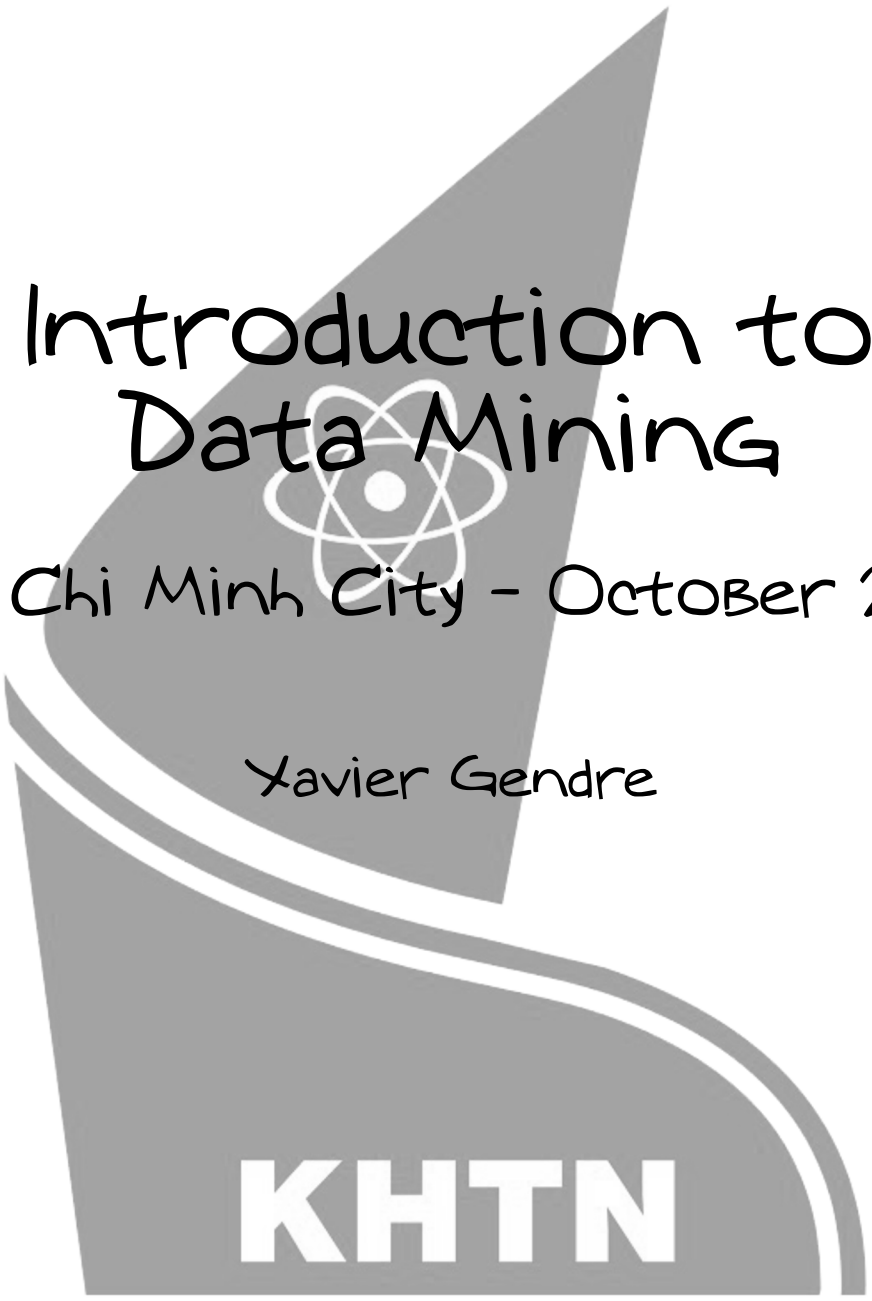
UNIVERSITÉ TOULOUSE III PAUL SABATIER

Université de Toulouse

Introduction to Data Mining

Ho Chi Minh City - October 2016

Xavier Gendre



Introduction to Data Mining

Ho Chi Minh City - October 2016

Xavier Gendre

Contents

Foreword

These lecture notes are related to the lecture 'Introduction to Data Mining' given at the Ho Chi Minh City University of Science during the period from October 24, 2016 to October 28, 2016.

The lectures were given by Gendre Xavier and the practical sessions were jointly given by Gendre Xavier and Risser Laurent from the Institut de Mathématiques de Toulouse.

For any request or comment, please contact the author at xavier.gendre@math.univ-toulouse.fr.

1 — Principal component analysis

1.1 Introduction

From a general point of view, the aim of statistics is to describe phenomena based on observations of variables related to these phenomena. Hereafter, we call **variable** anything that we can observe. If we consider some variable x, we call **data** any set of observations of x. In the sequel of this document, a data set of n observations of a variable x is usually denoted by $x_1,\dots,x_n$. We say that a variable x is **real** when its observations belong to $\mathbb{R}$. Unless stated otherwise, all the variables considered in this document are assumed to be real.

Let us consider a real variable x, we usually associate **weights** to a data set $x_1,\dots,x_n \in \mathbb{R}$, namely positive real numbers $w_1,\dots,w_n > 0$ such that $w_1+\cdots+w_n = 1$. For $i \in \{1,\dots,n\}$, the weight w_i represents the importance given to the observation x_i among the data. In the particular case of indentical importance, we refer to **normalized weights** for $w_1 = \cdots = w_n = 1/n$.

Definition 1.1. Considering a data set $x_1,\dots,x_n \in \mathbb{R}$ associated to a real variable x and some weights $w_1,\dots,w_n$, the **mean** $\bar{x}$ of x is given by

$$\bar{x} = \sum_{i=1}^{n} w_i x_i$$

and the **variance** $\sigma^2(x)$ of x by

$$\sigma^2(x) = \sum_{i=1}^{n} w_i \,(x_i - \bar{x})^2 .$$

The variance is a nonnegative quantity and measures the dispersal of the observations around the mean. Such dispersal measurements play a central role in this first chapter.

In order to get a more precise information about how the data are distributed, we can also consider the **quantiles** associated to a data set:

Definition 1.2. Let $p \in [0,1]$, the **p-quantile** $q_p(x) \in \mathbb{R}$ of a data set $x_1,\dots,x_n \in \mathbb{R}$ is

$$q_p(x) = \inf\left\{ t \in \mathbb{R} \text{ such that } \sum_{i=1}^{n} w_i \mathbf{1}_{\{x_i \leqslant t\}} \geqslant p \right\}.$$

If $p = 0.5$, the p-quantile is called the **median** and if $p = 0.25$ or $p = 0.75$, the p-quantiles are called the first and third **quartile**, respectively. The difference between the third and the first quartiles is called the **interquartile range** $\text{IQR} = q_{0.75}(x) - q_{0.25}(x)$.

The quantiles supply a convenient way, known as a **box plot**, for graphic representation of the distribution of a data set. Basically, five informations are summarized in a box plot: the smallest observation, the first quartile, the median, the third quartile and the largest observation. Moreover, one adds two whiskers to the box to indicate some additional informations. The lengths of these whiskers can vary from a representation to an other but an usual choice is to use the lowest observation still within $1.5 \times IQR$ of the first quartile and the highest observation still within $1.5 \times IQR$ of the third quartile (see Figure 1.1).

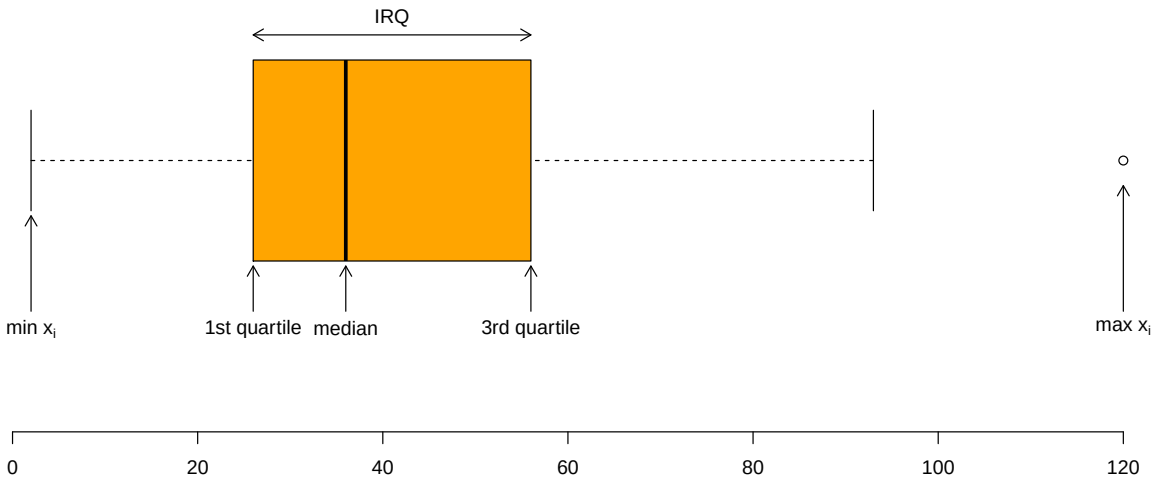


Figure 1.1: An example of box plot.

As mentioned previously, we focus on dispersal measurement in this chapter. A **principal component analysis** (PCA) mainly amounts to compute the directions in which the amount of dispersion is the largest. Let us consider a naive example to introduce the concepts beyond such a procedure. We consider two real variables x^1 and x^2 observed n times. Thus, we have at our disposal a data set of points $x_1 = (x_1^1, x_1^2), \ldots, x_n = (x_n^1, x_n^2) \in \mathbb{R}^2$ and weights $w_1, \ldots, w_n > 0$. The variances $\sigma^2(x^1)$ and $\sigma^2(x^2)$ measure the dispersal of each coordinate around their means $\overline{x^1}$ and $\overline{x^2}$, respectively (we assume here that these variances are positive). In such a simple framework, looking for the direction in which the amount of dispersion is maximal consists in finding a unit vector $v = (v_1, v_2) \in \mathbb{R}^2$ (*i.e.* $\|v\| = 1$) such that the variance of the variable created as the linear combination $v_1 x^1 + v_2 x^2$ is maximal. In other words, we want to maximize the following quantity,

$$\begin{aligned} \sigma^2\left(v_1 x^1 + v_2 x^2\right) &= \sum_{i=1}^{n} w_i \left(v_1\left(x_i^1 - \overline{x^1}\right) + v_2\left(x_i^2 - \overline{x^2}\right)\right)^2 \\ &= v_1^2 \sigma^2(x^1) + 2 v_1 v_2 \sigma(x^1, x^2) + v_2^2 \sigma^2(x^2) \end{aligned} \tag{1.1}$$

where the **covariance** between the observations of x^1 and x^2 has been denoted by

$$\sigma(x^1,x^2) = \sum_{i=1}^{n} w_i \left(x_i^1 - \overline{x^1}\right)\left(x_i^2 - \overline{x^2}\right).$$

Using matrix notations, we see that maximizing (1.1) boils down to maximize $v^\intercal \Sigma v$ with respect to the unit vector $v \in \mathbb{R}^2$ where Σ is the **covariance matrix**,

$$\Sigma = \begin{pmatrix} \sigma^2(x^1) & \sigma(x^1,x^2) \\ \sigma(x^1,x^2) & \sigma^2(x^2). \end{pmatrix}$$

Because Σ is symmetric, we know that the maximum of $v^\intercal \Sigma v$ is the largest eigenvalue λ_1 of Σ and is reached by any associated normalized eigenvector $v = v^1$. Taking any unit vector $v^2 \in \mathbb{R}^2$ which is orthogonal to v^1, you obtain an orthonormal basis $\{v^1, v^2\}$ of $\mathbb{R}^2$. This basis is adapted to our data set in the sense that its axes are aligned with the "principal directions" of the dispersion (see Figure 1.2).

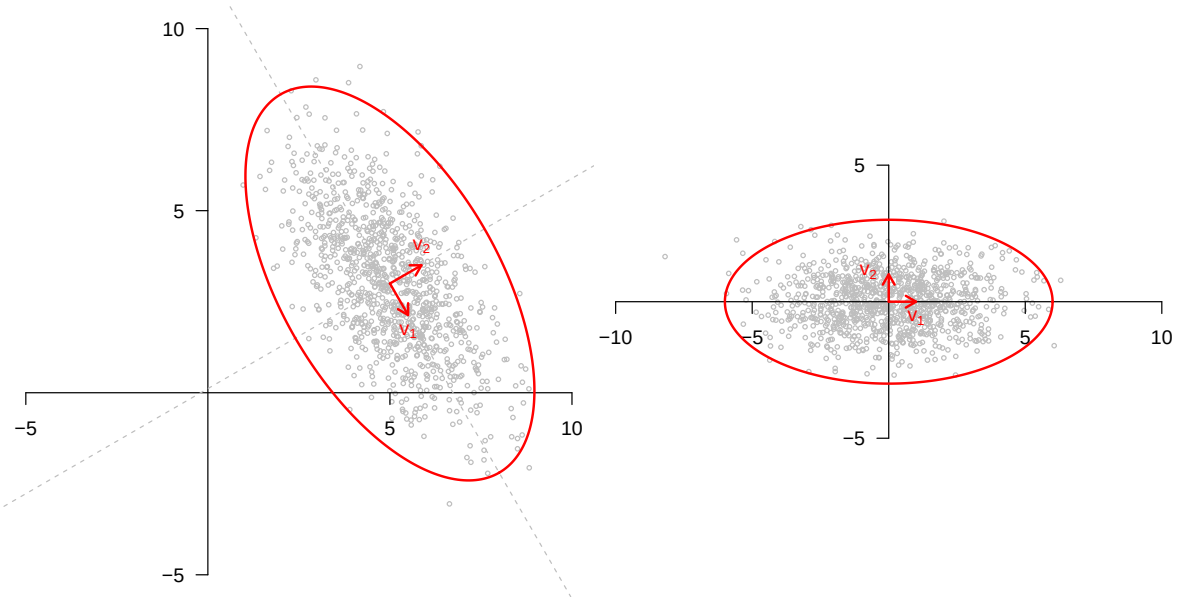


Figure 1.2: Principal directions of the dispersion of a data set in $\mathbb{R}^2$ (left) and its representation in principal axes (right).

1.2 Basic PCA

1.2.1 Principle

A PCA procedure is not restricted to two real variables and we now consider the common case of p real variables $x^1,\dots,x^p$ observed n times with weights $w_1,\dots,w_n > 0$. Our data set is then given by the vectors, for $i \in \{1,\dots,n\}$,

$$x_i = \begin{pmatrix} x_i^1 \\ \vdots \\ x_i^p \end{pmatrix} \in \mathbb{R}^p.$$

The mean $\bar{x}$ is defined as in the real case by the weighted sum of the observations

$$\bar{x} = \sum_{i=1}^{n} w_i x_i = \begin{pmatrix} \overline{x^1} \\ \vdots \\ \overline{x^p} \end{pmatrix} \in \mathbb{R}^p.$$

Handling vectorial data quickly induces some difficulties for the notations and we introduce some convenient matrix. First, the **weight matrix** W is the diagonal $n \times n$ matrix given by

$$W = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & w_n \end{pmatrix}.$$

The **data matrix** X is the $n \times p$ matrix containing the observations,

$$X = \begin{pmatrix} x_1^1 & x_1^2 & \dots & x_1^p \\ x_2^1 & x_2^2 & \dots & x_2^p \\ \vdots & \vdots & \vdots & \vdots \\ x_n^1 & x_n^2 & \dots & x_n^p \end{pmatrix},$$

and we denote by $\bar{X}$ its centered version,

$$\bar{X} = \begin{pmatrix} x_1^1 - \overline{x^1} & x_1^2 - \overline{x^2} & \dots & x_1^p - \overline{x^p} \\ x_2^1 - \overline{x^1} & x_2^2 - \overline{x^2} & \dots & x_2^p - \overline{x^p} \\ \vdots & \vdots & \vdots & \vdots \\ x_n^1 - \overline{x^1} & x_n^2 - \overline{x^2} & \dots & x_n^p - \overline{x^p} \end{pmatrix}.$$

Note that the matrices X and $\bar{X}$ can be read row by row to browse the observed individuals $x_1, \dots, x_n \in \mathbb{R}^p$ or column by column to browse the variables $x^1, \dots, x^p \in \mathbb{R}^n$. Finally, the pairwise covariance computations lead to the $p \times p$ **covariance matrix** Σ,

$$\Sigma = \begin{pmatrix} \sigma^2(x^1) & \sigma(x^1, x^2) & \dots & \sigma(x^1, x^p) \\ \sigma(x^1, x^2) & \sigma^2(x^2) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \sigma(x^{p-1}, x^p) \\ \sigma(x^1, x^p) & \dots & \sigma(x^{p-1}, x^p) & \sigma^2(x^p) \end{pmatrix}.$$

Proposition 1.1. *The covariance matrix is given by*

$$\Sigma = \bar{X}^\intercal W \bar{X}.$$

Moreover, the matrix Σ is symmetric and nonnegative-definite.

