top = \boldsymbol{U}_r \boldsymbol{S}_r \boldsymbol{V}_r^\top, \tag{A.60}$$

see (A.52). The r singular values $s_i > 0$ are decreasing. Intuitively, the "later" rank-one matrices $\boldsymbol{u}_i \boldsymbol{v}_i^\top$ with smaller singular values contribute less to $\boldsymbol{A}$ than the "earlier" rank-one matrices with larger singular values. In fact the best approximation $\hat{\boldsymbol{A}}$ of the matrix $\boldsymbol{A}$ by a matrix $\tilde{\boldsymbol{A}}$ of rank $k < r$ is given by the first k terms of the expansion above,

$$\hat{\boldsymbol{A}} = \sum_{i=1}^{k} s_i \boldsymbol{u}_i \boldsymbol{v}_i^\top. \tag{A.61}$$

This result is unique if and only if $s_k > s_{k+1}$. The result is obtained when the quality of the approximation is measured by the Frobenius norm

$$||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F = \sum_{ij} ((\boldsymbol{A})_{ij} - (\tilde{\boldsymbol{A}})_{ij})^2 \tag{A.62}$$

but also for other matrix norms (e.g. the spectral norm). For the Frobenius norm, the error when approximating $\boldsymbol{A}$ with $\hat{\boldsymbol{A}}$ is the sum of the squares of the remaining singular values

$$\sum_{ij}((\boldsymbol{A})_{ij} - (\hat{\boldsymbol{A}})_{ij})^2 = \sum_{i=k+1}^{r} s_i^2. \tag{A.63}$$

This result is known as the Eckart–Young–Mirsky theorem and a proof can be found in e.g. (Gentle, 2007, Section 3.10) or (Björck, 2015, Theorem 2.2.11).

A.10.2 Low rank approximation of positive semi-definite matrices

For positive semi-definite matrices, the above approximation based on the singular value decomposition carries over: The best approximation $\hat{\boldsymbol{A}}$ of a positive semi-definite matrix $\boldsymbol{A}$ of rank r by a matrix $\tilde{\boldsymbol{A}}$ of rank $k < r$ is

$$\hat{\boldsymbol{A}} = \sum_{i=1}^{k} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\top. \tag{A.64}$$

The smallest approximation error for the Frobenius norm is

$$||\boldsymbol{A} - \hat{\boldsymbol{A}}||_F = \sum_{ij=1}^{m}((\boldsymbol{A})_{ij} - (\hat{\boldsymbol{A}})_{ij})^2 = \sum_{i=k+1}^{r} \lambda_i^2, \tag{A.65}$$

so that $||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F \geq \sum_{i=k+1}^{r} \lambda_i^2$ for other candidates $\tilde{\boldsymbol{A}}$.

A.10.3 Approximating symmetric matrices by positive semi-definite matrices

A rank r symmetric matrix $\boldsymbol{A}$ that is not positive definite has the eigenvalue decomposition

$$\boldsymbol{A} = \sum_{i=1}^{r} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\top, \tag{A.66}$$

where some λ_i are negative. Let us assume that there are $p \geq 1$ positive eigenvalues and that $\lambda_1 \geq \ldots \geq \lambda_p > 0 > \lambda_{p+1} \geq \ldots \geq \lambda_r$. We would like to determine the positive semi-definite matrix closest to $\boldsymbol{A}$. Measuring closeness by the Frobenius norm, a result by Higham (1988) shows that the closest matrix $\hat{\boldsymbol{A}}$ is obtained by retaining the terms with positive eigenvalues only,

$$\hat{\boldsymbol{A}} = \sum_{i=1}^{p} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\top = \sum_{i=1}^{r} \max(\lambda_i, 0) \boldsymbol{u}_i \boldsymbol{u}_i^\top. \tag{A.67}$$

The approximation error is

$$||\boldsymbol{A} - \hat{\boldsymbol{A}}||_F = \sum_{i=p+1}^{r} \lambda_i^2, \tag{A.68}$$

and matrix $\hat{\boldsymbol{A}}$ has rank p.