Proof. Let $j, k \in \{1 \dots, p\}$, we compute

$$(\bar{X}^\intercal W \bar{X})_{jk} = \sum_{i=1}^{n} w_i \bar{X}_{ij} \bar{X}_{ik} = \sum_{i=1}^{n} w_i \left(x_i^j - \overline{x^j} \right) \left(x_i^k - \overline{x^k} \right) = \sigma(x^j, x^k) = \Sigma_{jk}.$$

By symmetry of covariance operator, Σ is symmetric and the nonnegative-definite property comes from, for any $u = (u_1, \dots, u_p)^\intercal \in \mathbb{R}^p$,

$$u^\intercal \Sigma u = (\bar{X}u)^\intercal W (\bar{X}u) = \sum_{i=1}^{n} w_i (\bar{X}u)_i^2 \geqslant 0.$$

□

A simple measurement of the dispersion of the data in $\mathbb{R}^p$ is to sum the variances of the coordinates. This approach gives us the **standard inertia**,

$$I(x) = \sum_{j=1}^{p} \sigma^2(x^j).$$

As we claimed in the previous section, we are interested in the directions that capture the maximal amount of dispersion. In our framework, the standard inertia plays the role of dispersal measurement. Let $d \leqslant p$ be some positive integer, we want to find d orthonormal vectors $v^1, \dots, v^d \in \mathbb{R}^p$ such that the standard inertia of the orthogonal projection of the initial data set onto the linear space E_d generated by $\{v^1, \dots, v^d\}$ is maximal. We introduce the usual notation for the standard scalar product in $\mathbb{R}^p$,

$$\forall u = (u_1, \dots, u_p)^\intercal, v = (v_1, \dots, v_p)^\intercal \in \mathbb{R}^p,\ \langle u, v \rangle = u^\intercal v = \sum_{j=1}^{p} u_j v_j$$

and its associated norm $\|\cdot\|$ given by

$$\forall u = (u_1, \dots, u_p)^\intercal \in \mathbb{R}^p,\ \|u\|^2 = \langle u, u \rangle = \sum_{j=1}^{p} u_j^2.$$

Note that, with these notations, we can write the standard inertia as a weighted mean of squared distance just like the variance in the real case,

$$I(x) = \sum_{i=1}^{n} w_i \sum_{j=1}^{p} \left(x_i^j - \overline{x^j}\right)^2 = \sum_{i=1}^{n} w_i \|x_i - \bar{x}\|^2. \tag{1.2}$$

Considering d orthonormal vectors $v^1, \dots, v^d \in \mathbb{R}^p$, we denote by π_d the orthogonal projection onto E_d and we have, for any $i \in \{1, \dots, n\}$,

$$\pi_d x_i = \sum_{k=1}^{d} \langle x_i, v^k \rangle v^k.$$

Let us compute the standard inertia of these projected data with the formula (1.2),

$$I(\pi_d x) = \sum_{i=1}^{n} w_i \|\pi_d x_i - \overline{\pi_d x}\|^2 = \sum_{i=1}^{n} w_i \|\pi_d (x_i - \bar{x})\|^2 = \sum_{i=1}^{n} w_i \sum_{k=1}^{d} \langle x_i - \bar{x}, v^k \rangle^2.$$

For any $i \in \{1, \dots, n\}$ and $k \in \{1, \dots, d\}$, we have

$$\langle x_i - \bar{x}, v^k \rangle = \sum_{j=1}^{p} \left(x_i^j - \overline{x^j}\right) v_j^k = \sum_{j=1}^{p} \bar{X}_{ij} v_j^k = \left(\bar{X} v^k\right)_i.$$

Then, with Proposition 1.1, we obtain the inertia

$$\begin{aligned} I(\pi_d x) &= \sum_{i=1}^{n} w_i \sum_{k=1}^{d} \left(\bar{X} v^k\right)_i^2 \\ &= \sum_{k=1}^{d} \left(\bar{X} v^k\right)^\intercal W \left(\bar{X} v^k\right) \\ &= \sum_{k=1}^{d} v^{k\intercal} \Sigma v^k \\ &= \sum_{k=1}^{d} \langle \Sigma v^k, v^k \rangle. \end{aligned} \tag{1.3}$$

In other words, to get the d directions which contain the maximal amount of inertia, we need to compute the orthonormal vectors $v^1, \dots, v^d \in \mathbb{R}^p$ that make the last sum in (1.3) as large as possible.

Because Σ is a symmetric nonnegative-definite matrix, we know that there exists an orthogonal matrix V (*i.e.* $V^\intercal = V^{-1}$) such that

$$V^\intercal \Sigma V = \Lambda = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \lambda_p \end{pmatrix}$$

with $\lambda_1 \geqslant \lambda_2 \geqslant \dots \geqslant \lambda_p \geqslant 0$. For any $j \in \{1, \dots, p\}$, λ_j is the eigenvalue of Σ associated to the eigenvector v^j given by the j-th column of V. Moreover, $\{v^1, \dots, v^p\}$ forms an orthonormal basis of $\mathbb{R}^p$. Because the λ_j's are ordered, the space E_d is generated by the d first eigenvectors $v^1, \dots, v^d$ which make the sum (1.3) maximal. We also have that the inertia of projected data onto E_d is

$$I(\pi_d x) = \sum_{k=1}^{d} \lambda_k.$$

Since $I(x) = \lambda_1 + \dots + \lambda_p$, we call **exlained inertia** of E_d the ratio

$$\frac{\sum_{k=1}^{d} \lambda_k}{\sum_{k=1}^{p} \lambda_k}.$$

This quantity measures the global quality of the representation of the data set in E_d in the sense that it quantifies the amount of dispersion captured by the d first principal directions.

1.2.2 Principal components

The eigenvectors $v^1, \dots, v^p$ provide an orthonormal basis adapted to represent the initial data set. For each individual, we can compute its coordinates in this basis. Let $i \in \{1, \dots, n\}$ and

$j \in \{1,\dots,p\}$, we denote by c_i^j the coordinate of the i-th centered individual $x_i - \bar{x}$ along the vector v^j,

$$c_i^j = \langle x_i - \bar{x}, v^j \rangle.$$

This leads us to consider p new centered vectors of observations $c^1,\dots,c^p \in \mathbb{R}^n$ given by, for any $j \in \{1,\dots,p\}$,

$$c^j = \begin{pmatrix} c_1^j \\ \vdots \\ c_n^j \end{pmatrix} = \bar{X} v^j.$$

These vectors $c^1,\dots,c^p$ are called **principal components**. Treating them as p variables gives a $n \times p$ data matrix C, namely the **principal components matrix**,

$$C = \bar{X} V.$$

Proposition 1.2. *The principal components are centered and noncorrelated. Moreover, for any $j \in \{1,\dots,p\}$, we have*

$$\sigma^2(c^j) = \lambda_j.$$

Proof. As linear combinations of centered variables given by the columns of $\bar{X}$, the principal components are obviously centered. Let us compute the covariance matrix associated to C,

$$C^\intercal W C = V^\intercal \bar{X}^\intercal W \bar{X} V = V^\intercal \Sigma V = \Lambda.$$

So, for any $j,k \in \{1,\dots,p\}$ with $j \neq k$, we have

$$\sigma^2(c^j) = \lambda_j \quad \text{and} \quad \sigma(c^j, c^k) = 0.$$

□

The columns of C contain the coordinates of the n individuals along the principal directions. Thus, considering the d first columns, we can represent the individuals in the space E_d. In particular, when $d = 2$, the space E_2 is called the **principal plane**.

A question that naturally arises is about the quality of the representation of the individuals. Although the explained inertia provides a global measurement, we are interested in measuring the quality for each individual. Let $i \in \{1,\dots,n\}$, this can be done for the i-th individual through the angle θ_i between the initial point $x_i - \bar{x} \in \mathbb{R}^p$ and its representation in E_d. Indeed, by orthonormality of the basis $\{v^1,\dots,v^p\}$, we have

$$\cos^2 \theta_i = \frac{\|\pi_d(x_i - \bar{x})\|^2}{\|x_i - \bar{x}\|^2} = \frac{(c_i^1)^2 + \cdots + (c_i^d)^2}{(c_i^1)^2 + \cdots + (c_i^p)^2}.$$

The closer this quantity is to 1, the better is the representation of the i-th individual in E_d.

If we have at our disposal a new individual $y \in \mathbb{R}^p$ which has not been taken into account in the PCA, it is possible to represent it in E_d without running the procedure again. Indeed, if we center y with $\bar{x}$, we can compute the coordinate of y in along v^j with

$$\langle y - \bar{x}, v^j \rangle.$$

1.2.3 Correlation circle

Now that we know how to represent the data set in E_d, we want to be able to interpret the coordinates of each individual with respect to the initial variables. In other terms, we want to discuss about the position of an individual in the space E_d.

The relation between the initial variables $x^1,\dots,x^p \in \mathbb{R}^n$ and the principal components $c^1,\dots,c^p \in \mathbb{R}^n$ is linear. Thus, for any $j,k \in \{1,\dots,p\}$, it seems reasonable to consider the **Pearson correlation coefficient** $\rho(x^j,c^k)$ between x^j and c^k, namely

$$\rho(x^j,c^k) = \frac{\sigma(x^j,c^k)}{\sqrt{\sigma^2(x^j)\sigma^2(c^k)}}.$$

We know that $\bar{X} = CV^\intercal$, thus, for any $i \in \{1,\dots,n\}$,

$$x_i^j - \overline{x^j} = \sum_{\ell=1}^{p} v_j^\ell c_i^\ell \tag{1.4}$$

and Proposition 1.2 leads to

$$\rho(x^j,c^k) = \frac{v_j^k \lambda_k}{\sqrt{\sigma^2(x^j)\lambda_k}} = \frac{v_j^k\sqrt{\lambda_k}}{\sqrt{\sigma^2(x^j)}}.$$

An interesting case is the principal plane E_2. Indeed, for each initial variable x^j, we can consider the vector $\vec{\tau}_j$ given by the correlation coefficients with c^1 and c^2,

$$\vec{\tau}_j = \big(\rho(x^j,c^1),\rho(x^j,c^2)\big) = \left(\frac{v_j^1\sqrt{\lambda_1}}{\sqrt{\sigma^2(x^j)}}, \frac{v_j^2\sqrt{\lambda_2}}{\sqrt{\sigma^2(x^j)}}\right).$$

With the help of (1.4) and Proposition 1.2, we can expand $\sigma^2(x^j)$ to get

$$\sigma^2(x^j) = \sum_{\ell=1}^{p} \left(v_j^\ell\right)^2 \lambda_\ell \geqslant \left(v_j^1\sqrt{\lambda_1}\right)^2 + \left(v_j^2\sqrt{\lambda_2}\right)^2.$$

Thus, the vectors $\vec{\tau}_j$ are all in the unit circle. The representation of these vectors is known as the **correlation circle**. For $j \in \{1,\dots,p\}$, the closer $\vec{\tau}_j$ is to the circle, the better x_j is represented in E_2. Moreover, the proximity of $\vec{\tau}_j$ to an axis reflects the correlation importance and its direction represents the sign of this correlation (see Practicals below for examples).

A common representation in the principal plane consists in plotting simultaneously the individuals and the correlation circle. Such a representation is known as a **biplot** and permits an easier reading of the results.

1.3 Normalizing the variances

1.3.1 Scale-dependency

An important point about variances has been ignored in the previous section. Indeed, the PCA procedure is based on the dispersal measurement provided by the variance operator but this

operator is scale-dependent, namely

$$\sigma^2(\alpha x^1) = \alpha^2 \sigma^2(x^1),\ \alpha \in \mathbb{R}.$$

As a consequence, if we change the unit of the variable x^1, then we deeply change its role in the PCA because its importance in the sense of the variance would be diminished if α is smaller than 1 or arbitrariy increased if α tends towards infinity. However, changing the unit does not give more or less information and our procedure should not suffer from scale-dependency.

To avoid such a drawback, a solution consists in normalizing the variables in order to set all the variances to 1. Let $j \in \{1,\dots,p\}$, we introduce the **reduced centred version** $\tilde{x}^j$ of x^j where, for any $i \in \{1,\dots,n\}$,

$$\tilde{x}_i^j = \frac{x_i^j - \overline{x^j}}{\sqrt{\sigma^2(x^j)}}.$$

Indeed, if we modify the unit of x^j, it does not change anything for $\tilde{x}^j$ which remains such that $\overline{\tilde{x}^j} = 0$ and $\sigma^2(\tilde{x}^j) = 1$. Moreover, this is straightforward to see that $\sigma(\tilde{x}^j, \tilde{x}^k) = \rho(x^j, x^k)$ and $I(\tilde{x}) = p$.

1.3.2 Revisiting PCA

The data matrix associated to $\tilde{x}^1,\dots,\tilde{x}^p$ is denoted by $\tilde{X}$ and its covariance matrix is known as the **correlation matrix**,

$$\tilde{X}^\top W \tilde{X} = \begin{pmatrix} 1 & \rho(x^1,x^2) & \dots & \rho(x^1,x^p) \\ \rho(x^1,x^2) & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \rho(x^{p-1},x^p) \\ \rho(x^1,x^p) & \dots & \rho(x^{p-1},x^p) & 1 \end{pmatrix}.$$

From there, we could apply the same procedure as the one described in the previous section. Such a PCA works well and will be experimented in the Practicals. However, we would like to take this opportunity to introduce a more general outlook on PCA procedure.

The matrix relation between the data matrices $\bar{X}$ and $\tilde{X}$ is given by

$$\tilde{X} = \bar{X} M^{1/2}$$

where we have set the $p \times p$ diagonal matrix

$$M^{1/2} = \begin{pmatrix} \frac{1}{\sqrt{\sigma^2(x^1)}} & 0 & \dots & 0 \\ 0 & \frac{1}{\sqrt{\sigma^2(x^2)}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{\sqrt{\sigma^2(x^p)}} \end{pmatrix}.$$

The diagonal elements of $M^{1/2}$ can be seen as weights for the variables similarly as the w_i on the diagonal of W are weights for the individuals. The diagonal elements of $M^{1/2}$ do not sum to 1 but when a variable has a large variance, it receives a weight inversely related, so that its importance is reduced with respect to other variables with smaller variances. Thus, measuring the distance between observed individuals for the reduced centred versions of the variables amounts to compute a sum weighted by the diagonal elements of $M = M^{1/2} \times M^{1/2}$. Namely, for any $i, j \in \{1, \dots, n\}$,

$$\begin{aligned}
\|\tilde{x}_i - \tilde{x}_j\|^2 &= \sum_{k=1}^{p} \left(\tilde{x}_i^k - \tilde{x}_j^k\right)^2 \\
&= \sum_{k=1}^{p} \left(\frac{x_i^k - \overline{x^k}}{\sqrt{\sigma^2(x^k)}} - \frac{x_j^k - \overline{x^k}}{\sqrt{\sigma^2(x^k)}} \right)^2 \\
&= \sum_{k=1}^{p} \frac{(x_i^k - x_j^k)^2}{\sigma^2(x^k)} \\
&= \sum_{k=1}^{p} M_{kk}(x_i^k - x_j^k)^2 \\
&= (x_i - x_j)^\intercal M (x_i - x_j).
\end{aligned}$$

The last line of the previous computation induces the definitions of the bilinear form $\langle \cdot, \cdot \rangle_M$, for any $u = (u_1, \dots, u_p)^\intercal, v = (v_1, \dots, v_p)^\intercal \in \mathbb{R}^p$,

$$\langle u, v \rangle_M = u^\intercal M v$$

and its derived quadratic form $\| \cdot \|_M^2$,

$$\|u\|_M^2 = \langle u, u \rangle_M = u^\intercal M u.$$

Assuming that the variances $\sigma^2(x^1), \dots, \sigma^2(x^p)$ are positive, the form $\langle \cdot, \cdot \rangle_M$ is a proper scalar product in $\mathbb{R}^p$ and $\| \cdot \|_M$ is its associated norm. The dispersal measurement given by the standard inertia of $\tilde{x}^1, \dots, \tilde{x}^p$ becomes the ***M*-inertia** $I_M(x)$ of the initial variables $x^1, \dots, x^p$,

$$I(\tilde{x}) = \sum_{i=1}^{n} w_i \|\tilde{x}_i\|^2 = \sum_{i=1}^{n} w_i \|x_i - \bar{x}\|_M^2 = I_M(x).$$

Instead of computing directions that preserve $I(\tilde{x})$ as linear combinations of $\tilde{x}^1, \dots, \tilde{x}^p$, we can now consider directions that preserve $I_M(x)$ as linear combinations of $x^1, \dots, x^p$. Seeing the PCA procedure on reduced centred versions of the variables as a PCA procedure defined with an unusual scalar product is a powerful point of view which gives a lot of flexibility for the applications.

1.4 General outlook

Motivated by the previous section, we are now able to define a quite general PCA procedure. Considering p real variables $x^1, \dots, x^p$ observed n times, we have at our disposal the $n \times p$ data matrix X. This matrix induces a linear map between the **space of individuals** which is isomorphic to $\mathbb{R}^p$ and the **space of variables** which is isomorphic to $\mathbb{R}^n$. Given weights $w_1 \dots, w_n > 0$

and the $n \times n$ symmetric positive definite matrix $W = \mathrm{diag}(w_1, \ldots, w_n)$, we endow the space of variables with a scalar product $\langle \cdot, \cdot \rangle_W$ defined by, for any $u = (u_1, \ldots, u_n)^\mathsf{T}, v = (v_1, \ldots, v_n)^\mathsf{T} \in \mathbb{R}^n$,

$$\langle u, v \rangle_W = u^\mathsf{T} W v = \sum_{i=1}^{n} w_i u_i v_i.$$

Similarly, given a $p \times p$ symmetric positive definite matrix M, we endow the space of individuals with a scalar product $\langle \cdot, \cdot \rangle_M$ defined by, for any $u = (u_1, \ldots, u_p)^\mathsf{T}, v = (v_1, \ldots, v_p)^\mathsf{T} \in \mathbb{R}^p$,

$$\langle u, v \rangle_W = u^\mathsf{T} M v.$$

Note that the positivity of the matrices W and M is crucial to deal with proper scalar products.

This triplet (X, W, M) allows us to define the M-inertia as a dispersal measurement,

$$I_M(x) = \sum_{i=1}^{n} w_i \|x_i - \bar{x}\|_M^2$$

and to consider the directions in which the amount of dispersion is maximal. To this end, we proceed as we did in the basic PCA: for some positive integer $d \leqslant p$, we look for M-orthonormal vectors $v^1, \ldots, v^d \in \mathbb{R}^p$ (*i.e.* $\langle v^j, v^k \rangle_M = 0$ if $j \neq k$ and 1 otherwise) such that the inertia $I_M(\pi_d x)$ is maximal where π_d is the orthogonal projection onto the linear space E_d generated by $v^1, \ldots, v^d$. As in (1.3), denoting $\Sigma = \bar{X}^\mathsf{T} W \bar{X}$, we obtain

$$\begin{aligned}
I_M(\pi_d x) &= \sum_{i=1}^{n} w_i \sum_{k=1}^{d} \langle x_i - \bar{x}, v^k \rangle_M^2 \\
&= \sum_{i=1}^{n} w_i \sum_{k=1}^{d} \left((x_i - \bar{x})^\mathsf{T} M v^k \right)^2 \\
&= \sum_{k=1}^{d} v^{k\mathsf{T}} M^\mathsf{T} \underbrace{\left(\sum_{i=1}^{n} w_i (x_i - \bar{x})(x_i - \bar{x})^\mathsf{T} \right)}_{\Sigma} M v^k \\
&= \sum_{k=1}^{d} \left(\Sigma M v^k \right)^\mathsf{T} M v^k \\
&= \sum_{k=1}^{d} \langle \Sigma M v^k, v^k \rangle_M. \qquad (1.5)
\end{aligned}$$

Proposition 1.3. *The matrix ΣM is nonnegative-definite and self-adjoint for the scalar product $\langle \cdot, \cdot \rangle_M$.*

Proof. Let $u, v \in \mathbb{R}^p$, because M and Σ are symmetric, we have

$$\langle \Sigma M u, v \rangle_M = u^\mathsf{T} M^\mathsf{T} \Sigma^\mathsf{T} M v = u^\mathsf{T} M (\Sigma M v) = \langle u, \Sigma M v \rangle_M.$$

Moreover, Σ is nonnegative-definite and, for any $u \in \mathbb{R}^p$,

$$\langle \Sigma M u, u \rangle_M = u^\mathsf{T} M^\mathsf{T} \Sigma^\mathsf{T} M u = (Mu)^\mathsf{T} \Sigma (Mu) \geqslant 0.$$

□

Proposition 1.3 implies the existence of $\lambda_1 \geqslant \cdots \geqslant \lambda_p \geqslant 0$ and a $p \times p$ matrix V whose columns form a M-orthonormal basis $v^1, \ldots, v^p$ of $\mathbb{R}^p$ such that

$$V^\intercal M \Sigma M V = \Lambda = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \lambda_p \end{pmatrix}$$

The d first vectors $v^1, \ldots, v^d$ lead to the maximal value of (1.5) which is $\lambda_1 + \cdots + \lambda_d$.

The coordinates of the data in the M-orthonormal basis are given by the principal components matrix, namely

$$C = \bar{X} M V$$

For any $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, p\}$, we have $c_i^j = \langle x_i - \bar{x}, v^j \rangle_M$ and the principal components $c^1, \ldots, c^p$ are noncorrelated and $\sigma^2(c^j) = \lambda_j$. Then, we have at our disposal all the tools and the quantities defined in the basic PCA to discuss and analyze these principal components.

1.5 Example: Multiple Discriminant Analysis

1.5.1 Introduction

The question that we want to address in this section is in relation with **classification**. In this framework, we have at our disposal p real variables $x^1, \ldots, x^p$ which we observe n times. Moreover, for each individual, we also observe a categorical variable t, called **label**, which can take m distinct values in $\mathbb{T} = \{\tau_1, \ldots, \tau_m\}$. Thus, our data set is given by the observations of $(x_1, t_1), \ldots, (x_n, t_n) \in \mathbb{R}^p \times \mathbb{T}$. As usual in classification, the goal is to provide a procedure solely based on the data which takes a vector $x \in \mathbb{R}^p$ as input and predicts the associated value of t as output.

Many methods to handle this kind of problem have been proposed and we will see some of them in the sequel of this lecture. In practice, a PCA procedure is sometimes used to do it. For example, the representation of the data in the principal plane E_2 could be considered to distinguish different groups of data. But how far such an approach is valid? Actually, we never claim that PCA was a good tool to discrimate groups. Such an analysis is only adapted to get a good representation and, a priori, there is no connection with label discrimination.

The idea consists in defining a dispersal measurement $I_M(x)$ based on a matrix M that takes into account the labels $t_1, \ldots, t_n \in \mathbb{T}$ and not only the values of $x_1, \ldots, x_n \in \mathbb{R}^p$. This approach aims to illustrate the flexibility of the general outlook introduced previously. Now, we have to clarify what is our triplet (X, W, M) to this end.

1.5.2 Framework

As mentioned above, the data set is formed by n observations $(x_1, t_1), \ldots, (x_n, t_n) \in \mathbb{R}^p \times \mathbb{T}$. As usual, the $n \times p$ matrix X designates the data matrix related to the observations of the real

variables $x^1, \ldots, x^p$ and we consider some weights $w_1, \ldots, w_n > 0$ with the associated diagonal matrix W.

The observations of the categorical variable t induce a partition of $\{1, \ldots, n\}$ into m separate subsets $\Omega_1, \ldots, \Omega_m$ defined by, for any $j \in \{1, \ldots, m\}$,

$$\Omega_j = \left\{ i \in \{1, \ldots, n\} \text{ such that } t_i = \tau_j \right\}.$$

This partition can be encoded in a $n \times m$ matrix T which has only one nonzero element in each line. Namely, for any $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$,

$$T_{ij} = \begin{cases} 1 & \text{if } i \in \Omega_j, \\ 0 & \text{otherwise.} \end{cases}$$

This matrix T makes it easier to handle the groups induced by the labels. For instance, instead of explicitly defining weights for each group by, for any $j \in \{1, \ldots, m\}$,

$$\bar{w}_j = \sum_{i \in \Omega_j} w_i,$$

we can directly introduce the $m \times m$ diagonal matrix $\bar{W}$ with

$$\bar{W} = T^{\mathsf{T}} W T = \begin{pmatrix} \bar{w}_1 & 0 & \ldots & 0 \\ 0 & \bar{w}_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \bar{w}_m \end{pmatrix}.$$

Note that we implicitely assume in the sequel that all the labels are at least observed one time, *i.e.* $\bar{w}_1, \ldots, \bar{w}_m > 0$.

1.5.3 Procedure

To take into account the labels, the key idea is to consider the means of each group. Thus, for any $j \in \{1, \ldots, m\}$ and any $k \in \{1, \ldots, p\}$, we introduce the mean $\bar{x}_j^k$ of the observations of x^k in Ω_j,

$$\bar{x}_j^k = \frac{1}{\bar{w}_j} \sum_{i \in \Omega_j} w_i x_i^k.$$

Then, treating these means by group as data, we consider the $m \times p$ data matrix of means G given by

$$G = \begin{pmatrix} \bar{x}_1^1 & \ldots & \bar{x}_1^p \\ \vdots & \vdots & \vdots \\ \bar{x}_m^1 & \ldots & \bar{x}_m^p \end{pmatrix}.$$

We can now insert the means of the groups in our procedure by using a proxy $n \times p$ data matrix X_b where each observation has been replaced by the mean of the group to which it belongs,

$$X_b = TG.$$

This proxy matrix allows us to decompose the centred data matrix $\bar{X}$,

$$\bar{X} = \bar{X}_w + \bar{X}_b$$

where $\bar{X}_w = X - X_b$ and $\bar{X}_b$ is the centred version of X_b, *i.e.* for any $j \in \{1,\dots,p\}$, $\overline{x^j}$ has been subtracted to the j-th column of X_b.

Theorem 1.1. *The covariance matrix $\Sigma = \bar{X}^\intercal W \bar{X}$ is the sum of*

- *the **covariance within matrix** $\Sigma_w = \bar{X}_w^\intercal W \bar{X}_w$,*
- *the **covariance between matrix** $\Sigma_b = \bar{X}_b^\intercal W \bar{X}_b$.*

Namely, we have the decomposition

$$\Sigma = \Sigma_w + \Sigma_b.$$

Proof. Let us expand the definition of Σ,

$$\Sigma = \bar{X}^\intercal W \bar{X} = (\bar{X}_w^\intercal + \bar{X}_b^\intercal) W (\bar{X}_w + \bar{X}_b) = \Sigma_w + \Sigma_b + A + A^\intercal$$

where we have set $A = \bar{X}_w^\intercal W \bar{X}_b$. To get the announced result, we need to prove that the $p \times p$ matrix A is zero. Let $j,k \in \{1,\dots,p\}$, we have

$$\begin{aligned}
A_{jk} &= \sum_{i=1}^{n} w_i \bar{X}_{w,ij} \bar{X}_{b,ik} \\
&= \sum_{i=1}^{n} w_i (x_i^j - X_{b,ij})(X_{b,ik} - \overline{x^k}) \\
&= \sum_{\ell=1}^{m} \sum_{i \in \Omega_\ell} w_i (x_i^j - \bar{x}_\ell^j)(\bar{x}_\ell^k - \overline{x^k}) \\
&= \sum_{\ell=1}^{m} (\bar{x}_\ell^k - \overline{x^k}) \left[\bar{w}_\ell \bar{x}_\ell^j - \bar{w}_\ell \bar{x}_\ell^j \right] = 0.
\end{aligned}$$

□

Theorem 1.1 contains an important statistical idea: Σ_b stands for what happens between the groups and Σ_w is related to what happens inside the groups. The aim of a multiple discriminant analysis (MDA) is to get the best linear discrimination between the groups. Thus, this is the term with Σ_b that will lead us to our goal.

Let M be some $p \times p$ symmetric positive definite matrix and $v^1,\dots,v^d \in \mathbb{R}^p$ be M-orthonormal vectors. From Theorem 1.1 and (1.5), we know that the M-inertia of $\pi_d x$ is given by

$$I_M(\pi_d x) = \sum_{k=1}^{d} \langle \Sigma M v^k, v^k \rangle_M = \sum_{k=1}^{d} \langle \Sigma_w M v^k, v^k \rangle_M + \sum_{k=1}^{d} \langle \Sigma_b M v^k, v^k \rangle_M.$$

We want to maximize what happens between the groups with respect to the global dispersal according to $v^1, \dots, v^d \in \mathbb{R}^p$, namely

$$\frac{\sum_{k=1}^{d} \langle \Sigma_b M v^k, v^k \rangle_M}{\sum_{k=1}^{d} \langle \Sigma M v^k, v^k \rangle_M}.$$

Stated in such a way, this problem is not the same as the one we have in a PCA procedure. Indeed, the presence of the vectors $v^1, \dots, v^d$ in the denominator prevents us from treating it in the same way. However, we can take $M = \Sigma^{-1}$ (or, at least, its right pseudoinverse if Σ is not invertible) to obtain an easier problem to solve: finding Σ^{-1}-orthonormal vectors $v^1, \dots, v^d \in \mathbb{R}^p$ which maximize the quantity

$$\sum_{k=1}^{d} \langle \Sigma_b \Sigma^{-1} v^k, v^k \rangle_{\Sigma^{-1}}. \tag{1.6}$$

The choice of $M = \Sigma^{-1}$ is known to endow the space of individuals with a distance called **Mahalanobis distance**. Problem (1.6) is now the same as the one introduced in the general outlook for PCA. Computing a MDA amounts then to do the PCA of the triplet (X_b, W, Σ^{-1}).

Because the rank of the matrix $\Sigma_b \Sigma^{-1}$ can not be larger than $\kappa = \min\{m-1, p\}$, we know that we just have to compute the κ first eigenvectors $v^1, \dots, v^\kappa \in \mathbb{R}^p$. These vectors give the columns of the matrix V and we define the principal component matrix as usual,

$$C = \bar{X}_b \Sigma^{-1} V.$$

Because $\bar{X}_b$ contains several times the same lines which correspond to the means of the groups, it can be faster to simply compute $\bar{G}\Sigma^{-1}V$ where $\bar{G}$ is the centred version of G. Moreover, the coordinates of the initial individual can also be computed with the matrix $\bar{X}\Sigma^{-1}V$. Then, we can simultaneously plot the data and the means of the groups in the principal plane, for instance. Actually, we have at our disposal all the tools defined for a PCA: quality of the representation, cosines, biplot, ... (see Practicals for an example).

Practicals 1 : R and Basic PCA

Data are available from the website of the author:

https://www.math.univ-toulouse.fr/~xgendre

Refresh R

During all the practicals in this document, we use the R software. R is a free software environment for statistical computing and graphics. There are various ways to work with R and we do not want to impose any specific solution. At his or her own convenience, the reader can follow the explanations with a raw R interpreter or use a graphics program like R Commander, RStudio, ... In the sequel, the R commands will be written in `verbatim` style and R commands to be experimented are highlighted in the following way,

```
print("Hello World!")
```

For the data files, we have paid a particular attention to follow CSV standards. Thus, to read these files, the function `read.csv` should always be sufficient:

```
my_data <- read.csv("grades.csv")
```

In the practicals, we will handle matrices and linear algebra operations. We feel that it is appropriate to remind the most common commands to this end.

```
# Create a matrix
m <- matrix(rnorm(100), nrow=25, ncol=4)

# Transpose matrix
```

```
mt <- t(m)

# Matrix multiplication operator is %*%
mm <- t(m) %*% m

# Inverse matrix
mminv <- solve(mm)

# Eigendecomposition
eg <- eigen(mm)
print(eg$values) # Ordered eigenvalues
print(eg$vectors) # Associated eigenvectors
```

The command `apply` and its related functions (`sapply`, `lapply`, ...) are one of the great strengths of R. Common usage consists in giving a matrix, specifying a margin (`1` for the lines and `2` for the columns) and a function to apply to all the given vectors. If this function takes more than one vector as argument, they can be specified after it).

```
# Compute the sum along each row
apply(m, 1, sum)
rowSums(m) # To verify ...

# Compute the means of the columns
apply(m, 2, mean)
colMeans(m) # To verify ...

# With your custom function (yes, you can return vectors)
apply(m, 2, function(v) { return(c(mean(v), var(v))) })

# More arguments (without return, the last value is returned)
apply(m, 2, function(v, a) { sum(v) + a }, 17)
colSums(m) + 17 # To verify ...
```

Basic PCA

To illustrate the basic PCA procedure, we propose to work with a toy example given by some fictive grades obtained by $n = 9$ students for $p = 4$ academic subjects, namely 'Mathematics', 'Physics', 'French' and 'English'.

```
grades_data <- read.csv("grades.csv")
X <- as.matrix(grades_data)
```

A first exploratory step consists in computing the usual statistical quantities for each variable.

```
colMeans(X)
apply(X, 2, var)
apply(X, 2, function(x) { mean((x - mean(x))^2) })
```

- Explain the differences between the results of the two last lines.

The function `summary` can also be used to get a bit more details about the variables and the function `boxplot` produces the box plots.

```
summary(X)
boxplot(X)
```

- Compare the ranges of each variable. In particular, note the fact that all the observations are at the same scale. In other word, the dispersion of the data is similar for each variable.

To compute the PCA, we begin by centring the data matrix and computing the weight and covariance matrices.

```
Xbar <- scale(X, scale=FALSE)
W <- diag(rep(1 / nrow(X), nrow(X))) # Uniform weights
Sigma <- t(Xbar) %*% W %*% Xbar
```

- Explain the command based on the `scale` function.
- Print the box plots of the centred variables.
- Compare the matrix `Sigma` and what you obtain with `cov(X)`.

We now compute the eigendecomposition of the covariance matrix Σ,

```
pca <- eigen(Sigma)

pca$values # Eigenvalues
pca$vectors # Eigenvectors
```

We first look at the eigenvalues:

```
cumsum(pca$values) / sum(pca$values)
```

- Explain what is computed by the previous command.
- What is the ratio of inertia explained by the principal plane?
- How many principal directions should be kept to explain a sufficient amount of dispersion?