Following (Higham, 1988), the proof exploits that the Frobenius norm is invariant under rotations, i.e. $||\boldsymbol{A}||_F = ||\boldsymbol{AU}||_F = ||\boldsymbol{UA}||_F$ for any orthogonal matrix $\boldsymbol{U}$. Let $\tilde{\boldsymbol{A}}$ be a positive semi-definite matrix. We then have

$$
\begin{aligned}
||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F &= ||\boldsymbol{U}_r \boldsymbol{\Lambda}_r \boldsymbol{U}_r^\top - \tilde{\boldsymbol{A}}||_F && \text{(A.69)}\\
&= ||\boldsymbol{U}_r^\top \boldsymbol{U}_r \boldsymbol{\Lambda}_r \boldsymbol{U}_r^\top \boldsymbol{U}_r - \boldsymbol{U}_r^\top \tilde{\boldsymbol{A}} \boldsymbol{U}_r||_F && \text{(A.70)}\\
&= ||\boldsymbol{\Lambda}_r - \underbrace{\boldsymbol{U}_r^\top \tilde{\boldsymbol{A}} \boldsymbol{U}_r}_{\boldsymbol{B}}||_F && \text{(A.71)}\\
&= \sum_{i=1}^{r} (\lambda_i - b_{ii})^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^{r} b_{ij}^2 && \text{(A.72)}
\end{aligned}
$$

where b_{ij} are the elements of the matrix $\boldsymbol{B} = \boldsymbol{U}_r^\top \tilde{\boldsymbol{A}} \boldsymbol{U}_r$. Because the $b_{ij}^2 \geq 0$, we have

$$
\begin{aligned}
||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F &\geq \sum_{i=1}^{r} (\lambda_i - b_{ii})^2 && \text{(A.73)}\\
&= \sum_{i=1}^{p} \underbrace{(\lambda_i - b_{ii})^2}_{\geq 0} + \sum_{i=p+1}^{r} (\lambda_i - b_{ii})^2 && \text{(A.74)}\\
&\geq \sum_{i=p+1}^{r} (\lambda_i - b_{ii})^2 && \text{(A.75)}
\end{aligned}
$$

Since $b_{ii} \geq 0$ as $\tilde{\boldsymbol{A}}$ is restricted to be positive semi-definite and $\lambda_i < 0$ for $i > p$, we have in the equation above that $\lambda_i - b_{ii} \leq \lambda_i < 0$ and thus $(\lambda_i - b_{ii})^2 \geq \lambda_i^2$. We thus obtain the following lower bound for $||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F$:

$$
||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F \geq \sum_{i=p+1}^{r} \lambda_i^2 \tag{A.76}
$$

A diagonal matrix $\boldsymbol{B}$ with elements $b_i = \max(\lambda_i, 0)$ achieves the lower bound. The result in (A.67) now follows from $\tilde{\boldsymbol{A}} = \boldsymbol{U}_r \boldsymbol{B} \boldsymbol{U}_r^\top$.

A.10.4 Low rank approximation of symmetric matrices by positive semi-definite matrices

As before let the symmetric matrix $\boldsymbol{A}$ of rank r have p positive eigenvalues,

$$
\boldsymbol{A} = \sum_{i=1}^{r} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\top, \tag{A.77}
$$

where $\lambda_1 \geq \ldots \geq \lambda_p > 0 > \lambda_{p+1} \geq \ldots \geq \lambda_r$. Combining (A.67) with (A.64) we show here that the best positive semi-definite approximation of rank $k < p$ is

$$
\hat{\boldsymbol{A}} = \sum_{i=1}^{k} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\top, \tag{A.78}
$$

and that the smallest approximation error is

$$||\boldsymbol{A} - \hat{\boldsymbol{A}}||_F = \sum_{i=k+1}^{r} \lambda_i^2. \tag{A.79}$$