A common way to discuss about the number of principal directions to keep is to consider the diagram plotting the decreasing eigenvalues.

```
plot(pca$values,
  type='b',
  las=1,
  ylab="Eigenvalues",
  xaxp=c(1, 4, 3))
```

- Understand the various graphical options used in this command.
- Compare this approach to the cumulated sums.

The matrix of eigenvectors gives us the coordinates of the individuals in the associated orthonormal basis and the possibility to plot them in the principal plane,

```
# Principal component matrix
C <- Xbar %*% pca$vectors

# Representation in the principal plane
plot(C[,1:2], xlab="Component 1", ylab="Component 2")
abline(h=0, lty=2, col="gray")
abline(v=0, lty=2, col="gray")
text(C[,1:2], rownames(X), pos=3)
```

- Focus on Benny and Coby. What are their respective positions in the principal plane? What are their grades? Comment.

To improve the readability of the results, we can compute the cosines to quantify the quality of the representation of each individual.

```
cos2 <- apply(C, 1, function(v) { (v[1]^2 + v[2]^2) / sum(v^2) })

# A new plot
plot(C[,1:2],
  xlab="Component 1",
```

```
  ylab="Component 2",
  col="transparent")
abline(h=0, lty=2, col="gray")
abline(v=0, lty=2, col="gray")
text(C[,1:2], rownames(X), cex=cos2)
```

- Comment the values contained in `cos2`.
- In the new plot, what is the role of `cex=cos2`? Although it is not really visible here, explain why it could be useful.

We now want to understand the relation between the principal components and the initial variables. Thus, we begin by computing the correlation coefficients,

```
Lambda <- diag(pca$values)
SigmaInvDiag <- diag(1 / diag(Sigma))

Rho <- sqrt(SigmaInvDiag) %*% pca$vectors %*% sqrt(Lambda)
rownames(Rho) <- colnames(X)
colnames(Rho) <- paste('C', 1:ncol(X), sep="")
```

- Inspect the matrix `Rho`. What are the signs of the correlation coefficients with the first principal component? And with the second?

To graphically interpret these correlations, we have at our disposal the correlation circle,

```
# Draw a circle
theta <- seq(0, 2*pi, length.out=256)
plot(cos(theta), sin(theta),
  xlab="Component 1",
  ylab="Component 2",
  type='l', asp=1)
abline(h=0, lty=2, col="gray")
abline(v=0, lty=2, col="gray")

# Plot the arrows
for (i in seq_len(nrow(Rho))) {
  arrows(0, 0, Rho[i,1], Rho[i,2], length=0.1, col="red")
  text(1.1*Rho[i,1], 1.1*Rho[i,2], rownames(Rho)[i], col="red")
}
```

- Comment the quality of the representation of each variable.

- Graphically retrieve the signs of the correlation coefficients with the first two components.
- Interpret the axes with respect to this correlation circle.

To discuss about the positions of the individuals in the principal plane, we consider the biplot diagram, *i.e.* the principal plane with the correlation circle in the same diagram,

```
# Some parameters to look pretty
xmax <- max(abs(C[,1]))
xrange <- c(-xmax, xmax)
ymax <- max(abs(C[,2]))
yrange <- c(-ymax, ymax)

# Principal plane
plot(C[,1:2], xlim=xrange, ylim=yrange,
  xlab="Component 1",
  ylab="Component 2",
  col="transparent", asp=1)
abline(h=0, lty=2, col="gray")
abline(v=0, lty=2, col="gray")
text(C[,1:2], rownames(X), cex=cos2)

# Correlation circle (overprint)
old_par <- par(new=TRUE)
plot(cos(theta), sin(theta),
  xlab="", ylab="", bty='n', xaxt='n', yaxt='n',
  type='l', col="gray", lty=2, asp=1)
for (i in seq_len(nrow(Rho))) {
  arrows(0, 0, Rho[i,1], Rho[i,2], length=0.1, col="red")
  text(1.1*Rho[i,1], 1.1*Rho[i,2], rownames(Rho)[i], col="red")
}
par(old_par)
```

- Spend some time to understand the graphical commands.
- Interpret the positions of the individuals in the principal plane. For example, what does it mean to be on the right side of the principal plane? At the bottom? At the top left?

Scale-dependency

Let us introduce the data set that we will handle in the next practicals about bows and crossbows of the video game "The Elder Scrolls V: Skyrim". This set contains the statistics of $n = 14$ weapons given by the $p = 4$ variables 'Damage', 'Speed', 'Value' and 'Weight'.

```
skyrim_bows_data <- read.csv("skyrim_bows.csv")
X <- as.matrix(skyrim_bows_data)
```

As above, we get some details with `summary` and `boxplot`.

```
summary(X)
boxplot(X)
```

- Compare the ranges of each variable. Does it seem weel-balanced?
- In terms of dispersion, how many directions do you need to represent such a data set?

To give a more precise answer to the last question, we propose to quickly compute the PCA of this data set.

```
Xbar <- scale(X, scale=FALSE)
Sigma <- t(Xbar) %*% Xbar / nrow(X) # Equivalent to unif. weights
pca <- eigen(Sigma)

cumsum(pca$values) / sum(pca$values)
plot(pca$values,
  type='b',
  las=1,
  ylab="Eigenvalues",
  xaxp=c(1, 4, 3))
```

- Comment these results with respect to the previous question.
- Draw the correlation circle and discuss about the arrow associated with the variable 'Value'.

To avoid such a drawback due to scale-dependency of the variance, a basic solution consists to deal with the reduced centred version of the variables,

```
Xbar <- scale(X, scale=sqrt(diag(Sigma)))
```

- Apply the basic PCA procedure to this reduced variables. Plot the biplot diagram and comment the results.
- In particular, which is the principal component that permits to distinguish the bows from the crossbows? Discuss its correlation coefficients with the initial variables.

Practicals 2 : Advanced PCA

Data are available from the website of the author:

https://www.math.univ-toulouse.fr/~xgendre

Reduced variables

In order to introduce the general outlook on PCA that we have studied previously, we handle data about bows and crossbows from the video game "The Elder Scrolls V: Skyrim". This set contains the statistics of $n = 14$ weapons given by the $p = 4$ variables 'Damage', 'Speed', 'Value' and 'Weight'.

```
skyrim_bows_data <- read.csv("skyrim_bows.csv")
X <- as.matrix(skyrim_bows_data)
```

In the previous practicals, we have seen that the scales of the observations vary a lot from one variable to the other. To avoid this problem, we have reduced the variables to force the variances to be equal to one. We have seen in the lecture that such a procedure can be seen through the triplet (X, W, M) where the matrices W and M endow the space of variables and the space of individuals with the scalar products $\langle \cdot, \cdot \rangle_W$ and $\langle \cdot, \cdot \rangle_M$, respectively.

```
# Weight matrix
W <- diag(1 / nrow(X), nrow(X))

# Matrix M
M <- diag(1 / apply(X, 2, function(x) { mean((x - mean(x))^2) }))
```

The idea beyond the general outlook is to avoid modifying the data. Instead, we want to consider an appropriate scalar product and its associated distance given by the corresponding norm. In such a way, we bring under control the meaning of closeness between data. Thus, we define similar objects but based on the triplet (X, W, M).

```
# Variable means with respect to W
Xmeans <- apply(X, 2, weighted.mean, diag(W))

# Centred data matrix
Xbar <- scale(X, center=Xmeans, scale=FALSE)

# Covariance matrix
Sigma <- t(Xbar) %*% W %*% Xbar
```

- Understand the usage of the function `scale`.

The quantity that we want to preserve is now the M-inertia which can be computed as a weighted sum similarly to the variance.

```
# Square of the norm induced by M
normM2 <- function(v, M) { t(v) %*% M %*% v }

# M-inertia
weighted.mean(apply(Xbar, 1, normM2, M), diag(W))
```

- Understand how we compute the M-inertia.
- Explain why the value of the M-inertia is equal to p in this example.

Our goal is now to determine the eigendecomposition of the matrix ΣM with respect to the scalar product $\langle \cdot, \cdot \rangle_M$. There is not straightforward algorithm do get such a decomposition but the problem can easily be rewritten. Indeed, for some positive integer $d \leqslant p$, we want to maximize the following quantity according to the M-orthonormal vectors $v^1, \dots, v^d \in \mathbb{R}^p$,

$$\sum_{k=1}^{d} \langle \Sigma M v^k, v^k \rangle_M = \sum_{k=1}^{d} v^{k\mathsf{T}} M \Sigma M v^k = \sum_{k=1}^{d} \left(M^{1/2} v^k \right)^{\mathsf{T}} M^{1/2} \Sigma M^{1/2} \left(M^{1/2} v^k \right)$$

where $M^{1/2}$ is the square root matrix of M, *i.e.* the symmetric matrix such that $M = M^{1/2} \times M^{1/2}$. Note that such a matrix exists because M is symmetric and positive definite.

For any $k \in \{1, \dots, d\}$, we set $u^k = M^{1/2} v^k$. By M-orthonormality of the vectors $v^1, \dots, v^d$, we know that the vectors $u^1, \dots, u^d$ are orthonormal with respect to the usual scalar product $\langle \cdot, \cdot \rangle$ in $\mathbb{R}^p$. Indeed, for any distinct $j, k \in \{1, \dots, d\}$,

$$\langle u^j, u^k \rangle = \langle v^j, v^k \rangle_M = 0 \quad \text{and} \quad \|u^j\|^2 = \|v^j\|_M^2 = 1.$$

In other words, we are looking for d orthonormal vectors $u^1, \dots, u^d \in \mathbb{R}^p$ which maximize the quantity

$$\sum_{k=1}^{d} \langle \Sigma M v^k, v^k \rangle_M = \sum_{k=1}^{d} \langle M^{1/2} \Sigma M^{1/2} u^k, u^k \rangle.$$

Because the problem is now stated with respect to the usual scalar product, we know that $u^1, \dots, u^d$ are the first d eigenvectors of $M^{1/2}\Sigma M^{1/2}$ associated with the eigenvalues $\lambda_1 \geqslant \cdots \geqslant \lambda_d \geqslant 0$. The vectors $v^1, \dots, v^k$ are simply obtained by, for any $k \in \{1, \dots, d\}$,

$$v^k = M^{-1/2} u^k$$

where $M^{-1/2}$ is well defined by definite positivity of M. Obviously, the eigenvalues remains the same and the vectors $v^1, \dots, v^d$ are M-orthonormal, for any distinct $j, k \in \{1, \dots, d\}$,

$$\langle \Sigma M v^j, v^j \rangle_M = \langle M^{1/2}\Sigma M^{1/2} u^j, u^j \rangle = \lambda_j \quad \text{and} \quad \langle v^j, v^k \rangle_M = \langle u^j, u^k \rangle = 0.$$

Let us apply this procedure to compute the PCA from the general point of view.

```
# Square root of M
eg <- eigen(M)
Msqrt <- eg$vectors %*% diag(sqrt(eg$values)) %*% t(eg$vectors)
max(abs(Msqrt %*% Msqrt - M)) # To verify ...

# Eigendecomposition with respect to the usual scalar product
pca <- eigen(Msqrt %*% Sigma %*% Msqrt)

# Adjust the eigenvectors to get the vectors v^k
pca$vectors <- solve(Msqrt) %*% pca$vectors
```

- Explain how we compute $M^{1/2}$.
- Verify that the resulting vectors are M-orthonormal.

Now, we can proceed as usual with the values and vectors provided by the R object `pca`.

```
cumsum(pca$values) / sum(pca$values)
plot(pca$values,
  type='b', las=1, ylab="Eigenvalues", xaxp=c(1, 4, 3))
```

- What is the ratio of inertia explained by the principal plane?
- How many principal directions should be kept to explain a sufficient amount of dispersion?

We compute the principal component matrix and we have at our disposal all the tools provided by a PCA procedure.

```
# Principal component matrix
C <- Xbar %*% M %*% pca$vectors

# Quality of the individuals in the principal plane
cos2 <- apply(C, 1, function(v) { (v[1]^2 + v[2]^2) / sum(v^2) })

# Correlation coefficients
Lambda <- diag(pca$values)
SigmaInvDiag <- diag(1 / diag(Sigma))
Rho <- sqrt(SigmaInvDiag) %*% pca$vectors %*% sqrt(Lambda)
rownames(Rho) <- colnames(X)
colnames(Rho) <- paste('C', 1:ncol(X), sep="")
```

- Plot the individual in the principal plane.
- Comment the values contained in `cos2`. What is the bow the least well represented?
- Represent the individuals in the principal plane with sizes proportional to their square cosine.
- Plot the biplot diagram and retrieve the results obtained at the end of the previous practicals.

Multiple discriminant analysis

To illustrate the advantages of the general outlook during the lecture, we have introduced the MDA procedure. Considering a data set formed by p real variables and one categorical variable, the MDA is given by the PCA of the triplet (X_b, W, Σ^{-1}) where X_b is the $n \times p$ proxy data matrix where each observation has been replaced by the mean of the group to which it belongs.

We handle the Lubischew data set, namely $p = 6$ morphological measurements have been done on $n = 74$ insects from three species denoted by 'A', 'B' and 'C'. We store the observations of the real variables in a matrix `X` and the species categories in a vector `S`.

```
lubischew_data <- read.csv("lubischew.csv")
X <- as.matrix(lubischew_data[,1:6])
S <- lubischew_data[,7]
```

The initial motivation for MDA was the ability to discriminate some known groups in a data set. Sometimes, the results of a PCA are used for such a purpose but there is no justification for it. To realize it, we begin by quickly representing the Lubischew data set in the principal plane obtained from reduced centred variables.

```
# Compute PCA
Xbar <- scale(X, scale=FALSE)
Sigma <- t(Xbar) %*% Xbar / nrow(X)
Xbar <- scale(X, scale=sqrt(diag(Sigma)))
pca <- eigen(Sigma)
C <- Xbar %*% pca$vectors
cos2 <- apply(C, 1, function(v)  (v[1]^2 + v[2]^2) / sum(v^2) )

# Plot without the group information
plot(C[,1:2],
  xlab="Component 1", ylab="Component 2",
  pch=2, las=1, cex=cos2)

# Plot with the group information
plot(C[,1:2],
  xlab="Component 1", ylab="Component 2",
  las=1, col="transparent")
text(C[,1:2], labels=S, col=as.integer(S), cex=cos2)
```

- Justify why we choose to deal with the reduced centred versions of the real variables.
- Without the group information, do you think that the frontiers between the groups are clear?
- With the group information, do the frontiers correspond to what you expected? In particular between species 'A' and 'B'?
- According to the results of the PCA, does it seem clear how to build a rule to classify a new individual?

We now consider the MDA approach. We compute the various values and matrices involved in the procedure of a PCA of the triplet (X_b, W, Σ^{-1}).

```
n <- nrow(X)
p <- ncol(X)
m <- nlevels(S)

# Weight matrix
W <- diag(1/n, n)

# Matrix M
Xmeans <- apply(X, 2, weighted.mean, diag(W))
Xbar <- scale(X, center=Xmeans, scale=FALSE)
Sigma <- t(Xbar) %*% W %*% Xbar
```

```
M <- solve(Sigma)

# Group matrix
T <- matrix(0, nrow=n, ncol=m)
for (i in seq_len(n)) {
  T[i, as.integer(S)[i]] <- 1
}

# Matrix of means
Wbar <- t(T) %*% W %*% T
G <- solve(Wbar) %*% t(T) %*% W %*% X
Gbar <- G - matrix(Xmeans, nrow=m, ncol=p, byrow=TRUE)

# Proxy matrix
XB <- T %*% G
XBbar <- scale(XB, scale=FALSE)
```

- Take some time to identify the objects introduced during the lecture in the previous code.
- Understand how we compute the matrix of means G and its centred version $\bar{G}$.

The PCA of triplet can then be computed as above,

```
SigmaB <- t(XBbar) %*% W %*% XBbar

eg <- eigen(M)
Msqrt <- eg$vectors %*% diag(sqrt(eg$values)) %*% t(eg$vectors)

pca <- eigen(Msqrt %*% SigmaB %*% Msqrt)
pca$vectors <- solve(Msqrt) %*% pca$vectors
```

- Again, spend some time to understand the role of each matrix of the triplet (X_b, W, Σ^{-1}) in the previous code.

Thus, we can handle the tools provided by any PCA. As noticed during the lecture, the principal component matrix contains a lot of duplications and it is sufficient to compute the coordinates of $\bar{G}$ rows only.

```
cumsum(pca$values) / sum(pca$values)
C <- Gbar %*% M %*% pca$vectors
```

- Explain why the explained inertia is equal to 1 with the first two principal directions.

We can also consider the coordinates of the individuals in the basis given by the PCA and represent them together with the means of the groups in the principal plane.

```
# Coordinates of the individuals
CInd <- Xbar %*% M %*% pca$vectors

# Principal plane
plot(CInd[,1:2],
  xlab="Component 1", ylab="Component 2",
  las=1, col=as.integer(S), pch=19)
points(C[,1:2], pch=15, cex=1.5)
```

- Compute the quality of the representation of each individual in the principal plane and comment the result.
- Compare the graphical representation with the one obtained with the initial PCA. Which one is better to discriminate the groups.
- Compute the correlation coefficients between the initial variables and the two first principal components. Interpret the obtained values and use it to build a rule to classify a new individual.

2 — Model selection

2.1 Introduction

In mathematics, a **model** is a representation of a system based on mathematical concepts. There exists a wide range of models and it would be beyond the scope of this document to make an exhaustive review. When we handle several models to represent a system, the idea behind is that each model is given by a distinct set of assumptions. Comparing these models together is then a way to compare the sets of assumptions. Choosing one model among a collection according to some criteria is the goal of **model selection**.

In this chapter, we consider a **regression framework**. Let n and p be some positive integers, we observe a vector $y = (y_1, \dots, y_n)^\intercal \in \mathbb{R}^n$ and p vectors $x^1 = (x_1^1, \dots, x_n^1)^\intercal, \dots, x^p = (x_1^p, \dots, x_n^p)^\intercal \in \mathbb{R}^n$. The goal of a regression approach is to explain y with respect to $x^1, \dots, x^p$. To this end, we commonly consider a probabilistic point of view, seeing the observed vectors $y, x^1, \dots, x^p \in \mathbb{R}^n$ as the data set associated to n realizations of some random variables also denoted by $y, x^1, \dots, x^p$ for the purpose of notation. Thus, we can define the **regression function** $h : \mathbb{R}^p \to \mathbb{R}$ as the conditional expectation of y given $x = (x^1, \dots, x^p)$,

$$h(x) = h(x^1, \dots, x^p) = \mathbb{E}\left[y \mid x^1, \dots, x^p\right].$$

In other words, we can write the observations in the following way, for any $i \in \{1, \dots, n\}$,

$$y_i = h(x_i^1, \dots, x_i^p) + \varepsilon_i \tag{2.1}$$

where we have set $\varepsilon_i = y_i - h(x_i^1, \dots, x_i^p) \in \mathbb{R}$.

By definition, the vector $\varepsilon = (\varepsilon_1, \dots, \varepsilon_n)^\intercal \in \mathbb{R}^n$ can also be seen as the n realizations of a real random variable ε called **noise** and such that $\mathbb{E}[\varepsilon] = 0$. For simplicity, we assume that the variable ε is independent of the variables $x^1, \dots, x^p$ and that the realizations $\varepsilon_1, \dots, \varepsilon_n$ are independent and such that $\mathbb{E}[\varepsilon_i^2] = \sigma^2 < \infty$, for any $i \in \{1, \dots, n\}$, where σ^2 is supposed to be known. These assumptions can be relaxed but at the cost of more mathematics.

The models considered in this chapter are **linear**, *i.e.* the function h is assumed to satisfy

$$h(x^1, \dots, x^p) = \beta_0 + \beta_1 x^1 + \cdots + \beta_p x^p \tag{2.2}$$

for some $\beta_0, \dots, \beta_p \in \mathbb{R}$. While we restrict this chapter to such models, the questions raised here are more general and can be applied in a lot of other frameworks.

2.2 Linear regression models

Let us start with the simple situation of $p = 1$. We have at our disposal two vectors $x = (x_1, \dots, x_n)^\intercal, y = (y_1, \dots, y_n)^\intercal \in \mathbb{R}^n$ and we look for $\beta_0, \beta_1 \in \mathbb{R}$ to fit the model (2.2). We need a criterion to estimate these values and a popular method is **least squares** in which we pick the coefficients $\beta = (\beta_0, \beta_1)^\intercal \in \mathbb{R}^2$ to minimize the residual sum $R_n(\beta)$,

$$R_n(\beta) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2 .$$

Theorem 2.1. *The sum $R_n(\beta)$ is minimal for*

$$\hat{\beta}_1 = \frac{\sigma(x,y)}{\sigma^2(x)} \quad \text{and} \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where the involved quantities are computed with the uniform weights.

We don't prove this result because it is just a particular case of Theorem 2.2. The line given by the graphical representation of the **estimator** function $\hat{h}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$ is known as the **regression line**. With such a linear model and a new observation of x, we can apply $\hat{h}$ to predict a value $\hat{h}(x)$ for y. To quantify the quality of this predictor according to the observations, we can consider the value $R_n(\hat{\beta})$.

In the general case of $p \geqslant 1$, we have at our disposal the vectors $y, x^1, \dots, x^p \in \mathbb{R}^n$. Thus, we want to pick the coefficients $\beta = (\beta_0, \dots, \beta_p)^\intercal \in \mathbb{R}^{p+1}$ to minimize the following residual sum,

$$R_n(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_i^1 + \cdots + \beta_p x_i^p\right)^2 .$$

To state the problem in terms of matrices, we introduce the $n \times (p+1)$ matrix X whose the first column is the constant vector of ones and other columns are given by the vectors of $x^1, \dots, x^p$,

$$X = \begin{pmatrix} 1 & x_1^1 & \dots & x_1^p \\ 1 & x_2^1 & \dots & x_2^p \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n^1 & \dots & x_n^p \end{pmatrix} .$$

Thus, the residual sum to minimize becomes

$$R_n(\beta) = \|y - X\beta\|^2 = \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{p} \beta_j X_{ij}\right)^2 .$$

Theorem 2.2. *Assuming that the matrix X has full column rank, the unique minimizer of $\beta \mapsto R_n(\beta)$ is*

$$\hat{\beta} = (X^\intercal X)^{-1} X^\intercal y.$$

Then, the predicted values are $\hat{h}(x_1),\dots,\hat{h}(x_n)$ are given by the vector

$$X\hat{\beta} = X(X^\intercal X)^{-1}X^\intercal y.$$

*The matrix $H = X(X^\intercal X)^{-1}X^\intercal$ is a projection (i.e. $H^2 = H$) and is sometimes called the **hat matrix**.*

Proof. The function $\beta \mapsto R_n(\beta)$ is quadratic in $\beta_0,\dots,\beta_p$. Hence, we easily differentiate it with respect to β to get

$$\frac{\partial R_n}{\partial \beta}(\beta) = -2X^\intercal(y - X\beta) \quad \text{and} \quad \frac{\partial^2 R_n}{\partial \beta \partial \beta^\intercal}(\beta) = 2X^\intercal X.$$

Because X has full column rank, the matrix $X^\intercal X$ is positive definite and we obtain the unique minimizer of R_n by setting the first derivative to zero, $X^\intercal(y - X\beta) = 0$, namely

$$\hat{\beta} = (X^\intercal X)^{-1}X^\intercal y.$$

The matrix H is obviously a projection,

$$H^2 = X(X^\intercal X)^{-1}X^\intercal X(X^\intercal X)^{-1}X^\intercal = X(X^\intercal X)^{-1}X^\intercal = H.$$

□

By symmetry, the projection H is the orthogonal projection onto the space generated by the columns of X. To quantify the quality of the predicted value Hy, we consider

$$R_n(\hat{\beta}) = \|y - Hy\|^2.$$

Taking the probabilistic point of view, the expectation of this quantity is worthwhile,

$$\begin{aligned}\mathbb{E}\left[\|y - Hy\|^2\right] &= \mathbb{E}\left[\|(y - \mathbb{E}[Hy]) - (Hy - \mathbb{E}[Hy])\|^2\right] \\ &= \mathbb{E}\left[\|y - \mathbb{E}[Hy]\|^2\right] + \mathbb{E}\left[\|Hy - \mathbb{E}[Hy]\|^2\right]\end{aligned}$$

by orthogonality of $y - Hy$ and Hy. The first term $\mathbb{E}\left[\|y - \mathbb{E}[Hy]\|^2\right]$ is known as the **bias term** and the second one $\mathbb{E}\left[\|Hy - \mathbb{E}[Hy]\|^2\right]$ as the **variance term**. This decomposition plays a central role in this chapter. The bias term quantifies the capacity of the model to fit to the observations of y and the variance term measures the complexity of the model. Indeed, because $y_i = h(x_i) + \varepsilon_i$ for any $i \in \{1,\dots,n\}$ and by independence of $\varepsilon_1,\dots,\varepsilon_n$, we have

$$\begin{aligned}\mathbb{E}\left[\|Hy - \mathbb{E}[Hy]\|^2\right] &= \sum_{i=1}^n \mathbb{E}\left[((Hy)_i - \mathbb{E}[(Hy)_i])^2\right] \\ &= \sum_{i=1}^n \mathbb{E}\left[\left(\sum_{j=1}^n H_{ij}(y_j - \mathbb{E}[y_j])\right)^2\right] \\ &= \sum_{i=1}^n \mathbb{E}\left[\left(\sum_{j=1}^n H_{ij}\varepsilon_j\right)^2\right] \\ &= \sigma^2 \sum_{i=1}^n \sum_{j=1}^n H_{ij}^2 \\ &= \sigma^2 \sum_{i=1}^n (HH^\intercal)_{ii}.\end{aligned}$$

Because the projection H is symmetric, the variance term becomes $\sigma^2\mathrm{tr}(H) = \sigma^2(p+1)$ and is proportional to the dimension of the image space of H.

2.3 Example: Polynomial regression

To illustrate the procedure, we introduce the particular case of the **polynomial regression**. Let us consider some nonnegative integer p, the regression function $h : \mathbb{R} \to \mathbb{R}$ is assumed to be a polynomial of degree p, namely, for any $x \in \mathbb{R}$,

$$h(x) = \beta_0 + \beta_1 x + \cdots + \beta_p x^p$$

where we have denoted the coefficients by $\beta = (\beta_0, \dots \beta_p)^\intercal \in \mathbb{R}^p$.

Given two vectors $x = (x_1, \dots, x_n)^\intercal, y = (y_1, \dots, y_n)^\intercal \in \mathbb{R}^n$, this framework corresponds to the regression framework with variables given by the consecutive powers of x, *i.e.* a matrix X of the form

$$X_p = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^p \\ 1 & x_2 & x_2^2 & \dots & x_2^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^p \end{pmatrix}.$$

Thus, taking the observations $(x_1, y_1), \dots, (x_n, y_n)$ as points in $\mathbb{R}^2$, the curve of $t \in \mathbb{R} \mapsto \hat{h}(t)$ is the "best" polynomial of degree p, in the sense of least squares, to represent the relation between x and y. On the observed points, this relation is then predicted by $H_p y$ where $H_p = X_p(X_p^\intercal X_p)^{-1} X_p^\intercal$.

According to the results obtained in the previous section, the expected value of $R_n(\hat{\beta})$ is equal to

$$\mathbb{E}\left[\|y - \mathbb{E}[H_p y]\|^2\right] + \sigma^2(p+1). \tag{2.3}$$

The question that naturally arises is the choice of a "good" value of p. Indeed, for any value of $p \geqslant 0$, the procedure gives a polynomial of degree p but, if we want to predict values for new observations, we still have to choose a value of p in practice. A first idea could be to take p as large as possible. Indeed, with a large value of p, common approximation results ensure us that we will be able to approximate any reasonable function h to represent the relation between x and y. This naive point of view only focuses on the bias term in (2.3) by making it as small as possible but neglects the variance term which is large when p is large. On the other hand, considering a polynomial with a small degree p only to keep the variance term small is not smart either because it would lead to a poor approximation capacity and a potential large bias term. The idea of model selection lies in finding a **trade-off** between these two opposite situations with the help of the data.

The case of a too large value of p is known to lead to a phenomenon called **overfitting**. Without the expectation, the quantity $R_n(\hat{\beta})$ is simply equal to $\|y - H_p y\|^2$. Taking the large value $p = n-1$, we obtain a polynomial that perfectly fits to the data, by construction. However, if we want to predict the value $h(x')$ for some $x' \in \mathbb{R}$ that does not belong to the design set $\{x_1, \dots, x_n\}$, the Runge's phenomenon will lead to very poor predictions. The moral is that

fitting to much to the data, namely overfitting them, gives an estimator $\hat{h}$ that seems to be perfect on the data but offers bad capacities of prediction out of the design points. This explanation is summarized in Figure 2.1.

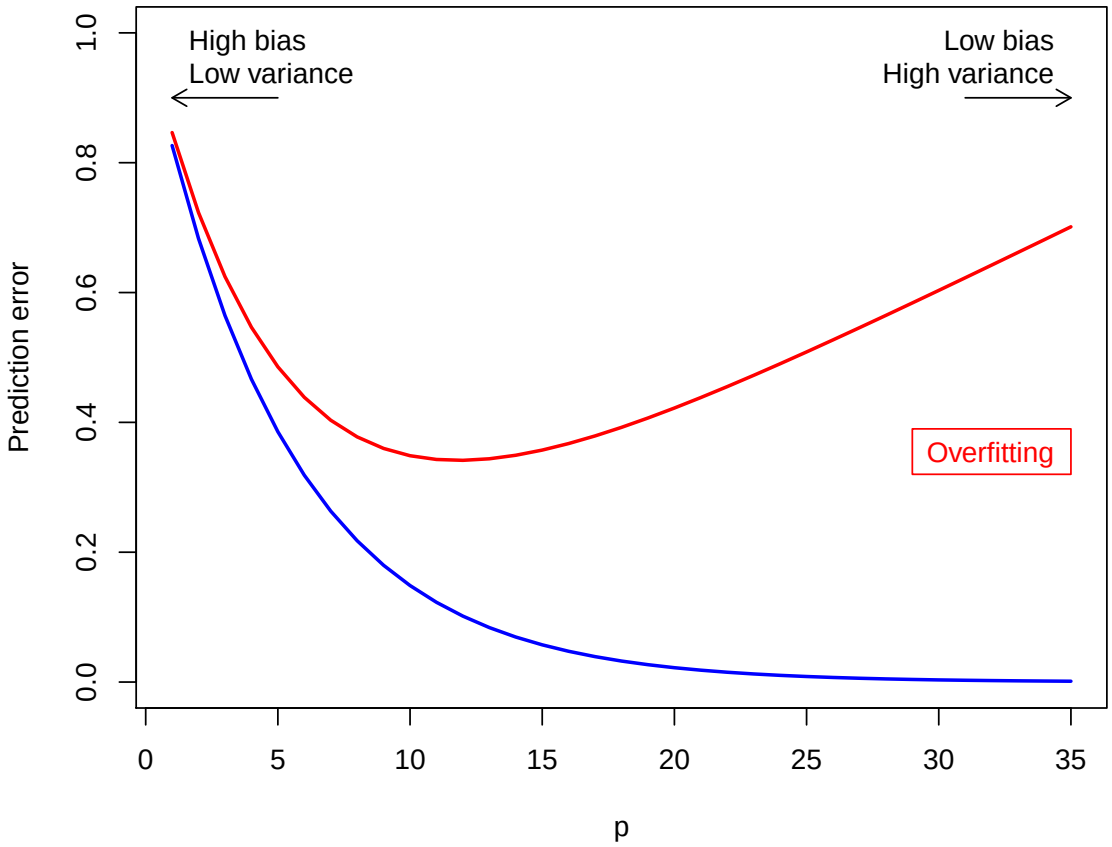


Figure 2.1: Prediciton error $R_n(\hat{\beta})$ (blue curve) and expected value $\mathbb{E}[R_n(\hat{\beta})]$ (red line).

2.4 Model selection criteria

We now discuss how to perform the model selection in itself. We consider that we have at our disposal the observed vector $y \in \mathbb{R}^n$ and a collection of orthogonal projections $\{H_m, m \in \mathscr{M}\}$ where $\mathscr{M}$ is some countable index space. These projections are built as above and the models by themselves are the image spaces of these projections. In the sequel, for any $m \in \mathscr{M}$, we denote by $\mathrm{tr}(H_m) = p_m + 1$ the dimension of the model and by $\hat{\beta}_m \in \mathbb{R}^{p_m}$ the minimizer of the residual sum.

Before introducing any selection method, we have to draw the reader's attention on a particular point. The collections considered in this chapter are said to be **nested** in the sense that any model in the collection with a dimension p is contained in all the models of the collection with

an higher dimension $p' > p$. For example, the model of polynomial with degree p is a linear subspace of the model of polynomial with degree $p' > p$. Such a property can be crucial in certain situations and larger collections of models can raise special problems. Although we do not consider such complications in the sequel, the interested reader has to keep in mind that the selection methods described hereafter could need some adjustments for more general frameworks.