Let $\tilde{\boldsymbol{A}}$ be a positive semi-definite matrix of rank $k < p$. As for the proof of (A.64), we write

$$\begin{aligned} ||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F &= ||\boldsymbol{U}_r \boldsymbol{\Lambda}_r \boldsymbol{U}_r^\top - \tilde{\boldsymbol{A}}||_F && \text{(A.80)} \\ &= ||\boldsymbol{U}_r^\top \boldsymbol{U}_r \boldsymbol{\Lambda}_r \boldsymbol{U}_r^\top \boldsymbol{U}_r - \boldsymbol{U}_r^\top \tilde{\boldsymbol{A}} \boldsymbol{U}_r||_F && \text{(A.81)} \\ &= ||\boldsymbol{\Lambda}_r - \underbrace{\boldsymbol{U}_r^\top \tilde{\boldsymbol{A}} \boldsymbol{U}_r}_{\boldsymbol{B}}||_F && \text{(A.82)} \\ &= \sum_{i=1}^{r} (\lambda_i - b_{ii})^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^{r} b_{ij}^2 && \text{(A.83)} \end{aligned}$$

where $b_{ij} = \boldsymbol{u}_i^\top \tilde{\boldsymbol{A}} \boldsymbol{u}_j$ are the elements of the matrix $\boldsymbol{B} = \boldsymbol{U}_r^\top \tilde{\boldsymbol{A}} \boldsymbol{U}_r$. Because the $b_{ij}^2 \geq 0$, we have

$$\sum_{\substack{i,j=1 \\ i \neq j}}^{r} b_{ij}^2 \geq \sum_{\substack{i,j=1 \\ i \neq j}}^{p} b_{ij}^2 \tag{A.84}$$

and hence

$$\begin{aligned} ||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F &\geq \sum_{i=1}^{r} (\lambda_i - b_{ii})^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^{p} b_{ij}^2 && \text{(A.85)} \\ &= \sum_{i=1}^{p} (\lambda_i - b_{ii})^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^{p} b_{ij}^2 + \sum_{i=p+1}^{r} (\lambda_i - b_{ii})^2 && \text{(A.86)} \\ &= ||\boldsymbol{\Lambda}_p - \boldsymbol{U}_p^\top \tilde{\boldsymbol{A}} \boldsymbol{U}_p||_F + \sum_{i=p+1}^{r} (\lambda_i - b_{ii})^2 && \text{(A.87)} \end{aligned}$$

As $\tilde{\boldsymbol{A}}$ is restricted to be positive semi-definite $b_{ii} \geq 0$, and since $\lambda_i < 0$ for $i > p$, we have in the equation above that $\lambda_i - b_{ii} \leq \lambda_i < 0$ and thus $(\lambda_i - b_{ii})^2 \geq \lambda_i^2$. Hence:

$$||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F \geq ||\boldsymbol{\Lambda}_p - \boldsymbol{U}_p^\top \tilde{\boldsymbol{A}} \boldsymbol{U}_p||_F + \sum_{i=p+1}^{r} \lambda_i^2 \tag{A.88}$$

The matrix $\boldsymbol{\Lambda}_p$ is a positive definite $p \times p$ matrix, while the matrix $\boldsymbol{U}_p^\top \tilde{\boldsymbol{A}} \boldsymbol{U}_p$ is a $p \times p$ matrix of rank k. The smallest approximation error of a positive definite matrix by a matrix of lower rank is with (A.65) equal to $\sum_{i=k+1}^{p} \lambda_i^2$. We can thus bound $||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F$ from below by $\sum_{i=k+1}^{r} \lambda_i^2$,

$$||\boldsymbol{A} - \tilde{\boldsymbol{A}}||_F \geq \sum_{i=k+1}^{r} \lambda_i^2. \tag{A.89}$$

The matrix $\hat{\boldsymbol{A}}$ in (A.78) achieves the lower bound which completes the proof.

References

[1] Å Björck. *Numerical Methods in Matrix Computations*. Springer, 2015.

[2] J.E. Gentle. *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer, 2007.

[3] N.J. Higham. "Computing a nearest symmetric positive semidefinite matrix". In: *Linear Algebra and its Applications* 103 (1988), pp. 103–118.