A first popular criterion to select a model among a collection is known as **Mallows' C_p**. Given some $m \in \mathscr{M}$, let us pursue a little bit the computation of $\mathbb{E}[R_n(\hat{\beta}_m)]$. According to (2.1), we can write $y \in \mathbb{R}^n$ as $y = h + \varepsilon$ where $h \in \mathbb{R}^n$ is the unknown quantity of interest. Because ε is assumed to be centred, we obtain

$$\begin{aligned}
\mathbb{E}[R_n(\hat{\beta}_m)] = \mathbb{E}\left[\|y - H_m y\|^2\right] &= \mathbb{E}\left[\|y - \mathbb{E}[H_m y]\|^2\right] + \sigma^2(p_m + 1) \\
&= \mathbb{E}\left[\|h + \varepsilon - H_m h\|^2\right] + \sigma^2(p_m + 1) \\
&= \mathbb{E}\left[\|h - H_m h\|^2\right] + \sigma^2 n + \sigma^2(p_m + 1) \\
&= \left\{\mathbb{E}\left[\|h\|^2\right] + \sigma^2 n\right\} + \left\{-\mathbb{E}\left[\|H_m h\|^2\right] + \sigma^2(p_m + 1)\right\}.
\end{aligned}$$

The first part of the sum does not depend on m, then choosing a "good" model amounts to pick $m \in \mathscr{M}$ to minimize $-\mathbb{E}\left[\|H_m h\|^2\right] + \sigma^2(p_m + 1)$. Of course, the term $\mathbb{E}\left[\|H_m h\|^2\right]$ is not available to the statistician (otherwise, there would be nothing to do) but can be estimated by $\|H_m y\|^2 - \sigma^2(p_m + 1)$ because

$$\mathbb{E}\left[\|H_m y\|^2 - \sigma^2(p_m + 1)\right] = \mathbb{E}\left[\|H_m h\|^2\right].$$

The idea of Mallows' C_p is then to select a model by minimizing the following criterion according to $m \in \mathscr{M}$,

$$\gamma_M(m) = -\|H_m y\|^2 + 2\sigma^2(p_m + 1). \tag{2.4}$$

In such a way, we try to mimic the behavior of the "best" estimator among the collection $\{H_m y, m \in \mathscr{M}\}$, *i.e.* the one which leads to the trade-off discussed above.

Criterion like (2.4) belongs to a larger group of criteria named **penalized critera**. We call **penalty** any nonnegative function pen : $\mathscr{M} \to \mathbb{R}_+$ and we define the associated penalized criterion by

$$\gamma_{\text{pen}}(m) = -\|H_m y\|^2 + \text{pen}(m).$$

Mallows' C_p corresponds to the case of $\text{pen}(m) = 2\sigma^2(p_m + 1)$ and it is common to consider more general criteria given by penalties of the form $\text{pen}_\lambda(m) = \lambda\sigma^2(p_m + 1)$ for some $\lambda \geqslant 0$. The parameter λ allows to tune the procedure. When λ tends to zero, it leads to a situation of overfitting and if we let λ grow towards infinity, the models with large dimensions are overpenalized and so, never selected. Selecting a good value for λ is critical but we defer this discussion to the last chapter in which we consider cross-validation.

There is a vast literature on penalized criteria and we cannot provide an exhaustive review. Picking a penalty is related to with what you want to find a trade-off and such a decision depends on the goal of the procedure. Moreover, penalized criteria are not restricted to model selection and the penalty function can take into account more than just a model. For

example, some popular methods are based on penalties on the coefficients $\beta \in \mathbb{R}^{p+1}$ in order to find a trade-off between adequacy of the data and the norm of the estimator. Such an approach can make sense when we handle a large model but we are not looking for any "good" subspace, just a representative with a reasonable norm. To illustrate it, let us introduce the example of the **ridge regression** procedure. As above, we have at our disposal vectors $y, x^1, \dots, x^p \in \mathbb{R}^n$ and we want to pick some coefficients $\beta = (\beta_0, \dots, \beta_p)^\intercal \in \mathbb{R}^{p+1}$ to fit a model as (2.2). To shrink the coefficients $\beta_1, \dots, \beta_p \in \mathbb{R}$, we consider the ridge regression coefficients $\hat{\beta}^R = (\hat{\beta}_0^R, \dots, \hat{\beta}_p^R)^\intercal \in \mathbb{R}^{p+1}$ which minimize the penalized criterion

$$\gamma_R(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_i^j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

where $\lambda \geqslant 0$ is a tuning parameter. Note that the intercept β_0 has been left out of the penalty term to not make the procedure depend on the origin chosen for y.

2.5 Example: Principal components regression

To end this chapter, we propose to connect with what we have seen in the first chapter. Indeed, the regression estimator is nothing else than the projection of $y \in \mathbb{R}^n$ onto the space $S_p \subset \mathbb{R}^n$ generated by the constant vector and $x^1, \dots, x^p \in \mathbb{R}^n$. In other words, we can build the same object by considering any orthonormal basis of S_p and dealing with it to get the projection of y. In particular, we can do it with the orthonormal basis provided by the principal directions.

The idea behind such an approach is that often a small number of principal components suffice to explain most of the variability in the data, as well as the relationship with the response. Care must be taken to avoid generalizing the previous sentence, this is just the idea of principal components regression but there is no claim for universality. However, in practice, it often turns out to be a reasonable enough approximation to give good results.

Let us consider the model generated by the first principal direction only, then the model generated by the two first principal directions and so on. We have at our disposal a nice nested collection of models to run our model selection procedures. In such a way, we are able to decide how much principal components should be kept to get a reasonable approximation of the relationship with the response y. Of course, if you feel inspired, you can also consider shrinkage penalties to go further.

Practicals 3 : Model selection

Refresh R

Contrarily to the others, this practicals session does not make a great use of real data sets. Indeed, most of the following examples are based on simulated data. This decision is mainly motivated by two reasons: the need for estimate some expectations and the opportunity to present some R functions to generate random data. There exist some ways to address the first point but they often need some explanation that will be only tackled in the last chapter. Hence, manipulating simulated data sets is easier for now. The second point is the purpose of this section.

The first thing to bear in mind when we work with random objects in R is the **random number generator** (RNG) which is the algorithm used to generate a sequence of numbers whose properties approximate the properties of sequences of random numbers. The default algorithm in R is known as Mersenne-Twister but other RNG are available. We are not going into details about RNG but we have to mention an important feature of such algorithms, namely the fact that the sequence of numbers generated by the RNG is initiliazed with the help of a **seed**. A seed can take various forms according to the RNG but when it is fixed, the following sequence of numbers is then deterministic. In other words, with the same seed, you can reproduce the "randomness" of a simulation.

```
# Set the seed to 12345
set.seed(12345)

# Generate a uniform real number between 0 and 1 (see further)
runif(1)

# Reset the seed to 12345
set.seed(12345)

# Generated number is the same as the previous one!
```

```
runif(1)
```

Maybe the most basic ways to produce sequences of random numbers are the functions `sample` and `sample.int`. The first function is useful to pick some elements at random in a given set of values with or without replacement and according to some discrete probability distribution. The function `sample.int` is the short version to pick at random in the set of integers $\{1,\dots,n\}$. The latter is ideal to take a subsample of an index set.

```
# Toss 10 coins
sample(c('Head', 'Tail'), 10, replace=TRUE)

# Sample multinomial distribution
sample(c(-1, 0, 1), 100, replace=TRUE, prob=c(0.25, 0.5, 0.25))

# Get a subsample of 1:100 with size 10
sample.int(100, 10)
```

Of course, R is not restricted to discrete probabilities and provides a lot of generators for common distributions (more are also available with dedicated packages). Most of them are simply called by rNAME(n) where `n` is the sample size and `NAME` is an abbreviation for a distribution name (`runif` for uniform distribution, `rnorm` for normal distribution, ...). Some additional parameters can be needed with specific distributions. Moreover, R generally also supplies useful associated functions whose names follows the following patterns:

dNAME for density function,

pNAME for distribution function,

qNAME for quantile function.

```
# A list of available distributions
help(distribution)

# Sample of size 10 from uniform distribution between -1 and 1
runif(10, min=-1, max=1)

# Sample of size 42 from standard Gaussian distribution
rnorm(42)

curve(dnorm, from=-3, to=3) # Gaussian density function
curve(pnorm, from=-3, to=3) # Gaussian distribution function
curve(qnorm, from=0, to=1) # Gaussian quantile function
```

- Generate a sample of size 100 from a Gaussian distribution with mean 3 and variance 2. Notice that the third argument is the standard deviation, not the variance.
- Generate a sample of size 100 from an exponential distribution with parameter 3.14.
- Generate 100000 steps of an equilibrated random walk, *i.e.* ± 1 at each step with same probability. Plot the result.

Polynomial regression

As in the lecture, we start with the simple case of two vectors $x, y \in \mathbb{R}^n$. In such a case, the regression coefficients are given by

$$\hat{\beta}_1 = \frac{\sigma(x,y)}{\sigma^2(x)} \quad \text{and} \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}.$$

Let us consider some points in $[0,1]$ and a simple relationship between x and y up to some Gaussian noise.

```
# Generate some data
n <- 10
x <- seq(0, 1, length.out=n)
y <- x + rnorm(n, sd=0.1)

# Centred data
xbar <- x - mean(x)
ybar <- y - mean(y)

# Regression coefficients
beta1 <- mean(xbar * ybar) / mean(xbar^2)
beta0 <- mean(y) - beta1*mean(x)

# Plot the result
plot(x, y)
abline(a=beta0, b=beta1, col="red")
```

R supplies the very useful command `lm` to compute the regression coefficients. In our simple case, it can only take one argument with a specific type called `formula`. To write a formula, we need to specify the variable to explain followed by the symbol `~` and the variable on which we base the regression, *i.e.* here, it gives `y ~ x`. The object returned by `lm` provides several worthwhile informations and can be directly used in various R functions like `abline`, for example.

```
# About R formulas
help(formula)

# Regression with lm
model <- lm(y ~ x)
abline(model, col="blue", lty=3, lwd=2)

model$coefficients # Regression coefficients
sum(model$residuals^2) # Residual square sum
# Et cetera ...
```

- Compare the values `beta0` and `beta1` with the results obtained with the function `lm`.
- What is the component of `model` that gives the dimension of the model?

A very convenient feature of an object returned by `lm` is to easily allow prediction with the associated linear model. To this end, we use the function `predict` with two arguments, namely the model returned by `lm` and the values for which we want to predict an output. Bear in mind that the function `predict` expects to receive the new data as a data frame with similar component names as those we used in the call to `lm`. If the names differ, the new data are ignored and the function returns the predictions for the initial data with a warning message.

```
# Predictions for initial data
initial_pred <- predict(model)

# Predictions for new data
new_data <- runif(2) # Two random new points in [0,1]
new_pred <- predict(model, data.frame(x=new_data)) # Name is x ...
predict(model, data.frame(new_data)) # ... otherwise, it fails.

# Plot the result
plot(x, y, col="grey")
abline(h=0, lty=2, col="grey")
abline(v=0, lty=2, col="grey")
abline(model, col="red")
segments(x, y, x, initial_pred, col="orange")
points(new_data, new_pred, pch=19)
segments(new_data, c(0, 0), new_data, new_pred)
```

- Pay attention to the use of function `predict`.

We are now considering the general case of $p \geqslant 1$ for polynomial regression. More specifically, we suggest to estimate the previous simple relationship between x and y through a polynomial with degree $p = 9$.

```
# Generate some data
n <- 10
x <- seq(0, 1, length.out=n)
y <- x + rnorm(n, sd=0.1)

# Compute the regression coefficients
p <- 9
X <- matrix(1, nrow=n, ncol=1)
for (i in seq_len(p)) X <- cbind(X, x^i)
beta <- solve(t(X) %*% X) %*% t(X) %*% y

# Predict the relationship
nn <- 256
xx <- seq(0, 1, length.out=nn)
XX <- matrix(1, nrow=nn, ncol=1)
for (i in seq_len(p)) XX <- cbind(XX, xx^i)
yy <- XX %*% beta

# Plot the result
plot(x, y, ylim=range(yy))
points(xx, yy, type="l", col="red")
```

- Comment the final graphics. Does the estimated relationship seem reasonable?

As in the simple case, the function `lm` can provide the results of the regression procedure. Note the use of the function `I` to inhibit the operators in the formula. Similarly, the function `predict` applied to the object returned by `lm` can supply predictions for new data.

```
# Regression with lm
formula_str <- paste(
  "y ~",
  paste("I(x^", 1:9, ")", sep="", collapse=" + ")
)
model <- lm(formula(formula_str))

# Regression coefficients
model$coefficients

# Plot the predicted function between 0 and 1
yy_pred <- predict(model, data.frame(x=xx))
plot(x, y, col="grey", ylim=range(yy_pred))
points(xx, yy_pred, type="l", col="red")
```

- Inspect and understand the content of the variable `formula_str`.

- Compare the components of `beta` and `model$coefficients`.

R supplies a useful function `poly` to produce orthogonal polynomials over a specified set of points. This function provides a matrix whose columns are the values taken by these orthogonal polynomials with degrees from 1 to p. In other words, it make easier the call to `lm` function.

```
# Equivalent regression as above
model <- lm(y ~ poly(x, 9))

# Plot the predicted function between 0 and 1
yy_pred <- predict(model, data.frame(x=xx))
plot(x, y, col="grey", ylim=range(yy_pred))
points(xx, yy_pred, type="l", col="red")
```

Overfitting

Of course, the phenomenon observed in the previous section is overfitting. Indeed, with $n = 10$ and $p = 9$, polynomial regression provides a polynomial that perfectly fits to the data. During the lecture, we have seen that such a phenomenon leads to a poor capacity to predict on new data. The goal of this section is to illustrate that.

We begin by defining a data set similar to the previous one but with more design points and a richer polynomial relation between x and y.

```
n <- 256
x <- seq(0, 1, length.out=n)
s <- 12*x^7 - 10*x^5 + x + 1
y <- s + rnorm(n)
```

Let us compute the residual sums for polynomials with degree varying from 1 to 25,

```
pmax <- 25
res <- rep(0, pmax)
for (i in seq_len(pmax)) {
  model <- lm(y ~ poly(x, i))
  res[i] <- sum(model$residuals^2)
}
plot(res, type="b")
```

- Observe how the residual square sum decreases when the degree increases.

We now focus on the expectation of this residual sum. To represent these quantities, we suggest to evaluate simultaneously the value of the residual sum on data and the value of residual sum on prediction for some newly generated data set. Repeating this procedure `nrep` times provides us a way to illustrate the phenomenon described in Figure 2.1.

```
# Initialization
nrep <- 100
pmax <- 25
error_on_data <- matrix(0, nrow=nrep, ncol=pmax)
error_on_prediction <- matrix(0, nrow=nrep, ncol=pmax)

# Repeat the procedure nrep times
for (i in seq_len(nrep)) {
  y <- s + rnorm(n)
  for (j in seq_len(pmax)) {
    # Compute the regression with degree j
    model <- lm(y ~ poly(x, j))
    error_on_data[i,j] <- sum(model$residuals^2)
    # Predict values on new data
    new_predict <- predict(model)
    new_out <- s + rnorm(n)
    error_on_prediction[i, j] <- sum((new_out - new_predict)^2)
  }
}

# Plot the results
plot(1:pmax,
  ylim=range(error_on_data, error_on_prediction),
  ylab="Error", col="transparent")
for (i in seq_len(nrep)) {
  points(error_on_data[i,], type="l", col="lightblue")
  points(error_on_prediction[i,], type="l", col="pink")
}
points(colMeans(error_on_data), type="l", col="blue")
points(colMeans(error_on_prediction), type="l", col="red")

# Mark some degrees
abline(v=1, lty=2)
abline(v=5, lty=3)
abline(v=25, lty=4)
```

- Understand the different steps of this code.
- Explain what is represented by red and blue lines.
- The blue line and the read line diverge on the right of the graphic, to which phenomenon is it due to?

Let us have a look to the situations with marked degrees.

```
# Plot the data with the real s (usually unknown)
y <- s + rnorm(n)
plot(x, y, col="grey", pch=4)
points(x, s, type="l", col="red")
legend("bottomright", c("p=1", "p=5", "p=25"), lty=2:4)

# Degree 1
model <- lm(y ~ poly(x, 1))
points(x, predict(model), type="l", lty=2)

# Degree 5
model <- lm(y ~ poly(x, 5))
points(x, predict(model), type="l",  lty=3)

# Degree 25
model <- lm(y ~ poly(x, 25))
points(x, predict(model), type="l",  lty=4)
```

- Between the three models, which one seems to be the "best"?
- Spend some time to think about the trade-off discussed during the lecure.

Model selection criteria

We now want to apply the model selection procedure called Mallows' C_p. To this end, for each polynomial model, we compute the criterion and the penalty. Note that we use here the fact that the variance factor $\sigma^2 = 1$ is assumed to be known. The estimated trade-off is given by the model that minimizes the sum of these two terms.

```
# Generate some data
n <- 256
x <- seq(0, 1, length.out=n)
s <- cos(2*pi*x) + sin(6*pi*x)
y <- s + rnorm(n)

# Initialization
pmax <- 25
crit <- rep(0, pmax)
pen <- rep(0, pmax)
lambda <- 2

# Model selection procedure
for (p in seq_len(pmax)) {
  model <- lm(y ~ poly(x, p))
```

```
  crit[p] <- -sum(predict(model)^2)
  pen[p] <- lambda * (p + 1)
}

# Select a model
phat <- which.min(crit + pen)

# Plot criterion and penalty
plot(crit+pen, type="l", col="red", ylab="Penalized criterion")
abline(v=phat, lty=2)

# Plot the estimator
plot(x, y, col="grey", pch=4)
points(x, s, type="l")
model_phat <- lm(y ~ poly(x, phat))
points(x, predict(model_phat), type="l", col="red")
```

- Comment the final result.
- Change the value of `lambda` to understand the key role of the tuning parameter in such a procedure.

3 — Classification and clustering

3.1 Introduction

In this chapter, we are interested in two statistical problems known as **classification** and **clustering**. Although the statements of these problems are similar, they can definitively not be tackled in the same way. There is a huge literature about procedures to solve such problems and it would be impossible to give here a global overview. We introduce some popular approaches but the interested reader is invited to refer to third-party books.

Let us first state the classification problem. Actually, we already did it in Section 1.5 when we introduced the MDA procedure as an example. Indeed, we assume that we have at our disposal p real variables $x^1,\dots,x^p$ which we observe n times with some associated weights $w_1,\dots,w_n > 0$. Moreover, for each individual, we also observe a label t which can take m distinct values in the **known** set $\mathbb{T} = \{\tau_1,\dots,\tau_m\}$. Thus, our data set is given by the observations of $(x_1,t_1),\dots,(x_n,t_n) \in \mathbb{R}^p \times \mathbb{T}$. The goal is now to provide a **classification rule** solely based on the data to decide what label among $\mathbb{T}$ should be given to an arbitrary element $x \in \mathbb{R}^p$.

The clustering problem statement is similar except that the label values are **unknown**. Actually, we do not even assume that such a set exists. Based on the n observations of the p real variables $x^1,\dots,x^p$, we want to provide a rule to cluster the data in a consistent way. In particular, we have to build an acceptable set for the hypothetical labels. Clustering is more difficult than classification mainly because the problem is not well posed. Indeed, for classification, we will be able to consider **classification error** as a criterion because we know that there exists some underlying clustering. However, in clustering problem, such a criterion is no more available and we do not have any straightforward way to quantify the quality of a procedure.

3.2 Classification

Let us first remind how we tackled the classification problem through MDA. We have computed the PCA of the triplet (X_b, W, Σ^{-1}) and obtained the coordinates of the means of each cluster in some principal space E_d. The easy way to derive a classification rule is then, for any $x \in \mathbb{R}^p$, to consider the projection of x onto E_d and to choose the cluster that has the closest mean

to this projection. There exist more subtle methods but this approach was sufficient for our purpose at the end of the practicals. An important drawback of MDA is due to the maximal nontrivial dimension of the principal space, namely $\kappa = \min\{m-1, p\}$. Indeed, if you handle data with labels that can only take two distinct values, *i.e.* $m = 2$, this dimension is capped by $m - 1 = 1$. In other words, whatever the number p of variables you have, you can not get more than one dimension for the classification rule.

The other popular approach that we introduce in this section does not suffer from this dimension problem and is based on binary trees. Tree methods are often appreciated in practice because tree structures lend themself well to algorithmic approaches. Our goal is here to deal with classification, then we speak about **classification tree** but similar ways of proceeding can be used to tackle regression problems for which we speak about **regression tree**. Actually, these ways are so similar than we use the same object called **CART** for 'Classification And Regression Tree'.

The CART method consists in successive conditions of the form $x^j \leqslant s$ for some $j \in \{1, \dots, p\}$ and some threshold $s \in \mathbb{R}$. We split observations into two groups depending on whether the condition is true or not. Then, we do it again with a new condition of the same form and so on. Because each condition leads to two possibilities, such a procedure can be seen as a binary tree. To properly define the method, we have to explain how each condition is provided, how we give a label to a group of data and when we stop the process.

Let us start with the easiest stage, namely the choice of a label for a given group of data. To this end, we first assume that we have already built some partition of $\mathbb{R}^p$ into M regions $R_1, \dots, R_M$. The question is then to give a label among $\mathbb{T}$ to each region, *i.e.* we look for a function $\lambda : \{1, \dots, M\} \to \{1, \dots, m\}$ such that, for any $x \in \mathbb{R}^p$, we predict the label $\tau_{\lambda(k)}$ if $x \in R_k$. A basic way to construct such a function λ is to consider the **frequencies** of each label in each region. Namely, for any $j \in \{1, \dots, M\}$ and $k \in \{1, \dots, m\}$, the frequency of the label τ_k in the region R_j is given by

$$p_{j,k} = \frac{1}{\bar{w}_j} \sum_{i \text{ s.t. } x_i \in R_j} w_i \mathbf{1}_{\{t_i = \tau_k\}}$$

where $\bar{w}_j$ is the sum of the weigths of individuals which belong to R_j. Then, we simply attribute to the region R_j the most represented label, *i.e.* for any $j \in \{1, \dots, M\}$,

$$\lambda(j) = \underset{k \in \{1, \dots, m\}}{\operatorname{argmax}} \; p_{j,k}.$$

Note that, for any $j \in \{1, \dots, M\}$, the vector $p_j = (p_{j,1}, \dots, p_{j,m})^\intercal \in [0,1]^m$ induces a discrete probability distribution on the labels in region R_j. This consideration allows us to quantify the **purity** of such a distribution with the **Gini index** of p_j,

$$\mathscr{G}_j = \sum_{k=1}^{m} p_{j,k}(1 - p_{j,k}).$$

This quantity is close to zero only if a region is pure in the sense that there mainly is only one label in this region. From a probabilistic point of view, the Gini index $\mathscr{G}_j$ is the error rate obtained by choosing the label of R_j at random with the probability distribution given by p_j.

Now that we know how to give a label to each region of an existing partition of $\mathbb{R}^p$, we have to see how to build such a partition. Because each region is the result of a condition of the form $x^j \leqslant s$ as mentioned above, this question is related to the way to provide the successive conditions. Actually, there exist "optimal" sequences of such conditions to fit the data but computing one of this sequence is generally infeasible in practice (this problem is known to be a NP-complete problem). Hence, the CART idea consists in a **greedy algorithm**, *i.e.* a step by step algorithm which aims to provide a global solution to the problem by simply solving it at each stage without considering the whole question. Let us describe the first step: for any $j \in \{1,\dots,p\}$ and $s \in \mathbb{R}$, we define the pair of half-planes given by the condition $x^j \leqslant s$, namely

$$R_1(j,s) = \left\{x = (x^1,\dots,x^p)^\intercal \in \mathbb{R}^p \text{ such that } x^j \leqslant s\right\}$$

and

$$R_2(j,s) = \left\{x = (x^1,\dots,x^p)^\intercal \in \mathbb{R}^p \text{ such that } x^j > s\right\}.$$

These two regions form a partition of $\mathbb{R}^p$ and we can compute the vectors of probability $p_1(j,s)$ and $p_2(j,s)$ as above for $R_1(j,s)$ and $R_2(j,s)$, respectively. We also can compute the associated weights $\bar{w}_1(j,s)$ and $\bar{w}_2(j,s)$. To quantify the purity of the split given by the pair (j,s), we consider the mean Gini index, namely

$$\bar{\mathscr{G}}(j,s) = \bar{w}_1(j,s)\mathscr{G}_1(j,s) + \bar{w}_2(j,s)\mathscr{G}_2(j,s)$$

where $\mathscr{G}_1(j,s)$ and $\mathscr{G}_2(j,s)$ are the Gini indices for $p_1(j,s)$ and $p_2(j,s)$, respectively. Then, we pick $(j^*,s^*) \in \{1,\dots,p\} \times \mathbb{R}$ to minimize $(j,s) \mapsto \bar{\mathscr{G}}(j,s)$. The sequel of the algorithm then consists in repeating the same procedure in each region.

The last point to clarify is how to stop growing the tree. Indeed, we could pursue the procedure till each region becomes perfectly pure. However, such a large tree that perfectly fit to the data could be to specific and induce overfitting phenomenon. We retrieve here a problem seen in the previous chapter: we need some trade-off between the ability to fit the data and the complexity of the tree. During the eighties, Breiman proposed the **pruning** idea to this end. Let us consider some too large tree T_0 with a number of leaf M_{T_0}. The goal is to provide a way to prune this tree. Let $T \subset T_0$ be a subtree with $1 \leqslant M_T \leqslant M_{T_0}$ leaves, T gives regions $R_1,\dots,R_{M_T}$ and we can compute the associated weights $\bar{w}_1,\dots,\bar{w}_{M_T}$ and Gini indices $\mathscr{G}_1(T),\dots,\mathscr{G}_{M_T}(T)$. Then, to estimate the announced trade-off, we consider the subtree to minimize the penalized criterion

$$\gamma_\lambda(T) = \sum_{k=1}^{M_T} \bar{w}_k \mathscr{G}_k(T) + \lambda M_T$$

where $\lambda \geqslant 0$ is a tuning parameter. The first term in the criterion is related to the purity of the regions and the second one is a penalty which punishes the subtrees with too much leaves.

3.3 Clustering

In contrast with the previous section, we now consider only data without label. Thus, any procedure based on the label observations is unavailable and we have to provide clusters in other way. Of course, fully exploring the set of the partitions of $\{1,\dots,n\}$ is not a solution because of the size of such set, even for small value of n (such an approach is also known

to be a NP-complete problem). In order to make emerge some meaningful clusters, the basic idea consists in grouping close observations in a sense of some **similarity** measurement $d : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_+$ which satisfies few properties, namely

- for any $x, x' \in \mathbb{R}^p$, $d(x,x') = d(x',x)$,
- for any $x \in \mathbb{R}^p$, $d(x,x) = 0$.

Note that a mathematical distance is a similarity but the converse is not.

The first popular method we present needs a number K of clusters, a priori, and is known as **K-means**. Such an assumption could be considered as a drawback because we don't even know if cluster existence is justifiable. However, in practice, it is often acceptable to look for the "best" partition of the data formed by K clusters $C_1, \ldots, C_K$. The procedure is iterative and actually quite simple: we consider K special points called **centroids** $c_1, \ldots, c_k \in \mathbb{R}^p$ and each observation is put in the cluster associated to its closest centroid. Some dispersal criterion is then computed and we iterate while this criterion is decreasing. Let us properly state this method:

1. pick K initial centroids $c_1, \ldots, c_k$ at random among the observations,
2. get the K clusters by putting each observation in the cluster associated to the closest centroid,
3. the means of each cluster become the new centroids,
4. compute the inertia within criterion
$$\sum_{k=1}^{K} \frac{1}{\bar{w}_k} \sum_{i \in C_k} w_i d^2(x_i, c_k),$$
5. if the criterion decreases, go to step 2, stop otherwise.

An important remark is to notice that such a procedure is not guaranteed to reach a global minimum of the criterion. Indeed, according to the choice of the similarity d, there is no reason to obtain a convex criterion and the procedure may converge only towards some local minimum. There exist a lot of variants of K-means procedure which we do not list in this document. The interested reader will easily get leads to follow in the literature.

An other famous way to tackle clustering problem is called **Hierarchical Clustering Analysis** (HCA). This approach does not need to know the number of clusters in advance and is based on successive aggregations of similar groups of individuals in the sense of the similarity d. At each step, the idea is to merge the two current clusters which are the most similar. To measure this similarity between two groups G_1 and G_2 of individuals, we have several options:

- **single linkage**:
$$d(G_1, G_2) = \min \left\{ d(x,x') \text{ with } x \in G_1 \text{ and } x' \in G_2 \right\},$$

- **complete linkage**:

$$d(G_1, G_2) = \max \left\{ d(x, x') \text{ with } x \in G_1 \text{ and } x' \in G_2 \right\},$$

- **average linkage**: denoting by $|G_1|$ and $|G_2|$ the respective sizes of G_1 and G_2,

$$d(G_1, G_2) = \frac{1}{|G_1| \times |G_2|} \sum_{(x,x') \in G_1 \times G_2} d(x, x'),$$

- **Ward distance**: denoting by g_1 and g_2 the means of observations in G_1 and G_2 and by $\bar{w}_1$ and $\bar{w}_2$ their associated weights, respectively,

$$d(G_1, G_2) = \frac{\bar{w}_1 \times \bar{w}_2}{\bar{w}_1 + \bar{w}_2} d(g_1, g_2).$$

In practice, the common choices are the average linkage and the Ward distance. Note, however, that to use the Ward distance, we have to be able to explicitly compute the means g_1 and g_2. Indeed, this is not always possible when we only have at our disposal the similarities between individuals but not their coordinates. Thus, the HCA method proceeds as follows:

1. consider n trivial clusters of one individual,
2. merge the two closest clusters,
3. update the similarities with the new cluster,
4. if there is more than one cluster, go to step 2, stop otherwise.

When the procedure ends, we obtain a path of clusterings between the two extreme situations of n clusters of one elements and one cluster of n elements. This path can be represented as a tree called a **dendrogram** which we have to cut at some height in order to get a candidate for our clustering problem. Cutting the dendrogram is again a trade-off problem between adequacy of data and complexity of the solution, *i.e.* here, it is measured by the number of clusters.

Pratically speaking, K-means and HCA have their strengths and weaknesses. For K-means, the knowledge of K is debatable but the provided clustering is known to be stable in the sense that a new observation will not be able to rush the clusters. With HAC, there is no need for specific knowledge a priori but the computed solution needs some trade-off and suffer from instability because a new observation can widely modify the clusters. To take advantage of both methods, a popular way consists in starting with a HCA with some reasonable criterion to cut the tree and obtain a number of clusters K. Then, we can use this clustering as the initial clusters of a K-means procedure which will stabilize it.

Practicals 4 : CART and clustering

Data are available from the website of the author:

https://www.math.univ-toulouse.fr/~xgendre

Wines classification

To illustrate the classification method based on CART, we consider a data set of $p = 12$ real variables given by physicochemical properties and quality evaluation of $n = 6497$ red and white wines. The label is the wine color and we aim to predict it from the real variables. Let us start by loading this data set.

```
wines_data <- read.csv("wines.csv")
```

Though this is a good algorithmic exercise, we are not going to implement the CART algorithm by ourselves. For the sake of simplicity, we use the package `rpart` provided by default with R.

```
library(rpart)
```

The main function of `rpart` is wisely called `rpart` and works similarly as the function `lm`. In particular, to say which column is the label to explain in the data set and with respect to which variables this procedure should perform, we use a formula. Moreover, we specify that we handle trees to classify with the option `method`.

```
f <- paste("category ~ ",
  paste(colnames(wines_data)[1:12], collapse = " + ")
)
```

```
tree <- rpart(formula=f, data=wines_data, method="class")
```

- Print and understand the content of the variable `f`.
- Print the R object `tree`. Do you spot the tree structure based on successive conditions?

An advantage of tree models is the facility to display them. Indeed, the reading of a tree from top to bottom, picking some branches according to the results of the conditions is quite natural and easy to represent.

```
plot(tree, uniform = TRUE)
text(tree)
```

- Read the manual page for `text.rpart`. Plot again the tree and set argument `fancy` to `TRUE` in the call to `text`. Is the representation easier to read?
- If some wine has a total sulfur dioxide measurement lower than 67.5 and a chlorides measurement larger than 0.0465, what is its predicted color?
- Set the argument `all` to `TRUE` in the call to funtion `text`. What is the new information which appears?
- Set the argument `use.n` to `TRUE` in the call to funtion `text`. For our previous measurements example of total sulfur dioxide and chlorides, what is the number of misclassified wines?
- Is there any pure leaf in this tree?

The implementation of CART algorithm in the package `rpart` provides a way to prune the tree through some complexity parameter `cp` in a similar way as what we have seen in the lecture. To use this parameter, let us rebuild the tree with the `control` argument.

```
tree <- rpart(formula=f, data=wines_data, method="class",
  control=rpart.control(cp=0.1))
plot(tree, uniform = TRUE)
text(tree, use.n=TRUE)
```

- With the help of the manual page of `rpart.control`, explain what is the role of `cp`.
- Change the value of `cp` to `0.01`, `0.001` and `0.0001`. What do you observe?
- What is the value of `cp` that misclassifies the lowest amount of data? Do you think that such a value for `cp` is a good choice? Why?

Vietnamese cities clustering

We have seen two clustering procedures during the lecture and we propose here to apply them to $n = 60$ Vietnamese cities for which we have at our disposal the road distances between each pair of cities. This data set is loaded as follows, the GPS data are put aside for graphics purposes and the distances are converted to some suitable R object.

```
vietnam_cities_data <- read.csv("vietnam_cities.csv")
gps_data <- vietnam_cities_data[,1:2]
X <- as.dist(vietnam_cities_data[,3:62])
```

The easiest method to apply is K-means because it only needs the distance matrix and the number of desired clusters.

```
k4 <- kmeans(X, 4) # With 4 clusters
```

- Print the content of the R object `k4` and locate the value obtained for the variance criterion.
- Run again the procedure and look at the criterion value again. What do you note? Explain this fact.

We now propose to display the obtained clustering on a Vietnam map. To this end, we need to install and to load the package `mapdata`.

```
install.packages("mapdata") # In Vietnam, install from Thailand
library(mapdata)
```

To reveal the clusters on the map, we plot the cities with distinct colors according to GPS coordinates.

```
map("worldHires", "Vietnam")
points(gps_data$longitude, gps_data$latitude,
  pch=16, col=k4$cluster)
```

- Comment the result. Do the clusters seem reasonable with respect to the Vietnamese road network?
- Run again the 4-means procedure and observe again the phenomenon mention earlier.
- Run the K-means procedure with other values for K such as 2, 5, 9, ... Comment what you obtain.

The second approach we studied for clustering was based on Hierarchical Clustering (HCA). R supplies the function `hclust` to apply such a procedure.

```
help(hclust)

hca <- hclust(X) # Compute HCA
plot(hca) # View the dendrogram
```

To see the sequence of values of the criterion at each split, the R object `hca` provides a vector `hca$height` which we just have to reverse.

```
plot(rev(hca$height), type='b', ylab="Clustering height")
```

- Does this representation recall an other one previously seen? With respect to this graphics similarity, say roughly how many indices we could keep.

The previous graphics representation obviously depends on the way we merge the clusters. The default way to measure a similarity between two clusters is complete linkage. Let us experiment the other ways.

```
# Two columned graphics and parameters backup
old_par <- par(mfrow=c(1,2))

# Single linkage
hca <- hclust(X, method="single")
plot(hca)
plot(rev(hca$height), type='b', ylab="Clustering height")

# Complete linkage (default)
hca <- hclust(X, method="complete")
plot(hca)
plot(rev(hca$height), type='b', ylab="Clustering height")

# Average linkage
hca <- hclust(X, method="average")
plot(hca)
plot(rev(hca$height), type='b', ylab="Clustering height")

# Ward distance
hca <- hclust(X, method="ward.D2")
plot(hca)
plot(rev(hca$height), type='b', ylab="Clustering height")

# Restore graphics parameters
par(old_par)
```

- Compare the differents methods. In particular, understand what it means when the curve of heights decreases more or less faster.

We have now to deduce a proper clustering from these dendrogram. The principle consists in cutting the tree with `cutree` whether at some specified height or to get a fixed number of clusters.

```
help(cutree)

# HCA with complete linkage
hca <- hclust(X)

# Cut to get 4 cluters
hca_cluster <- cutree(hca, k=4)
map("worldHires", "Vietnam")
points(gps_data$longitude, gps_data$latitude,
  pch=16, col=hca_cluster)

# Choose some height ...
old_par <- par(mfrow=c(1,2))
plot(rev(hca$height), type='b', ylab="Clustering height")
abline(h=500, col="red")
plot(hca)
abline(h=500, col="red")
par(old_par)

# ... and get the clusters
hca_cluster <- cutree(hca, h=500)
map("worldHires", "Vietnam")
points(gps_data$longitude, gps_data$latitude,
  pch=16, col=hca_cluster)
```

- When you fix the number of clusters, compare the result with you obtained with K-means.
- From the graphics with a red horizontal line, how many clusters do you obtain with a cut at height 500?
- Comment the clusters obtained by cutting the tree at a given height.
- Apply the same steps for clustering with HCA computed with other linkage methods.

Stabilizing clustering

As we discussed during the lecture, K-means and HCA have their own strengths and weaknesses. Combining both methods permits us to set a clustering procedure that take advantage

from each method by providing an automated choice for K based on HCA and a stable clustering with well initialized K-means.

First, let us consider the HCA of the Vietnamese cities GPS positions data set with complete linkage.

```
hca <- hclust(dist(gps_data))
```

Heights of successive agglomerations corresponds to the gaps between two steps of the procedure. Because the criterion of HCA is related to some decreasing "between dispersal" measurements, we can see the agglomerations similarly to principal components in the sense that the more agglomerations we keep, the more "between dispersal" we capture. In other words, the decreasing values of the heights can be interpreted as the lack of fit according to this "between criterion". We choose to handle its normalized value as criterion to fit here.

```
crit <- rev(hca$height) / sum(hca$height)
plot(crit, type="b",
  xlab="Number of agglomerations",
  ylab="Between dispersal criterion")
```

Of course, taking into account all the agglomerations leads to overfitting. Thus, we are interested to find some trade-off between this capacity to fit the data and a reasonalbe number of cluster. To this end, we introduce a penalty term proportional to this number as well normalized. Then, we pick K to minimize the obtained penalized criterion.

```
lambda <- 2
pen <- lambda * 1:length(hca$height) / length(hca$height)
plot(crit + pen, type="b",
  xlab="Number of agglomerations",
  ylab="Penalized criterion")

# Minimize the penalized criterion
K <- which.min(crit + pen)
cat("We selected", K, "clusters.\n")
```

Note that such an approach is comparable to select a height to cut the dendrogram with respect to some criterion penalized by the number of clusters such a cut gives. An important drawback of the clustering into K clusters provided by previous HCA is its instability. The idea consists in considering this proxy clustering to initialize a K-means procedure in order to stabilize it.

```
# Proxy clustering
proxy_cluster <- cutree(hca, k=K)
centroids <- matrix(0, nrow=K, ncol=2)
for (i in 1:K) {
  centroids[i,] <- colMeans(gps_data[proxy_cluster==i,])
}

# Run kmeans initialized with proxy clustering
final <- kmeans(gps_data, centroids)

# Print proxy and final clusterings
old_par <- par(mfrow=c(1,2))
map("worldHires", "Vietnam")
points(gps_data$longitude, gps_data$latitude,
  pch=16, col=proxy_cluster)
map("worldHires", "Vietnam")
points(gps_data$longitude, gps_data$latitude,
  pch=16, col=final$cluster)
par(old_par)
```

- Locate the cities that have been moved from a cluster to an other.

4 — Cross-validation

4.1 Introduction

In Chapter 2, we used the quantity $R_n(\hat{\beta})$ to quantify the empirical quality of a linear model and discussed its expected version $\mathbb{E}[R_n(\hat{\beta})]$. This expected value raised the dilemma between capacity of approximation and complexity of the model. This duality illustrated how to give a sense to the expected error of the model if we apply it to predict on new data. The main idea to find a trade-off was to mimic the behavior of the expectation $\mathbb{E}[R_n(\hat{\beta})]$ through some penalized criterion with a penalty similar to the variance term, up to a multiplicative tuning parameter $\lambda \geqslant 0$. As we have seen during the practicals, the choice of λ is important to get a competitive procedure. However, from a general point of view, there is no straightforward justification to choose λ.

Tuning parameters are the topic of this last chapter and we propose several popular approaches to choose them. Although we often refer to the multiplicative factor in penalties, the procedures described hereafter can be used with a lot of common frameworks such as statistical testing, for example. The main idea consists in putting aside a part of the initial observations, called **testing observations**, and to only handle our models using the remaining observations, called **training observations**. In such a way, we can apply our procedure that depends on some parameter λ with various values for it and validate with the testing observations what is the candidate that gives the better results. In this way, even if the expected value $\mathbb{E}[R_n(\hat{\beta})]$ is unavailable, we can evaluate the performances of the procedure on data outside of the training observations. Such approaches are known as **cross-validation** procedures.

4.2 Validation set

To give sense to the concepts discussed above, a simple way consists in picking the training observations at random among the initial observations. The proportion $\alpha \in (0,1)$ of retained data can vary from one situation to an other but is often around 70% or 80% in practice. Based on this **training set**, for any value $\lambda \geqslant 0$, we can compute an estimator $\hat{h}_\lambda$ of the quantity of interest and evaluate its quality on the remaining data in the **validation set**. We then keep the cross-validated value λ^* which gives the better result.

This approach is easy to use in practice but it suffers from two drawbacks:

1. repeating the procedure with other random picked values leads to high variability in the measurements of the quality with the corresponding validation tests,

2. because we do not use all the data to fit the model, we perform theoretically less well than with the whole data set.

4.3 Leave-One-Out

The **leave-one-out** approach is a refinement of the validation set which tries to tackle its main drawbacks. As previously, we consider a training set and a validation set but we only put one observation in the validation set. Let us assume that this validation observation is the first one among n, we then train the procedure for various values for λ with the $n-1$ remaining observations and we call $\hat{h}_1(\lambda)$ the obtained estimators. Since only one observation has been put aside, the estimation of the quality of $\hat{h}_1(\lambda)$ is poorly done by comparing its prediction on this singleton and this value is denoted by $e_1(\lambda)$.

Even on a single point, the estimation of the quality of $\hat{h}_1(\lambda)$ remains an unbiased estimator of the expected value. Thus, we can proceed in the same way with only the second observation, then only the third and so on ... The consecutive estimations of the quality supply n values $e_1(\lambda),\dots,e_n(\lambda)$. The leave-one-out procedure consider the average of these estimates to calibrate a value λ^* for the parameter,

$$\mathrm{LOO}_n(\lambda) = \frac{1}{n}\sum_{i=1}^{n} e_i(\lambda).$$

The leave-one-out approach offers several advantages. First, it is less biased than the validation test because it uses quite all the observations at each step when the validation set approach is limited by the proportion α. Moreover, the leave-one-out approach does not need to be repeated to stabilize the result. Avoiding the random repetitions we had to do with validation set implies that the leave-one-out always leads to the same result.

However, the leave-one-out approach can be complicated to run when n is large. Indeed, too get $\mathrm{LOO}_n(\lambda)$, we have to apply n times our statistical procedure. If a computation of this procedure takes a bit of time, n times can be very expensive for large data set.

4.4 k-Fold

The last approach that we consider in this chapter is an alternative to leave-one-out in order to address its computability drawback. The k-**fold** cross-validation procedure is based on a random partition of the initial set of observations into k groups, called **folds**, of approximately equal size. Then, we proceed in a manner similar to what we did for leave-one-out. First, for some λ, we treat the first fold as a validation set on which we evaluate the quality of an estimator $\hat{h}_1(\lambda)$ built from the $k-1$ remaining folds. This quality estimation is denoted by $e_1(\lambda)$. The second fold gives a value $e_2(\lambda)$ in the same way and so on. To calibrate a value

λ^*, we then average these k estimates $e_1(\lambda), \ldots, e_k(\lambda)$ with

$$\mathrm{KF}_n(\lambda) = \frac{1}{k} \sum_{i=1}^{k} e_i(\lambda).$$

Obviously, the leave-one-out approach is nothing else than a k-fold with $k = n$. In practice, we consider k around 10 to cross-validate a tuning parameter. The main advantage of $k = 10$ rather than $k = n$ is that we only have to run the statistical procedure ten times instead of n. Thus, tuning a parameter with a k-fold approach is less expensive from a computability point of view, mainly when the data set becomes large.

Practicals 5 : Cross-validation

Introduction

As in Practicals 3, we consider here a regression framework and a model selection approach. Then, no real data set are used and we only handle simulated data. Our goal is to illustrate the cross-validation procedures introduced in the lecture to choose a value for the multiplicative factor $\lambda \geqslant 0$ in the penalty.

The regression framework we manipulate in the next sections is based on n observations $(x_1, y_1), \ldots, (x_n, y_n) \in [0,1] \times \mathbb{R}$ where the design points $x_1, \ldots, x_n$ are uniformly distributed in $[0,1]$.

```
n <- 256
x <- sort(runif(n))
s <- 2*cos(2*pi*x) + 3*sin(6*pi*x)
y <- s + rnorm(n)

plot(x, y, pch=4, col="grey")
points(x, s, type="l", lty=3)
```

Validation set

Let us start with the validation set approach for a proportion $\alpha = 70\%$ of initial observation in the training set.

```
alpha <- 0.7

# Training set
train <- sample.int(n, round(alpha * n))
x_train <- x[train]
```

```
y_train <- y[train]

# Validation set
test <- (1:n)[-train]
x_test <- x[test]
y_test <- y[test]
```

To estimate the regression function, we use polynomial models with degree between 1 and 25 and a selection model procedure. Let $p \in \{1,\dots,25\}$, we denote by H_p the projection matrix and by $\hat{\beta}_p$ the coefficients for polynomial model of degree p. For any $\lambda \geqslant 0$, we pick $p^*(\lambda) \in \{1,\dots,25\}$ to minimize the penalized criterion

$$p^*(\lambda) = \underset{p\in\{1,\dots,25\}}{\operatorname{argmin}} \left\{ -\|H_p y_{\text{train}}\|^2 + \lambda(p+1) \right\}.$$

Because our goal is to choose an appropriate value for λ, we consider several values between 0 and 10.

```
pmax <- 25
lambda <- seq(0, 10, length.out=100)

# Initialization
crit <- rep(0, pmax)
pen <- 2:(pmax+1)

# Model selection procedure on training set
for (p in seq_len(pmax)) {
  model <- lm(y_train ~ poly(x_train, p))
  crit[p] <- -sum(predict(model)^2)
}
p_star <- sapply(lambda, function(l) { which.min(crit + l*pen) })
```

- Explain what does the vector `p_star` contain.

We now evaluate the selected models with the validation set. To this end, for any $\lambda \geqslant 0$, we consider the predicted values $X_{\text{test}}\hat{\beta}_{p^*(\lambda)}$ and we compare them to y_{test},

$$e(\lambda) = \|X_{\text{test}}\hat{\beta}_{p^*(\lambda)} - y_{\text{test}}\|^2.$$

Note that the predictor $\hat{\beta}_{p^*(\lambda)}$ is solely based on data in training set.

```
# Compute validation errors
e <- sapply(p_star,
  function(p) {
    model <- lm(y_train ~ poly(x_train, p))
    # Prediction command is misleading!
    y_pred <- predict(model, data.frame(x_train=x_test))
    return(sum((y_pred - y_test)^2))
  }
)

# Get parameter of minimal error
lambda_star <- lambda[which.min(e)]
cat("Minimal error for lambda =", lambda_star, "\n")

# Plot the result
plot(lambda, e, type="l", col="red", lwd=2)
abline(v=lambda_star, lty=2)

# Cross-validated estimator
cat("Cross-validated dimension is", p_star[which.min(e)], "\n")
model_star <- lm(y ~ poly(x, p_star[which.min(e)]))
plot(x, y, pch=4, col="grey")
points(x, s, type="l", lty=3)
points(x, predict(model_star), type="l", col="red")
```

- Note that it does not remain any tuning parameter in the procedure.
- Comment the graphics of validation error with respect to λ.

To illustrate how the validation set is to the initial random choice, we compute the validation error several times.

```
n_try <- 16
e <- matrix(0, nrow=length(lambda), ncol=n_try)

for (i_try in seq_len(n_try)) {
  train <- sample.int(n, round(alpha * n))
  crit <- rep(0, pmax)
  for (p in seq_len(pmax)) {
    model <- lm(y[train] ~ poly(x[train], p))
    crit[p] <- -sum(predict(model)^2)
    p_star <- sapply(lambda, function(l) which.min(crit + l*pen))
  }
```

```
  e[,i_try] <- sapply(p_star,
    function(p) {
      design <- x[train]
      model <- lm(y[train] ~ poly(design, p))
      y_pred <- predict(model, data.frame(design=x[-train]))
      return(sum((y_pred - y[-train])^2))
    }
  )
}

matplot(e, type="l", lwd=2, col=rainbow(n_try))
```

- Comment the variety of the results. Does it seem stable?

Leave-One-Out

To address the drawbacks of validation set, we have introduced the leave-one-out approach. The concept is similar except that we consider validation sets with one observation only.

```
e <- matrix(0, nrow=length(lambda), ncol=n)

for (i in seq_len(n)) {
  # Training set is no more random
  train <- (1:n)[-i]

  crit <- rep(0, pmax)
  for (p in seq_len(pmax)) {
    model <- lm(y[train] ~ poly(x[train], p))
    crit[p] <- -sum(predict(model)^2)
    p_star <- sapply(lambda, function(l) which.min(crit + l*pen))
  }

  e[,i] <- sapply(p_star,
    function(p) {
      design <- x[train]
      model <- lm(y[train] ~ poly(design, p))

      # Quality estimate is easier
      y_pred <- predict(model, data.frame(design=x[i]))
      return(sum((y_pred - y[i])^2))
    }
  )
}
```

```
# Average the estimates to get cross-validates parameter
Loo <- rowMeans(e)
lambda_star <- lambda[which.min(Loo)]
cat("Minimal error for lambda =", lambda_star, "\n")

# Plot the result
plot(lambda, Loo, type="l", lwd=2, col="orange")
abline(v=lambda_star, lty=2)
```

- Take note of the long time spent to perform the procedure.
- Discuss the final plot with respect to the instability of the validation approach.
- Compute the cross-validated estimator.

k-Fold

Finally, we consider the leave-one-out alternative supplied by the k-fold approach. We divide the initial data set into $k = 8$ folds to make division easier. Thus, we successively run validation set procedure with each fold as validation set.

```
k <- 8
fold <- matrix(sample.int(n), nrow=k)
e <- matrix(0, nrow=length(lambda), ncol=k)

for (i in seq_len(k)) {
  # Training set of k-1 folds
  train <- as.vector(fold[-i,])

  crit <- rep(0, pmax)
  for (p in seq_len(pmax)) {
    model <- lm(y[train] ~ poly(x[train], p))
    crit[p] <- -sum(predict(model)^2)
    p_star <- sapply(lambda, function(l) which.min(crit + l*pen))
  }

  e[,i] <- sapply(p_star,
    function(p) {
      design <- x[train]
      model <- lm(y[train] ~ poly(design, p))

      # Quality estimate with the fold
      y_pred <- predict(model, data.frame(design=x[-train]))
      return(sum((y_pred - y[i])^2))
```

```
    }
  )
}

# Average the estimates to get cross-validates parameter
KF <- rowMeans(e)
lambda_star <- lambda[which.min(KF)]
cat("Minimal error for lambda =", lambda_star, "\n")

# Plot the result
plot(lambda, Loo, type="l", lwd=2, col="orange")
abline(v=lambda_star, lty=2)
```

- Compare the time spent to perform the procedure with the leave-one-out approach.
- Discuss the final plot with respect to the one obtained with leave-one-out method.
- Compute the cross-validated estimator.